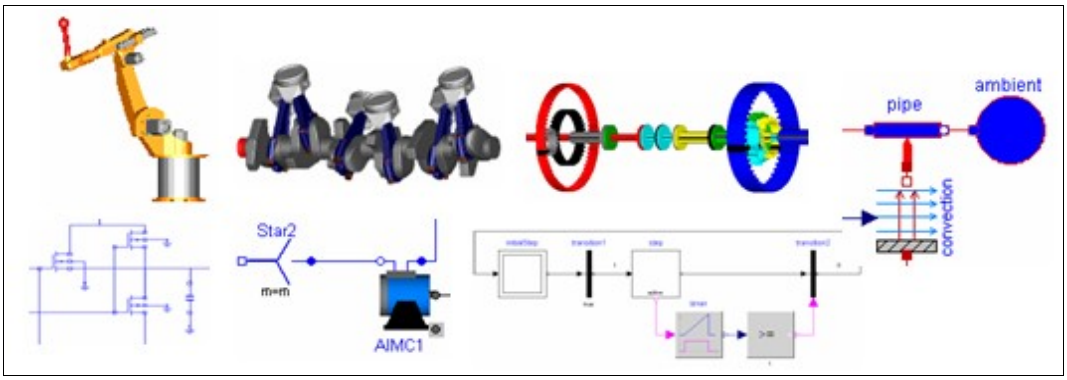
Modelica Standard Library

Version 2.2.2

August 2007

Tutorial and Reference



Modelica Association
http://www.Modelica.org

Copyright © 1998-2007, Modelica Association (http://www.Modelica.org)

Modelica® is a registered trademark of the Modelica Association

This manual can be freely distributed according to the Modelica License (which holds for the source code of the Modelica Standard Library, including its documentation):

Redistribution and use in source and binary forms, with or without modification are permitted, provided that the following conditions are met:

1. The author and copyright notices in the source files, these license conditions and the disclaimer below are (a) retained and (b) reproduced in the documentation provided with the distribution.
2. Modifications of the original source files are allowed, provided that a prominent notice is inserted in each changed file and the accompanying documentation, stating how and when the file was modified, and provided that the conditions under (1) are met.
3. It is not allowed to charge a fee for the original version or a modified version of the software, besides a reasonable fee for distribution and support. Distribution in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution is permitted, provided that it is not advertised as a product of your own.

Disclaimer:

The software (sources, binaries, etc.) in their original or in a modified form are provided "as is" and the copyright holders assume no responsibility for its contents what so ever. Any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the copyright holders, or any party who modify and/or redistribute the package, be liable for any direct, indirect, incidental, special, exemplary, or consequential damages, arising in any way out of the use of this software, even if advised of the possibility of such damage.

Modelica is a freely available, object-oriented language for modeling of large, complex, and heterogeneous physical systems. It is suited for multi-domain modeling, for example, mechatronic models in robotics, automotive and aerospace applications involving mechanical, electrical, hydraulic and control subsystems, process oriented applications and generation and distribution of electric power. Models in Modelica are mathematically described by differential, algebraic and discrete equations. No particular variable needs to be solved for manually. A Modelica tool will have enough information to decide that automatically. Modelica is designed such that available, specialized algorithms can be utilized to enable efficient handling of large models having more than one hundred thousand equations. Modelica is suited and used for hardware-in-the-loop simulations and for embedded control systems.

Modelica is developed by the Modelica Association, a non-profit organization with seat in Linköping, Sweden. The Modelica Association also develops the **free Modelica Standard Library** containing model components in many domains that are based on standardized interface definitions. This manual was automatically produced from the documentation included in the Modelica Standard Library.

The source code of the **Modelica Standard Library**, as well as this manual, is available at http://www.Modelica.org/libraries/Modelica

Links to other **free** and **commercial Modelica libraries** (utilizing the Modelica Standard Library) are available at http://www.Modelica.org/libraries.

Links and short descriptions of **Modelica modeling and simulation environments** are available at http://www.Modelica.org/tools.

Contents

Modelica....41
Modelica.UsersGuide....42
Modelica.UsersGuide.Overview....42
Modelica.UsersGuide.Connectors....44
Modelica.UsersGuide.Conventions....46
Modelica.UsersGuide.ReleaseNotes....47
Modelica.UsersGuide.ReleaseNotes.Version_2_2_2....48
Modelica.UsersGuide.ReleaseNotes.Version_2_2_1....57
Modelica.UsersGuide.ReleaseNotes.Version_2_2....62
Modelica.UsersGuide.ReleaseNotes.Version_2_1....63
Modelica.UsersGuide.ReleaseNotes.Version_1_6....66
Modelica.UsersGuide.ReleaseNotes.Version_1_5....66
Modelica.UsersGuide.ReleaseNotes.Version_1_4....71
Modelica.UsersGuide.Contact....72
Modelica.UsersGuide.ModelicaLicense....73
Modelica.Blocks....73
Modelica.Blocks.Examples....74
Modelica.Blocks.Examples.PID_Controller....75
Modelica.Blocks.Examples.ShowLogicalSources....76
Modelica.Blocks.Examples.LogicalNetwork1....77
Modelica.Blocks.Examples.BusUsage....78
Modelica.Blocks.Examples.BusUsage_Utilities....81
Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces....81
Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.ControlBus....82
Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.SubControlBus....82
Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.InternalConnectors....82
Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.InternalConnectors.StandardControlBus....83
Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.InternalConnectors.StandardSubControlBus....83
Modelica.Blocks.Examples.BusUsage_Utilities.Part....83
Modelica.Blocks.Continuous....83
Modelica.Blocks.Continuous.Integrator....85
Modelica.Blocks.Continuous.LimIntegrator....85
Modelica.Blocks.Continuous.Derivative....86
Modelica.Blocks.Continuous.FirstOrder....87
Modelica.Blocks.Continuous.SecondOrder....88
Modelica.Blocks.Continuous.PI....88
Modelica.Blocks.Continuous.PID....89
Modelica.Blocks.Continuous.LimPID....90
Modelica.Blocks.Continuous.TransferFunction....92
Modelica.Blocks.Continuous.StateSpace....93
Modelica.Blocks.Continuous.Der....94
Modelica.Blocks.Continuous.LowpassButterworth....95
Modelica.Blocks.Continuous.CriticalDamping....95
Modelica.Blocks.Discrete....96
Modelica.Blocks.Discrete.Sampler....97
Modelica.Blocks.Discrete.ZeroOrderHold....97
Modelica.Blocks.Discrete.FirstOrderHold....97
Modelica.Blocks.Discrete.UnitDelay....98
Modelica.Blocks.Discrete.TransferFunction....98
Modelica.Blocks.Discrete.StateSpace....99
Modelica.Blocks.Discrete.TriggeredSampler....100
Modelica.Blocks.Discrete.TriggeredMax....101
Modelica.Blocks.Interfaces....101

Modelica.Blocks.Interfaces.RealSignal 103
Modelica.Blocks.Interfaces.BooleanSignal 103
Modelica.Blocks.Interfaces.IntegerSignal 103
Modelica.Blocks.Interfaces.RealInput 103
Modelica.Blocks.Interfaces.RealOutput 104
Modelica.Blocks.Interfaces.BooleanInput 104
Modelica.Blocks.Interfaces.BooleanOutput 104
Modelica.Blocks.Interfaces.IntegerInput 104
Modelica.Blocks.Interfaces.IntegerOutput 104
Modelica.Blocks.Interfaces.BlockIcon 104
Modelica.Blocks.Interfaces.SO 105
Modelica.Blocks.Interfaces.MO 105
Modelica.Blocks.Interfaces.SISO 105
Modelica.Blocks.Interfaces.SI2SO 106
Modelica.Blocks.Interfaces.SIMO 106
Modelica.Blocks.Interfaces.MISO 106
Modelica.Blocks.Interfaces.MIMO 107
Modelica.Blocks.Interfaces.MIMOs 107
Modelica.Blocks.Interfaces.MI2MO 107
Modelica.Blocks.Interfaces.SignalSource 108
Modelica.Blocks.Interfaces.SVcontrol 108
Modelica.Blocks.Interfaces.MVcontrol 109
Modelica.Blocks.Interfaces.DiscreteBlockIcon 109
Modelica.Blocks.Interfaces.DiscreteBlock 109
Modelica.Blocks.Interfaces.DiscreteSISO 110
Modelica.Blocks.Interfaces.DiscreteMIMO 110
Modelica.Blocks.Interfaces.DiscreteMIMOs 110
Modelica.Blocks.Interfaces.SVdiscrete 111
Modelica.Blocks.Interfaces.MVdiscrete 111
Modelica.Blocks.Interfaces.BooleanBlockIcon 112
Modelica.Blocks.Interfaces.BooleanSISO 112
Modelica.Blocks.Interfaces.BooleanMIMOs 112
Modelica.Blocks.Interfaces.MI2BooleanMOs 113
Modelica.Blocks.Interfaces.SI2BooleanSO 113
Modelica.Blocks.Interfaces.BooleanSignalSource 114
Modelica.Blocks.Interfaces.IntegerBlockIcon 114
Modelica.Blocks.Interfaces.IntegerSO 114
Modelica.Blocks.Interfaces.IntegerMO 114
Modelica.Blocks.Interfaces.IntegerSignalSource 115
Modelica.Blocks.Interfaces.IntegerSIBooleanSO 115
Modelica.Blocks.Interfaces.IntegerMIBooleanMOs 115
Modelica.Blocks.Interfaces.partialBooleanBlockIcon 116
Modelica.Blocks.Interfaces.partialBooleanSISO 116
Modelica.Blocks.Interfaces.partialBooleanSI2SO 116
Modelica.Blocks.Interfaces.partialBooleanSI3SO 116
Modelica.Blocks.Interfaces.partialBooleanSI 117
Modelica.Blocks.Interfaces.partialBooleanSO 117
Modelica.Blocks.Interfaces.partialBooleanSource 117
Modelica.Blocks.Interfaces.partialBooleanThresholdComparison 118
Modelica.Blocks.Interfaces.partialBooleanComparison 118
Modelica.Blocks.Interfaces.Adaptors 118
Modelica.Blocks.Interfaces.Adaptors.SendReal 119
Modelica.Blocks.Interfaces.Adaptors.SendBoolean 119
Modelica.Blocks.Interfaces.Adaptors.SendInteger 120
Modelica.Blocks.Interfaces.Adaptors.ReceiveReal 120
Modelica.Blocks.Interfaces.Adaptors.ReceiveBoolean 120
Modelica.Blocks.Interfaces.Adaptors.ReceiveInteger 121
Modelica.Blocks.Interfaces.Adaptors.AdaptorReal 121

Modelica.Blocks.Interfaces.Adaptors.AdaptorBoolean....121
Modelica.Blocks.Interfaces.Adaptors.AdaptorInteger....122
Modelica.Blocks.Interfaces.PartialConversionBlock....122
Modelica.Blocks.Logical....122
Modelica.Blocks.Logical.And....124
Modelica.Blocks.Logical.Or....124
Modelica.Blocks.Logical.Xor....124
Modelica.Blocks.Logical.Nor....125
Modelica.Blocks.Logical.Nand....125
Modelica.Blocks.Logical.Not....125
Modelica.Blocks.Logical.Pre....126
Modelica.Blocks.Logical.Edge....126
Modelica.Blocks.Logical.FallingEdge....127
Modelica.Blocks.Logical.Change....127
Modelica.Blocks.Logical.GreaterThreshold....127
Modelica.Blocks.Logical.GreaterEqualThreshold....128
Modelica.Blocks.Logical.LessThreshold....128
Modelica.Blocks.Logical.LessEqualThreshold....128
Modelica.Blocks.Logical.Greater....129
Modelica.Blocks.Logical.GreaterEqual....129
Modelica.Blocks.Logical.Less....129
Modelica.Blocks.Logical.LessEqual....130
Modelica.Blocks.Logical.ZeroCrossing....130
Modelica.Blocks.Logical.LogicalSwitch....131
Modelica.Blocks.Logical.Switch....131
Modelica.Blocks.Logical.Hysteresis....131
Modelica.Blocks.Logical.OnOffController....132
Modelica.Blocks.Logical.TriggeredTrapezoid....132
Modelica.Blocks.Logical.Timer....133
Modelica.Blocks.Logical.TerminateSimulation....133
Modelica.Blocks.Math....134
Modelica.Blocks.Math.UnitConversions....135
Modelica.Blocks.Math.UnitConversions.ConvertAllUnits....136
Modelica.Blocks.Math.UnitConversions.To_degC....137
Modelica.Blocks.Math.UnitConversions.From_degC....137
Modelica.Blocks.Math.UnitConversions.To_degF....138
Modelica.Blocks.Math.UnitConversions.From_degF....138
Modelica.Blocks.Math.UnitConversions.To_degRk....138
Modelica.Blocks.Math.UnitConversions.From_degRk....138
Modelica.Blocks.Math.UnitConversions.To_deg....138
Modelica.Blocks.Math.UnitConversions.From_deg....138
Modelica.Blocks.Math.UnitConversions.To_rpm....139
Modelica.Blocks.Math.UnitConversions.From_rpm....139
Modelica.Blocks.Math.UnitConversions.To_kmh....139
Modelica.Blocks.Math.UnitConversions.From_kmh....139
Modelica.Blocks.Math.UnitConversions.To_day....139
Modelica.Blocks.Math.UnitConversions.From_day....139
Modelica.Blocks.Math.UnitConversions.To_hour....140
Modelica.Blocks.Math.UnitConversions.From_hour....140
Modelica.Blocks.Math.UnitConversions.To_minute....140
Modelica.Blocks.Math.UnitConversions.From_minute....140
Modelica.Blocks.Math.UnitConversions.To_litre....140
Modelica.Blocks.Math.UnitConversions.From_litre....140
Modelica.Blocks.Math.UnitConversions.To_kWh....141
Modelica.Blocks.Math.UnitConversions.From_kWh....141
Modelica.Blocks.Math.UnitConversions.To_bar....141
Modelica.Blocks.Math.UnitConversions.From_bar....141
Modelica.Blocks.Math.UnitConversions.To_gps....141

Modelica.Blocks.Math.UnitConversions.From_gps ... 141
Modelica.Blocks.Math.TwoInputs ... 142
Modelica.Blocks.Math.TwoOutputs ... 142
Modelica.Blocks.Math.Gain ... 142
Modelica.Blocks.Math.MatrixGain ... 143
Modelica.Blocks.Math.Sum ... 143
Modelica.Blocks.Math.Feedback ... 144
Modelica.Blocks.Math.Add ... 144
Modelica.Blocks.Math.Add3 ... 145
Modelica.Blocks.Math.Product ... 146
Modelica.Blocks.Math.Division ... 146
Modelica.Blocks.Math.Abs ... 146
Modelica.Blocks.Math.Sign ... 147
Modelica.Blocks.Math.Sqrt ... 147
Modelica.Blocks.Math.Sin ... 147
Modelica.Blocks.Math.Cos ... 148
Modelica.Blocks.Math.Tan ... 149
Modelica.Blocks.Math.Asin ... 149
Modelica.Blocks.Math.Acos ... 150
Modelica.Blocks.Math.Atan ... 151
Modelica.Blocks.Math.Atan2 ... 151
Modelica.Blocks.Math.Sinh ... 152
Modelica.Blocks.Math.Cosh ... 153
Modelica.Blocks.Math.Tanh ... 153
Modelica.Blocks.Math.Exp ... 154
Modelica.Blocks.Math.Log ... 155
Modelica.Blocks.Math.Log10 ... 155
Modelica.Blocks.Math.RealToInteger ... 156
Modelica.Blocks.Math.IntegerToReal ... 156
Modelica.Blocks.Math.BooleanToReal ... 157
Modelica.Blocks.Math.BooleanToInteger ... 157
Modelica.Blocks.Math.RealToBoolean ... 158
Modelica.Blocks.Math.IntegerToBoolean ... 158
Modelica.Blocks.Math.Max ... 159
Modelica.Blocks.Math.Min ... 159
Modelica.Blocks.Math.Edge ... 159
Modelica.Blocks.Math.BooleanChange ... 160
Modelica.Blocks.Math.IntegerChange ... 160
Modelica.Blocks.Nonlinear ... 160
Modelica.Blocks.Nonlinear.Limiter ... 161
Modelica.Blocks.Nonlinear.VariableLimiter ... 161
Modelica.Blocks.Nonlinear.DeadZone ... 162
Modelica.Blocks.Nonlinear.FixedDelay ... 162
Modelica.Blocks.Nonlinear.PadeDelay ... 162
Modelica.Blocks.Nonlinear.VariableDelay ... 163
Modelica.Blocks.Routing ... 164
Modelica.Blocks.Routing.ExtractSignal ... 164
Modelica.Blocks.Routing.Extractor ... 165
Modelica.Blocks.Routing.Multiplex2 ... 166
Modelica.Blocks.Routing.Multiplex3 ... 166
Modelica.Blocks.Routing.Multiplex4 ... 167
Modelica.Blocks.Routing.Multiplex5 ... 167
Modelica.Blocks.Routing.Multiplex6 ... 168
Modelica.Blocks.Routing.DeMultiplex2 ... 168
Modelica.Blocks.Routing.DeMultiplex3 ... 169
Modelica.Blocks.Routing.DeMultiplex4 ... 169
Modelica.Blocks.Routing.DeMultiplex5 ... 170
Modelica.Blocks.Routing.DeMultiplex6 ... 170

Modelica.Blocks.Routing.RealPassThrough....171
Modelica.Blocks.Routing.IntegerPassThrough....171
Modelica.Blocks.Routing.BooleanPassThrough....172
Modelica.Blocks.Sources....172
Modelica.Blocks.Sources.RealExpression....173
Modelica.Blocks.Sources.IntegerExpression....174
Modelica.Blocks.Sources.BooleanExpression....174
Modelica.Blocks.Sources.Clock....175
Modelica.Blocks.Sources.Constant....175
Modelica.Blocks.Sources.Step....176
Modelica.Blocks.Sources.Ramp....177
Modelica.Blocks.Sources.Sine....178
Modelica.Blocks.Sources.ExpSine....179
Modelica.Blocks.Sources.Exponentials....180
Modelica.Blocks.Sources.Pulse....181
Modelica.Blocks.Sources.SawTooth....182
Modelica.Blocks.Sources.Trapezoid....183
Modelica.Blocks.Sources.KinematicPTP....184
Modelica.Blocks.Sources.KinematicPTP2....185
Modelica.Blocks.Sources.TimeTable....186
Modelica.Blocks.Sources.CombiTimeTable....188
Modelica.Blocks.Sources.BooleanConstant....190
Modelica.Blocks.Sources.BooleanStep....191
Modelica.Blocks.Sources.BooleanPulse....192
Modelica.Blocks.Sources.SampleTrigger....193
Modelica.Blocks.Sources.BooleanTable....194
Modelica.Blocks.Sources.IntegerConstant....195
Modelica.Blocks.Sources.IntegerStep....196
Modelica.Blocks.Tables....197
Modelica.Blocks.Tables.CombiTable1D....197
Modelica.Blocks.Tables.CombiTable1Ds....199
Modelica.Blocks.Tables.CombiTable2D....201
Modelica.Blocks.Types....203
Modelica.Blocks.Types.Extrapolation....203
Modelica.Blocks.Types.Init....204
Modelica.Blocks.Types.InitPID....205
Modelica.Blocks.Types.SimpleController....205
Modelica.Blocks.Types.Smoothness....206
Modelica.Blocks.Types.StateSelection....207
Modelica.Constants....207
Modelica.Electrical....209
Modelica.Electrical.Analog....210
Modelica.Electrical.Analog.Examples....211
Modelica.Electrical.Analog.Examples.CauerLowPassAnalog....211
Modelica.Electrical.Analog.Examples.CauerLowPassOPV....212
Modelica.Electrical.Analog.Examples.CauerLowPassSC....214
Modelica.Electrical.Analog.Examples.CharacteristicIdealDiodes....215
Modelica.Electrical.Analog.Examples.CharacteristicThyristors....216
Modelica.Electrical.Analog.Examples.ChuaCircuit....216
Modelica.Electrical.Analog.Examples.DifferenceAmplifier....217
Modelica.Electrical.Analog.Examples.HeatingMOSInverter....218
Modelica.Electrical.Analog.Examples.HeatingNPN_OrGate....218
Modelica.Electrical.Analog.Examples.HeatingRectifier....219
Modelica.Electrical.Analog.Examples.NandGate....220
Modelica.Electrical.Analog.Examples.Rectifier....220
Modelica.Electrical.Analog.Examples.ShowSaturatingInductor....221
Modelica.Electrical.Analog.Examples.ShowVariableResistor....222
Modelica.Electrical.Analog.Examples.Utilities....222

Modelica.Electrical.Analog.Examples.Utilities.Nand 223
Modelica.Electrical.Analog.Examples.Utilities.NonlinearResistor 223
Modelica.Electrical.Analog.Examples.Utilities.RealSwitch 224
Modelica.Electrical.Analog.Examples.Utilities.Transistor 224
Modelica.Electrical.Analog.Basic 224
Modelica.Electrical.Analog.Basic.Ground 225
Modelica.Electrical.Analog.Basic.Resistor 225
Modelica.Electrical.Analog.Basic.HeatingResistor 226
Modelica.Electrical.Analog.Basic.Conductor 226
Modelica.Electrical.Analog.Basic.Capacitor 227
Modelica.Electrical.Analog.Basic.Inductor 227
Modelica.Electrical.Analog.Basic.SaturatingInductor 228
Modelica.Electrical.Analog.Basic.Transformer 228
Modelica.Electrical.Analog.Basic.Gyrator 229
Modelica.Electrical.Analog.Basic.EMF 229
Modelica.Electrical.Analog.Basic.VCV 230
Modelica.Electrical.Analog.Basic.VCC 230
Modelica.Electrical.Analog.Basic.CCV 231
Modelica.Electrical.Analog.Basic.CCC 231
Modelica.Electrical.Analog.Basic.OpAmp 232
Modelica.Electrical.Analog.Basic.VariableResistor 232
Modelica.Electrical.Analog.Basic.VariableConductor 233
Modelica.Electrical.Analog.Basic.VariableCapacitor 233
Modelica.Electrical.Analog.Basic.VariableInductor 234
Modelica.Electrical.Analog.Ideal 234
Modelica.Electrical.Analog.Ideal.IdealThyristor 235
Modelica.Electrical.Analog.Ideal.IdealGTOThyristor 236
Modelica.Electrical.Analog.Ideal.IdealCommutingSwitch 236
Modelica.Electrical.Analog.Ideal.IdealIntermediateSwitch 237
Modelica.Electrical.Analog.Ideal.ControlledIdealCommutingSwitch 238
Modelica.Electrical.Analog.Ideal.ControlledIdealIntermediateSwitch 238
Modelica.Electrical.Analog.Ideal.IdealOpAmp 239
Modelica.Electrical.Analog.Ideal.IdealOpAmp3Pin 240
Modelica.Electrical.Analog.Ideal.IdealOpAmpLimited 240
Modelica.Electrical.Analog.Ideal.IdealDiode 240
Modelica.Electrical.Analog.Ideal.IdealTransformer 241
Modelica.Electrical.Analog.Ideal.IdealGyrator 242
Modelica.Electrical.Analog.Ideal.Idle 242
Modelica.Electrical.Analog.Ideal.Short 242
Modelica.Electrical.Analog.Ideal.IdealOpeningSwitch 243
Modelica.Electrical.Analog.Ideal.IdealClosingSwitch 243
Modelica.Electrical.Analog.Ideal.ControlledIdealOpeningSwitch 244
Modelica.Electrical.Analog.Ideal.ControlledIdealClosingSwitch 244
Modelica.Electrical.Analog.Interfaces 245
Modelica.Electrical.Analog.Interfaces.Pin 246
Modelica.Electrical.Analog.Interfaces.PositivePin 246
Modelica.Electrical.Analog.Interfaces.NegativePin 246
Modelica.Electrical.Analog.Interfaces.TwoPin 246
Modelica.Electrical.Analog.Interfaces.OnePort 247
Modelica.Electrical.Analog.Interfaces.TwoPort 247
Modelica.Electrical.Analog.Interfaces.AbsoluteSensor 247
Modelica.Electrical.Analog.Interfaces.RelativeSensor 248
Modelica.Electrical.Analog.Interfaces.VoltageSource 248
Modelica.Electrical.Analog.Interfaces.CurrentSource 248
Modelica.Electrical.Analog.Lines 249
Modelica.Electrical.Analog.Lines.OLine 249
Modelica.Electrical.Analog.Lines.ULine 250
Modelica.Electrical.Analog.Lines.TLine1 250

Modelica.Electrical.Analog.Lines.TLine2 251
Modelica.Electrical.Analog.Lines.TLine3 252
Modelica.Electrical.Analog.Semiconductors 252
Modelica.Electrical.Analog.Semiconductors.Diode 253
Modelica.Electrical.Analog.Semiconductors.PMOS 253
Modelica.Electrical.Analog.Semiconductors.NMOS 254
Modelica.Electrical.Analog.Semiconductors.NPN 256
Modelica.Electrical.Analog.Semiconductors.PNP 257
Modelica.Electrical.Analog.Semiconductors.HeatingDiode 258
Modelica.Electrical.Analog.Semiconductors.HeatingNMOS 258
Modelica.Electrical.Analog.Semiconductors.HeatingPMOS 260
Modelica.Electrical.Analog.Semiconductors.HeatingNPN 261
Modelica.Electrical.Analog.Semiconductors.HeatingPNP 262
Modelica.Electrical.Analog.Sensors 263
Modelica.Electrical.Analog.Sensors.PotentialSensor 264
Modelica.Electrical.Analog.Sensors.VoltageSensor 264
Modelica.Electrical.Analog.Sensors.CurrentSensor 264
Modelica.Electrical.Analog.Sensors.PowerSensor 265
Modelica.Electrical.Analog.Sources 265
Modelica.Electrical.Analog.Sources.SignalVoltage 266
Modelica.Electrical.Analog.Sources.ConstantVoltage 266
Modelica.Electrical.Analog.Sources.StepVoltage 266
Modelica.Electrical.Analog.Sources.RampVoltage 267
Modelica.Electrical.Analog.Sources.SineVoltage 267
Modelica.Electrical.Analog.Sources.ExpSineVoltage 268
Modelica.Electrical.Analog.Sources.ExponentialsVoltage 268
Modelica.Electrical.Analog.Sources.PulseVoltage 268
Modelica.Electrical.Analog.Sources.SawToothVoltage 269
Modelica.Electrical.Analog.Sources.TrapezoidVoltage 269
Modelica.Electrical.Analog.Sources.TableVoltage 270
Modelica.Electrical.Analog.Sources.SignalCurrent 271
Modelica.Electrical.Analog.Sources.ConstantCurrent 271
Modelica.Electrical.Analog.Sources.StepCurrent 271
Modelica.Electrical.Analog.Sources.RampCurrent 272
Modelica.Electrical.Analog.Sources.SineCurrent 272
Modelica.Electrical.Analog.Sources.ExpSineCurrent 272
Modelica.Electrical.Analog.Sources.ExponentialsCurrent 273
Modelica.Electrical.Analog.Sources.PulseCurrent 273
Modelica.Electrical.Analog.Sources.SawToothCurrent 274
Modelica.Electrical.Analog.Sources.TrapezoidCurrent 274
Modelica.Electrical.Analog.Sources.TableCurrent 275
Modelica.Electrical.Digital 275
Modelica.Electrical.Digital.UsersGuide 277
Modelica.Electrical.Digital.UsersGuide.OverView 277
Modelica.Electrical.Digital.UsersGuide.FirstExample 277
Modelica.Electrical.Digital.UsersGuide.ApplicationExample 278
Modelica.Electrical.Digital.UsersGuide.ReleaseNotes 278
Modelica.Electrical.Digital.UsersGuide.Literature 279
Modelica.Electrical.Digital.UsersGuide.Contact 279
Modelica.Electrical.Digital.Examples 280
Modelica.Electrical.Digital.Examples.Multiplexer 280
Modelica.Electrical.Digital.Examples.FlipFlop 281
Modelica.Electrical.Digital.Examples.HalfAdder 281
Modelica.Electrical.Digital.Examples.FullAdder 282
Modelica.Electrical.Digital.Examples.Adder4 283
Modelica.Electrical.Digital.Examples.Counter3 284
Modelica.Electrical.Digital.Examples.Counter 284
Modelica.Electrical.Digital.Examples.Utilities 284

Modelica.Electrical.Digital.Examples.Utilities.MUX4 285
Modelica.Electrical.Digital.Examples.Utilities.RS 285
Modelica.Electrical.Digital.Examples.Utilities.RSFF 286
Modelica.Electrical.Digital.Examples.Utilities.DFF 286
Modelica.Electrical.Digital.Examples.Utilities.JKFF 287
Modelica.Electrical.Digital.Examples.Utilities.HalfAdder 287
Modelica.Electrical.Digital.Examples.Utilities.FullAdder 287
Modelica.Electrical.Digital.Examples.Utilities.Adder 288
Modelica.Electrical.Digital.Examples.Utilities.Counter3 288
Modelica.Electrical.Digital.Examples.Utilities.Counter 289
Modelica.Electrical.Digital.Interfaces 289
Modelica.Electrical.Digital.Interfaces.LogicValue 290
Modelica.Electrical.Digital.Interfaces.DigitalSignal 290
Modelica.Electrical.Digital.Interfaces.DigitalInput 290
Modelica.Electrical.Digital.Interfaces.DigitalOutput 291
Modelica.Electrical.Digital.Interfaces.SISO 291
Modelica.Electrical.Digital.Interfaces.MISO 291
Modelica.Electrical.Digital.Tables 291
Modelica.Electrical.Digital.Delay 293
Modelica.Electrical.Digital.Delay.DelayParams 293
Modelica.Electrical.Digital.Delay.TransportDelay 294
Modelica.Electrical.Digital.Delay.InertialDelay 294
Modelica.Electrical.Digital.Delay.InertialDelaySensitive 295
Modelica.Electrical.Digital.Basic 295
Modelica.Electrical.Digital.Basic.Not 295
Modelica.Electrical.Digital.Basic.And 296
Modelica.Electrical.Digital.Basic.Nand 296
Modelica.Electrical.Digital.Basic.Or 296
Modelica.Electrical.Digital.Basic.Nor 297
Modelica.Electrical.Digital.Basic.Xor 297
Modelica.Electrical.Digital.Basic.Xnor 298
Modelica.Electrical.Digital.Gates 298
Modelica.Electrical.Digital.Gates.InvGate 298
Modelica.Electrical.Digital.Gates.AndGate 299
Modelica.Electrical.Digital.Gates.NandGate 299
Modelica.Electrical.Digital.Gates.OrGate 300
Modelica.Electrical.Digital.Gates.NorGate 300
Modelica.Electrical.Digital.Gates.XorGate 301
Modelica.Electrical.Digital.Gates.XnorGate 301
Modelica.Electrical.Digital.Gates.BufGate 302
Modelica.Electrical.Digital.Sources 302
Modelica.Electrical.Digital.Sources.Set 302
Modelica.Electrical.Digital.Sources.Step 303
Modelica.Electrical.Digital.Sources.Table 304
Modelica.Electrical.Digital.Sources.Pulse 305
Modelica.Electrical.Digital.Sources.Clock 305
Modelica.Electrical.Digital.Converters 306
Modelica.Electrical.Digital.Converters.LogicToXO1 306
Modelica.Electrical.Digital.Converters.LogicToXO1Z 307
Modelica.Electrical.Digital.Converters.LogicToUX01 308
Modelica.Electrical.Digital.Converters.BooleanToLogic 308
Modelica.Electrical.Digital.Converters.LogicToBoolean 309
Modelica.Electrical.Digital.Converters.RealToLogic 309
Modelica.Electrical.Digital.Converters.LogicToReal 310
Modelica.Electrical.Machines 311
Modelica.Electrical.Machines.Examples 312
Modelica.Electrical.Machines.Examples.AIMC_DOL 312
Modelica.Electrical.Machines.Examples.AIMC_YD 313

Modelica.Electrical.Machines.Examples.AIMS_start....314
Modelica.Electrical.Machines.Examples.AIMC_Inverter....315
Modelica.Electrical.Machines.Examples.SMR_Inverter....316
Modelica.Electrical.Machines.Examples.SMPM_Inverter....317
Modelica.Electrical.Machines.Examples.SMEE_Gen....318
Modelica.Electrical.Machines.Examples.DCPM_start....319
Modelica.Electrical.Machines.Examples.DCEE_start....320
Modelica.Electrical.Machines.Examples.DCSE_start....321
Modelica.Electrical.Machines.Examples.TransformerTestbench....322
Modelica.Electrical.Machines.Examples.Rectifier6pulse....323
Modelica.Electrical.Machines.Examples.Rectifier12pulse....324
Modelica.Electrical.Machines.Examples.AIMC_Steinmetz....324
Modelica.Electrical.Machines.Examples.Utilities....325
Modelica.Electrical.Machines.Examples.Utilities.VfController....326
Modelica.Electrical.Machines.Examples.Utilities.SwitchYD....326
Modelica.Electrical.Machines.Examples.Utilities.TerminalBox....326
Modelica.Electrical.Machines.BasicMachines....327
Modelica.Electrical.Machines.BasicMachines.AsynchronousInductionMachines....327
Modelica.Electrical.Machines.BasicMachines.AsynchronousInductionMachines.AIM_SquirrelCage....328
Modelica.Electrical.Machines.BasicMachines.AsynchronousInductionMachines.AIM_SlipRing....329
Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines....331
Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines.SM_PermanentMagnetDamperCage....331
Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines.SM_ElectricalExcitedDamperCage....333
Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines.SM_ReluctanceRotorDamperCage....335
Modelica.Electrical.Machines.BasicMachines.DCMachines....336
Modelica.Electrical.Machines.BasicMachines.DCMachines.DC_PermanentMagnet....336
Modelica.Electrical.Machines.BasicMachines.DCMachines.DC_ElectricalExcited....337
Modelica.Electrical.Machines.BasicMachines.DCMachines.DC_SeriesExcited....339
Modelica.Electrical.Machines.BasicMachines.Transformers....340
Modelica.Electrical.Machines.BasicMachines.Transformers.Transformer....341
Modelica.Electrical.Machines.BasicMachines.Transformers.Yy....342
Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy00....342
Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy02....343
Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy04....344
Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy06....344
Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy08....345
Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy10....345
Modelica.Electrical.Machines.BasicMachines.Transformers.Yd....346
Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd01....347
Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd03....347
Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd05....348
Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd07....348
Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd09....349
Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd11....350
Modelica.Electrical.Machines.BasicMachines.Transformers.Yz....350
Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz01....351
Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz03....351
Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz05....352
Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz07....353
Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz09....353
Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz11....354
Modelica.Electrical.Machines.BasicMachines.Transformers.Dy....355
Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy01....355
Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy03....356
Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy05....356

Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy07 357
Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy09 358
Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy11 358
Modelica.Electrical.Machines.BasicMachines.Transformers.Dd 359
Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd00 359
Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd02 360
Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd04 361
Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd06 361
Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd08 362
Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd10 362
Modelica.Electrical.Machines.BasicMachines.Transformers.Dz 363
Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz00 363
Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz02 364
Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz04 365
Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz06 365
Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz08 366
Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz10 367
Modelica.Electrical.Machines.BasicMachines.Components 367
Modelica.Electrical.Machines.BasicMachines.Components.BasicAIM 368
Modelica.Electrical.Machines.BasicMachines.Components.BasicSM 369
Modelica.Electrical.Machines.BasicMachines.Components.PartialAirGap 369
Modelica.Electrical.Machines.BasicMachines.Components.AirGapS 370
Modelica.Electrical.Machines.BasicMachines.Components.AirGapR 370
Modelica.Electrical.Machines.BasicMachines.Components.SquirrelCage 371
Modelica.Electrical.Machines.BasicMachines.Components.DamperCage 371
Modelica.Electrical.Machines.BasicMachines.Components.ElectricalExcitation 371
Modelica.Electrical.Machines.BasicMachines.Components.PermanentMagnet 372
Modelica.Electrical.Machines.BasicMachines.Components.BasicDCMachine 372
Modelica.Electrical.Machines.BasicMachines.Components.PartialAirGapDC 373
Modelica.Electrical.Machines.BasicMachines.Components.AirGapDC 374
Modelica.Electrical.Machines.BasicMachines.Components.BasicTransformer 374
Modelica.Electrical.Machines.BasicMachines.Components.PartialCore 375
Modelica.Electrical.Machines.BasicMachines.Components.IdealCore 375
Modelica.Electrical.Machines.Sensors 376
Modelica.Electrical.Machines.Sensors.VoltageRMSsensor 376
Modelica.Electrical.Machines.Sensors.CurrentRMSsensor 376
Modelica.Electrical.Machines.Sensors.ElectricalPowerSensor 377
Modelica.Electrical.Machines.Sensors.MechanicalPowerSensor 377
Modelica.Electrical.Machines.Sensors.RotorAngle 378
Modelica.Electrical.Machines.SpacePhasors 378
Modelica.Electrical.Machines.SpacePhasors.Components 378
Modelica.Electrical.Machines.SpacePhasors.Components.SpacePhasor 379
Modelica.Electrical.Machines.SpacePhasors.Components.Rotator 379
Modelica.Electrical.Machines.SpacePhasors.Blocks 380
Modelica.Electrical.Machines.SpacePhasors.Blocks.ToSpacePhasor 380
Modelica.Electrical.Machines.SpacePhasors.Blocks.FromSpacePhasor 380
Modelica.Electrical.Machines.SpacePhasors.Blocks.Rotator 381
Modelica.Electrical.Machines.SpacePhasors.Blocks.ToPolar 381
Modelica.Electrical.Machines.SpacePhasors.Blocks.FromPolar 381
Modelica.Electrical.Machines.SpacePhasors.Functions 382
Modelica.Electrical.Machines.SpacePhasors.Functions.ToSpacePhasor 382
Modelica.Electrical.Machines.SpacePhasors.Functions.FromSpacePhasor 383
Modelica.Electrical.Machines.SpacePhasors.Functions.Rotator 383
Modelica.Electrical.Machines.SpacePhasors.Functions.ToPolar 383
Modelica.Electrical.Machines.SpacePhasors.Functions.FromPolar 384
Modelica.Electrical.Machines.Interfaces 384
Modelica.Electrical.Machines.Interfaces.SpacePhasor 385
Modelica.Electrical.Machines.Interfaces.Adapter 385

Modelica.Electrical.Machines.Interfaces.PartialBasicMachine 385
Modelica.Electrical.Machines.Interfaces.PartialBasicInductionMachine 386
Modelica.Electrical.Machines.Interfaces.PartialBasicDCMachine 387
Modelica.Electrical.MultiPhase 387
Modelica.Electrical.MultiPhase.Basic 388
Modelica.Electrical.MultiPhase.Basic.Star 389
Modelica.Electrical.MultiPhase.Basic.Delta 389
Modelica.Electrical.MultiPhase.Basic.PlugToPin_p 389
Modelica.Electrical.MultiPhase.Basic.PlugToPin_n 390
Modelica.Electrical.MultiPhase.Basic.Resistor 390
Modelica.Electrical.MultiPhase.Basic.Conductor 391
Modelica.Electrical.MultiPhase.Basic.Capacitor 391
Modelica.Electrical.MultiPhase.Basic.Inductor 391
Modelica.Electrical.MultiPhase.Basic.SaturatingInductor 392
Modelica.Electrical.MultiPhase.Basic.Transformer 392
Modelica.Electrical.MultiPhase.Basic.VariableResistor 393
Modelica.Electrical.MultiPhase.Basic.VariableConductor 393
Modelica.Electrical.MultiPhase.Basic.VariableCapacitor 394
Modelica.Electrical.MultiPhase.Basic.VariableInductor 394
Modelica.Electrical.MultiPhase.Examples 395
Modelica.Electrical.MultiPhase.Examples.TransformerYY 395
Modelica.Electrical.MultiPhase.Examples.TransformerYD 395
Modelica.Electrical.MultiPhase.Examples.Rectifier 396
Modelica.Electrical.MultiPhase.Ideal 396
Modelica.Electrical.MultiPhase.Ideal.IdealThyristor 397
Modelica.Electrical.MultiPhase.Ideal.IdealGTOThyristor 397
Modelica.Electrical.MultiPhase.Ideal.IdealCommutingSwitch 398
Modelica.Electrical.MultiPhase.Ideal.IdealIntermediateSwitch 398
Modelica.Electrical.MultiPhase.Ideal.IdealDiode 399
Modelica.Electrical.MultiPhase.Ideal.IdealTransformer 399
Modelica.Electrical.MultiPhase.Ideal.Idle 400
Modelica.Electrical.MultiPhase.Ideal.Short 400
Modelica.Electrical.MultiPhase.Ideal.IdealOpeningSwitch 401
Modelica.Electrical.MultiPhase.Ideal.IdealClosingSwitch 401
Modelica.Electrical.MultiPhase.Interfaces 401
Modelica.Electrical.MultiPhase.Interfaces.Plug 402
Modelica.Electrical.MultiPhase.Interfaces.PositivePlug 402
Modelica.Electrical.MultiPhase.Interfaces.NegativePlug 403
Modelica.Electrical.MultiPhase.Interfaces.TwoPlug 403
Modelica.Electrical.MultiPhase.Interfaces.OnePort 404
Modelica.Electrical.MultiPhase.Interfaces.FourPlug 404
Modelica.Electrical.MultiPhase.Interfaces.TwoPort 404
Modelica.Electrical.MultiPhase.Sensors 405
Modelica.Electrical.MultiPhase.Sensors.PotentialSensor 405
Modelica.Electrical.MultiPhase.Sensors.VoltageSensor 406
Modelica.Electrical.MultiPhase.Sensors.CurrentSensor 406
Modelica.Electrical.MultiPhase.Sensors.PowerSensor 407
Modelica.Electrical.MultiPhase.Sources 407
Modelica.Electrical.MultiPhase.Sources.SignalVoltage 408
Modelica.Electrical.MultiPhase.Sources.ConstantVoltage 408
Modelica.Electrical.MultiPhase.Sources.SineVoltage 408
Modelica.Electrical.MultiPhase.Sources.SignalCurrent 409
Modelica.Electrical.MultiPhase.Sources.ConstantCurrent 409
Modelica.Electrical.MultiPhase.Sources.SineCurrent 410
Modelica.Icons 410
Modelica.Icons.Info 412
Modelica.Icons.Library 412
Modelica.Icons.Library2 412

Modelica.Icons.Example 412
Modelica.Icons.Function 412
Modelica.Icons.Record 412
Modelica.Icons.Enumeration 413
Modelica.Icons.TranslationalSensor 413
Modelica.Icons.RotationalSensor 413
Modelica.Icons.GearIcon 413
Modelica.Icons.MotorIcon 413
Modelica.Icons.SignalBus 414
Modelica.Icons.SignalSubBus 414
Modelica.Math 414
Modelica.Math.Vectors 415
Modelica.Math.Vectors.isEqual 416
Modelica.Math.Vectors.norm 417
Modelica.Math.Vectors.length 418
Modelica.Math.Vectors.normalize 418
Modelica.Math.Vectors.reverse 419
Modelica.Math.Vectors.sort 420
Modelica.Math.Matrices 421
Modelica.Math.Matrices.isEqual 422
Modelica.Math.Matrices.norm 423
Modelica.Math.Matrices.sort 424
Modelica.Math.Matrices.solve 425
Modelica.Math.Matrices.solve2 426
Modelica.Math.Matrices.leastSquares 427
Modelica.Math.Matrices.equalityLeastSquares 427
Modelica.Math.Matrices.LU 428
Modelica.Math.Matrices.LU_solve 429
Modelica.Math.Matrices.LU_solve2 431
Modelica.Math.Matrices.QR 432
Modelica.Math.Matrices.eigenValues 433
Modelica.Math.Matrices.eigenValueMatrix 434
Modelica.Math.Matrices.singularValues 435
Modelica.Math.Matrices.det 436
Modelica.Math.Matrices.inv 436
Modelica.Math.Matrices.rank 437
Modelica.Math.Matrices.balance 437
Modelica.Math.Matrices.exp 438
Modelica.Math.Matrices.integralExp 439
Modelica.Math.Matrices.integralExpT 440
Modelica.Math.sin 441
Modelica.Math.cos 442
Modelica.Math.tan 443
Modelica.Math.asin 443
Modelica.Math.acos 444
Modelica.Math.atan 445
Modelica.Math.atan2 445
Modelica.Math.atan3 446
Modelica.Math.sinh 447
Modelica.Math.cosh 447
Modelica.Math.tanh 448
Modelica.Math.asinh 449
Modelica.Math.acosh 449
Modelica.Math.exp 450
Modelica.Math.log 451
Modelica.Math.log10 452
Modelica.Math.baseIcon1 452
Modelica.Math.baseIcon2 453

Modelica.Math.tempInterpol1 ..453
Modelica.Math.tempInterpol2 ..453
Modelica.Mechanics ..453
Modelica.Mechanics.MultiBody ..454
Modelica.Mechanics.MultiBody.UsersGuide ..455
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial ..455
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.OverView ..456
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.FirstExample ..457
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.LoopStructures ..461
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.LoopStructures.Introduction ..461
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.LoopStructures.PlanarLoops ..462
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.LoopStructures.AnalyticLoopHandling ..464
Modelica.Mechanics.MultiBody.UsersGuide.Upgrade ..470
Modelica.Mechanics.MultiBody.UsersGuide.ReleaseNotes ..471
Modelica.Mechanics.MultiBody.UsersGuide.Literature ..474
Modelica.Mechanics.MultiBody.UsersGuide.Contact ..475
Modelica.Mechanics.MultiBody.World ..475
Modelica.Mechanics.MultiBody.Examples ..478
Modelica.Mechanics.MultiBody.Examples.Elementary ..478
Modelica.Mechanics.MultiBody.Examples.Elementary.DoublePendulum ..482
Modelica.Mechanics.MultiBody.Examples.Elementary.ForceAndTorque ..483
Modelica.Mechanics.MultiBody.Examples.Elementary.FreeBody ..483
Modelica.Mechanics.MultiBody.Examples.Elementary.InitSpringConstant ..484
Modelica.Mechanics.MultiBody.Examples.Elementary.LineForceWithTwoMasses ..485
Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum ..486
Modelica.Mechanics.MultiBody.Examples.Elementary.PendulumWithSpringDamper ..487
Modelica.Mechanics.MultiBody.Examples.Elementary.PointGravity ..488
Modelica.Mechanics.MultiBody.Examples.Elementary.PointGravityWithPointMasses ..489
Modelica.Mechanics.MultiBody.Examples.Elementary.PointGravityWithPointMasses2 ..490
Modelica.Mechanics.MultiBody.Examples.Elementary.SpringDamperSystem ..491
Modelica.Mechanics.MultiBody.Examples.Elementary.SpringMassSystem ..492
Modelica.Mechanics.MultiBody.Examples.Elementary.SpringWithMass ..493
Modelica.Mechanics.MultiBody.Examples.Elementary.ThreeSprings ..494
Modelica.Mechanics.MultiBody.Examples.Loops ..495
Modelica.Mechanics.MultiBody.Examples.Loops.Engine1a ..497
Modelica.Mechanics.MultiBody.Examples.Loops.Engine1b ..498
Modelica.Mechanics.MultiBody.Examples.Loops.Engine1b_analytic ..499
Modelica.Mechanics.MultiBody.Examples.Loops.EngineV6 ..500
Modelica.Mechanics.MultiBody.Examples.Loops.EngineV6_analytic ..501
Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar1 ..501
Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar2 ..502
Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar_analytic ..504
Modelica.Mechanics.MultiBody.Examples.Loops.PlanarLoops_analytic ..504
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities ..505
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.Cylinder ..506
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.GasForce ..507
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.GasForce2 ..507
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.CylinderBase ..507
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.Cylinder_analytic_CAD ..508
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.EngineV6_analytic ..509
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.Engine1bBase ..510
Modelica.Mechanics.MultiBody.Examples.Systems ..510
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3 ..511
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.oneAxis ..512
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot ..513
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components ..515
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisControlBus ..516
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.ControlBus ..516

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.PathPlanning1....516
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.PathPlanning6....517
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.PathToAxisControlBus....518
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.GearType1....518
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.GearType2....519
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.Motor....519
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.Controller....520
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisType1....520
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisType2....521
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.MechanicalStructure....522
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.InternalConnectors....522
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.InternalConnectors.AxisControl Bus....523
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.InternalConnectors.ControlBus 523
Modelica.Mechanics.MultiBody.Forces....524
Modelica.Mechanics.MultiBody.Forces.WorldForce....526
Modelica.Mechanics.MultiBody.Forces.WorldTorque....527
Modelica.Mechanics.MultiBody.Forces.WorldForceAndTorque....528
Modelica.Mechanics.MultiBody.Forces.FrameForce....529
Modelica.Mechanics.MultiBody.Forces.FrameTorque....531
Modelica.Mechanics.MultiBody.Forces.FrameForceAndTorque....532
Modelica.Mechanics.MultiBody.Forces.Force....534
Modelica.Mechanics.MultiBody.Forces.Torque....535
Modelica.Mechanics.MultiBody.Forces.ForceAndTorque....537
Modelica.Mechanics.MultiBody.Forces.LineForceWithMass....539
Modelica.Mechanics.MultiBody.Forces.LineForceWithTwoMasses....540
Modelica.Mechanics.MultiBody.Forces.Spring....543
Modelica.Mechanics.MultiBody.Forces.Damper....544
Modelica.Mechanics.MultiBody.Forces.SpringDamperParallel....545
Modelica.Mechanics.MultiBody.Forces.SpringDamperSeries....546
Modelica.Mechanics.MultiBody.Frames....547
Modelica.Mechanics.MultiBody.Frames.Orientation....550
Modelica.Mechanics.MultiBody.Frames.orientationConstraint....551
Modelica.Mechanics.MultiBody.Frames.angularVelocity1....551
Modelica.Mechanics.MultiBody.Frames.angularVelocity2....552
Modelica.Mechanics.MultiBody.Frames.resolve1....552
Modelica.Mechanics.MultiBody.Frames.resolve2....552
Modelica.Mechanics.MultiBody.Frames.resolveRelative....552
Modelica.Mechanics.MultiBody.Frames.resolveDyade1....553
Modelica.Mechanics.MultiBody.Frames.resolveDyade2....553
Modelica.Mechanics.MultiBody.Frames.nullRotation....553
Modelica.Mechanics.MultiBody.Frames.inverseRotation....554
Modelica.Mechanics.MultiBody.Frames.relativeRotation....554
Modelica.Mechanics.MultiBody.Frames.absoluteRotation....554
Modelica.Mechanics.MultiBody.Frames.planarRotation....555
Modelica.Mechanics.MultiBody.Frames.planarRotationAngle....555
Modelica.Mechanics.MultiBody.Frames.axisRotation....556
Modelica.Mechanics.MultiBody.Frames.axesRotations....556
Modelica.Mechanics.MultiBody.Frames.axesRotationsAngles....556
Modelica.Mechanics.MultiBody.Frames.smallRotation....557
Modelica.Mechanics.MultiBody.Frames.from_nxy....558
Modelica.Mechanics.MultiBody.Frames.from_nxz....558
Modelica.Mechanics.MultiBody.Frames.from_T....559
Modelica.Mechanics.MultiBody.Frames.from_T2....559
Modelica.Mechanics.MultiBody.Frames.from_T_inv....560
Modelica.Mechanics.MultiBody.Frames.from_Q....560
Modelica.Mechanics.MultiBody.Frames.to_T....560

Modelica.Mechanics.MultiBody.Frames.to_T_inv....561
Modelica.Mechanics.MultiBody.Frames.to_Q....561
Modelica.Mechanics.MultiBody.Frames.to_vector....561
Modelica.Mechanics.MultiBody.Frames.to_exy....561
Modelica.Mechanics.MultiBody.Frames.length....562
Modelica.Mechanics.MultiBody.Frames.normalize....562
Modelica.Mechanics.MultiBody.Frames.axis....562
Modelica.Mechanics.MultiBody.Frames.Quaternions....563
Modelica.Mechanics.MultiBody.Frames.Quaternions.orientationConstraint....565
Modelica.Mechanics.MultiBody.Frames.Quaternions.angularVelocity1....565
Modelica.Mechanics.MultiBody.Frames.Quaternions.angularVelocity2....566
Modelica.Mechanics.MultiBody.Frames.Quaternions.resolve1....566
Modelica.Mechanics.MultiBody.Frames.Quaternions.resolve2....566
Modelica.Mechanics.MultiBody.Frames.Quaternions.multipleResolve1....566
Modelica.Mechanics.MultiBody.Frames.Quaternions.multipleResolve2....567
Modelica.Mechanics.MultiBody.Frames.Quaternions.nullRotation....567
Modelica.Mechanics.MultiBody.Frames.Quaternions.inverseRotation....567
Modelica.Mechanics.MultiBody.Frames.Quaternions.relativeRotation....568
Modelica.Mechanics.MultiBody.Frames.Quaternions.absoluteRotation....568
Modelica.Mechanics.MultiBody.Frames.Quaternions.planarRotation....568
Modelica.Mechanics.MultiBody.Frames.Quaternions.smallRotation....568
Modelica.Mechanics.MultiBody.Frames.Quaternions.from_T....569
Modelica.Mechanics.MultiBody.Frames.Quaternions.from_T_inv....569
Modelica.Mechanics.MultiBody.Frames.Quaternions.to_T....569
Modelica.Mechanics.MultiBody.Frames.Quaternions.to_T_inv....570
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices....570
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.orientationConstraint....573
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.angularVelocity1....573
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.angularVelocity2....574
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.resolve1....574
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.resolve2....574
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.multipleResolve1....575
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.multipleResolve2....575
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.resolveDyade1....575
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.resolveDyade2....575
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.nullRotation....576
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.inverseRotation....576
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.relativeRotation....576
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.absoluteRotation....577
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.planarRotation....577
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.planarRotationAngle....577
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.axisRotation....578
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.axesRotations....578
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.axesRotationsAngles....579
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.smallRotation....580
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_nxy....580
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_nxz....580
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_T....581
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_T_inv....581
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_Q....582
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_T....582
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_T_inv....582
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_Q....582
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_vector....583
Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_exy....583
Modelica.Mechanics.MultiBody.Interfaces....583
Modelica.Mechanics.MultiBody.Interfaces.Frame....584
Modelica.Mechanics.MultiBody.Interfaces.Frame_a....585

Modelica.Mechanics.MultiBody.Interfaces.Frame_b....585
Modelica.Mechanics.MultiBody.Interfaces.Frame_resolve....585
Modelica.Mechanics.MultiBody.Interfaces.FlangeWithBearing....586
Modelica.Mechanics.MultiBody.Interfaces.FlangeWithBearingAdaptor....586
Modelica.Mechanics.MultiBody.Interfaces.PartialTwoFrames....587
Modelica.Mechanics.MultiBody.Interfaces.PartialTwoFramesDoubleSize....587
Modelica.Mechanics.MultiBody.Interfaces.PartialOneFrame_a....588
Modelica.Mechanics.MultiBody.Interfaces.PartialOneFrame_b....588
Modelica.Mechanics.MultiBody.Interfaces.PartialElementaryJoint....588
Modelica.Mechanics.MultiBody.Interfaces.PartialForce....589
Modelica.Mechanics.MultiBody.Interfaces.PartialLineForce....589
Modelica.Mechanics.MultiBody.Interfaces.PartialAbsoluteSensor....590
Modelica.Mechanics.MultiBody.Interfaces.PartialRelativeSensor....590
Modelica.Mechanics.MultiBody.Interfaces.PartialCutForceSensor....591
Modelica.Mechanics.MultiBody.Interfaces.PartialVisualizer....591
Modelica.Mechanics.MultiBody.Joints....591
Modelica.Mechanics.MultiBody.Joints.Prismatic....594
Modelica.Mechanics.MultiBody.Joints.ActuatedPrismatic....596
Modelica.Mechanics.MultiBody.Joints.Revolute....597
Modelica.Mechanics.MultiBody.Joints.ActuatedRevolute....599
Modelica.Mechanics.MultiBody.Joints.Cylindrical....600
Modelica.Mechanics.MultiBody.Joints.Universal....602
Modelica.Mechanics.MultiBody.Joints.Planar....603
Modelica.Mechanics.MultiBody.Joints.Spherical....605
Modelica.Mechanics.MultiBody.Joints.FreeMotion....607
Modelica.Mechanics.MultiBody.Joints.SphericalSpherical....609
Modelica.Mechanics.MultiBody.Joints.UniversalSpherical....611
Modelica.Mechanics.MultiBody.Joints.GearConstraint....613
Modelica.Mechanics.MultiBody.Joints.Assemblies....614
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUPS....616
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUSR....618
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUSP....621
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointSSR....624
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointSSP....626
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointRRR....628
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointRRP....630
Modelica.Mechanics.MultiBody.Parts....632
Modelica.Mechanics.MultiBody.Parts.Fixed....635
Modelica.Mechanics.MultiBody.Parts.FixedTranslation....636
Modelica.Mechanics.MultiBody.Parts.FixedRotation....637
Modelica.Mechanics.MultiBody.Parts.Body....639
Modelica.Mechanics.MultiBody.Parts.BodyShape....641
Modelica.Mechanics.MultiBody.Parts.BodyBox....644
Modelica.Mechanics.MultiBody.Parts.BodyCylinder....645
Modelica.Mechanics.MultiBody.Parts.PointMass....647
Modelica.Mechanics.MultiBody.Parts.Mounting1D....648
Modelica.Mechanics.MultiBody.Parts.Rotor1D....649
Modelica.Mechanics.MultiBody.Parts.BevelGear1D....650
Modelica.Mechanics.MultiBody.Sensors....651
Modelica.Mechanics.MultiBody.Sensors.AbsoluteSensor....653
Modelica.Mechanics.MultiBody.Sensors.RelativeSensor....655
Modelica.Mechanics.MultiBody.Sensors.Distance....658
Modelica.Mechanics.MultiBody.Sensors.CutForce....659
Modelica.Mechanics.MultiBody.Sensors.CutTorque....661
Modelica.Mechanics.MultiBody.Sensors.CutForceAndTorque....662
Modelica.Mechanics.MultiBody.Sensors.Power....663
Modelica.Mechanics.MultiBody.Types....663
Modelica.Mechanics.MultiBody.Types.RotationTypes....665

Modelica.Mechanics.MultiBody.Types.GravityTypes....665
Modelica.Mechanics.MultiBody.Types.Init....666
Modelica.Mechanics.MultiBody.Types.Defaults....667
Modelica.Mechanics.MultiBody.Visualizers....668
Modelica.Mechanics.MultiBody.Visualizers.FixedShape....669
Modelica.Mechanics.MultiBody.Visualizers.FixedShape2....671
Modelica.Mechanics.MultiBody.Visualizers.FixedFrame....673
Modelica.Mechanics.MultiBody.Visualizers.FixedArrow....674
Modelica.Mechanics.MultiBody.Visualizers.SignalArrow....675
Modelica.Mechanics.MultiBody.Visualizers.Advanced....676
Modelica.Mechanics.MultiBody.Visualizers.Advanced.Arrow....677
Modelica.Mechanics.MultiBody.Visualizers.Advanced.DoubleArrow....677
Modelica.Mechanics.MultiBody.Visualizers.Advanced.Shape....678
Modelica.Mechanics.Rotational....680
Modelica.Mechanics.Rotational.UsersGuide....682
Modelica.Mechanics.Rotational.UsersGuide.Overview....682
Modelica.Mechanics.Rotational.UsersGuide.FlangeConnectors....683
Modelica.Mechanics.Rotational.UsersGuide.SupportTorques....683
Modelica.Mechanics.Rotational.UsersGuide.SignConventions....684
Modelica.Mechanics.Rotational.UsersGuide.UserDefinedComponents....686
Modelica.Mechanics.Rotational.UsersGuide.RequirementsForSimulationTool....687
Modelica.Mechanics.Rotational.UsersGuide.ReleaseNotes....688
Modelica.Mechanics.Rotational.UsersGuide.Contact....689
Modelica.Mechanics.Rotational.Examples....689
Modelica.Mechanics.Rotational.Examples.First....690
Modelica.Mechanics.Rotational.Examples.Friction....690
Modelica.Mechanics.Rotational.Examples.CoupledClutches....691
Modelica.Mechanics.Rotational.Examples.LossyGearDemo1....692
Modelica.Mechanics.Rotational.Examples.LossyGearDemo2....692
Modelica.Mechanics.Rotational.Examples.ElasticBearing....693
Modelica.Mechanics.Rotational.Sensors....693
Modelica.Mechanics.Rotational.Sensors.AngleSensor....694
Modelica.Mechanics.Rotational.Sensors.SpeedSensor....694
Modelica.Mechanics.Rotational.Sensors.AccSensor....695
Modelica.Mechanics.Rotational.Sensors.RelAngleSensor....695
Modelica.Mechanics.Rotational.Sensors.RelSpeedSensor....695
Modelica.Mechanics.Rotational.Sensors.RelAccSensor....696
Modelica.Mechanics.Rotational.Sensors.TorqueSensor....696
Modelica.Mechanics.Rotational.Sensors.PowerSensor....696
Modelica.Mechanics.Rotational.Interfaces....697
Modelica.Mechanics.Rotational.Interfaces.Flange_a....697
Modelica.Mechanics.Rotational.Interfaces.Flange_b....698
Modelica.Mechanics.Rotational.Interfaces.Rigid....698
Modelica.Mechanics.Rotational.Interfaces.Compliant....699
Modelica.Mechanics.Rotational.Interfaces.TwoFlanges....699
Modelica.Mechanics.Rotational.Interfaces.Bearing....699
Modelica.Mechanics.Rotational.Interfaces.TwoFlangesAndBearing....700
Modelica.Mechanics.Rotational.Interfaces.TwoFlangesAndBearingH....700
Modelica.Mechanics.Rotational.Interfaces.FrictionBase....700
Modelica.Mechanics.Rotational.Interfaces.AbsoluteSensor....701
Modelica.Mechanics.Rotational.Interfaces.RelativeSensor....701
Modelica.Mechanics.Rotational.Interfaces.PartialSpeedDependentTorque....701
Modelica.Mechanics.Rotational.Inertia....702
Modelica.Mechanics.Rotational.IdealGear....702
Modelica.Mechanics.Rotational.IdealPlanetary....703
Modelica.Mechanics.Rotational.IdealGearR2T....703
Modelica.Mechanics.Rotational.Spring....704
Modelica.Mechanics.Rotational.Damper....704

Modelica.Mechanics.Rotational.SpringDamper 705
Modelica.Mechanics.Rotational.ElastoBacklash 705
Modelica.Mechanics.Rotational.BearingFriction 706
Modelica.Mechanics.Rotational.Clutch 708
Modelica.Mechanics.Rotational.OneWayClutch 709
Modelica.Mechanics.Rotational.Brake 711
Modelica.Mechanics.Rotational.LossyGear 712
Modelica.Mechanics.Rotational.GearEfficiency 714
Modelica.Mechanics.Rotational.Gear 714
Modelica.Mechanics.Rotational.Gear2 715
Modelica.Mechanics.Rotational.Position 716
Modelica.Mechanics.Rotational.Speed 716
Modelica.Mechanics.Rotational.Accelerate 717
Modelica.Mechanics.Rotational.Move 718
Modelica.Mechanics.Rotational.Fixed 718
Modelica.Mechanics.Rotational.Torque 719
Modelica.Mechanics.Rotational.Torque2 719
Modelica.Mechanics.Rotational.LinearSpeedDependentTorque 719
Modelica.Mechanics.Rotational.QuadraticSpeedDependentTorque 720
Modelica.Mechanics.Rotational.ConstantTorque 720
Modelica.Mechanics.Rotational.ConstantSpeed 721
Modelica.Mechanics.Rotational.TorqueStep 721
Modelica.Mechanics.Rotational.RelativeStates 722
Modelica.Mechanics.Rotational.InitializeFlange 722
Modelica.Mechanics.Rotational.Types 723
Modelica.Mechanics.Rotational.Types.Init 723
Modelica.Mechanics.Rotational.Types.InitRel 724
Modelica.Mechanics.Translational 725
Modelica.Mechanics.Translational.Examples 727
Modelica.Mechanics.Translational.Examples.SignConvention 727
Modelica.Mechanics.Translational.Examples.InitialConditions 728
Modelica.Mechanics.Translational.Examples.WhyArrows 730
Modelica.Mechanics.Translational.Examples.Accelerate 730
Modelica.Mechanics.Translational.Examples.Damper 730
Modelica.Mechanics.Translational.Examples.Oscillator 731
Modelica.Mechanics.Translational.Examples.Sensors 732
Modelica.Mechanics.Translational.Examples.Friction 732
Modelica.Mechanics.Translational.Examples.PreLoad 733
Modelica.Mechanics.Translational.Sensors 735
Modelica.Mechanics.Translational.Sensors.ForceSensor 736
Modelica.Mechanics.Translational.Sensors.PositionSensor 736
Modelica.Mechanics.Translational.Sensors.SpeedSensor 736
Modelica.Mechanics.Translational.Sensors.AccSensor 737
Modelica.Mechanics.Translational.Interfaces 737
Modelica.Mechanics.Translational.Interfaces.Flange_a 738
Modelica.Mechanics.Translational.Interfaces.Flange_b 738
Modelica.Mechanics.Translational.Interfaces.Rigid 738
Modelica.Mechanics.Translational.Interfaces.Compliant 739
Modelica.Mechanics.Translational.Interfaces.TwoFlanges 739
Modelica.Mechanics.Translational.Interfaces.AbsoluteSensor 740
Modelica.Mechanics.Translational.Interfaces.RelativeSensor 740
Modelica.Mechanics.Translational.Interfaces.FrictionBase 740
Modelica.Mechanics.Translational.SlidingMass 741
Modelica.Mechanics.Translational.Stop 741
Modelica.Mechanics.Translational.Rod 744
Modelica.Mechanics.Translational.Spring 744
Modelica.Mechanics.Translational.Damper 744
Modelica.Mechanics.Translational.SpringDamper 745

Modelica.Mechanics.Translational.ElastoGap....745
Modelica.Mechanics.Translational.Position....746
Modelica.Mechanics.Translational.Speed....746
Modelica.Mechanics.Translational.Accelerate....747
Modelica.Mechanics.Translational.Move....748
Modelica.Mechanics.Translational.Fixed....748
Modelica.Mechanics.Translational.Force....748
Modelica.Mechanics.Translational.RelativeStates....749
Modelica.Media....749
Modelica.Media.UsersGuide....750
Modelica.Media.UsersGuide.MediumUsage....751
Modelica.Media.UsersGuide.MediumUsage.BasicUsage....752
Modelica.Media.UsersGuide.MediumUsage.BalanceVolume....755
Modelica.Media.UsersGuide.MediumUsage.ShortPipe....757
Modelica.Media.UsersGuide.MediumUsage.OptionalProperties....758
Modelica.Media.UsersGuide.MediumUsage.Constants....760
Modelica.Media.UsersGuide.MediumUsage.TwoPhase....761
Modelica.Media.UsersGuide.MediumUsage.Initialization....764
Modelica.Media.UsersGuide.MediumDefinition....765
Modelica.Media.UsersGuide.MediumDefinition.BasicStructure....765
Modelica.Media.UsersGuide.MediumDefinition.BasicDefinition....769
Modelica.Media.UsersGuide.MediumDefinition.MultipleSubstances....770
Modelica.Media.UsersGuide.MediumDefinition.SpecificEnthalpyAsFunction....770
Modelica.Media.UsersGuide.MediumDefinition.StaticStateSelection....772
Modelica.Media.UsersGuide.MediumDefinition.TestOfMedium....774
Modelica.Media.UsersGuide.ReleaseNotes....775
Modelica.Media.UsersGuide.Contact....775
Modelica.Media.Examples....776
Modelica.Media.Examples.SimpleLiquidWater....777
Modelica.Media.Examples.IdealGasH2O....777
Modelica.Media.Examples.WaterIF97....777
Modelica.Media.Examples.MixtureGases....778
Modelica.Media.Examples.MoistAir....778
Modelica.Media.Examples.TwoPhaseWater....778
Modelica.Media.Examples.TwoPhaseWater.ExtendedProperties....783
Modelica.Media.Examples.TwoPhaseWater.TestTwoPhaseStates....784
Modelica.Media.Examples.TestOnly....784
Modelica.Media.Examples.TestOnly.MixIdealGasAir....784
Modelica.Media.Examples.TestOnly.FlueGas....784
Modelica.Media.Examples.TestOnly.IdealGasN2....785
Modelica.Media.Examples.TestOnly.TestMedia....785
Modelica.Media.Examples.TestOnly.TestMedia.TemplateMedium....785
Modelica.Media.Examples.Tests....785
Modelica.Media.Examples.Tests.Components....786
Modelica.Media.Examples.Tests.Components.FluidPort....786
Modelica.Media.Examples.Tests.Components.FluidPort_a....787
Modelica.Media.Examples.Tests.Components.FluidPort_b....787
Modelica.Media.Examples.Tests.Components.PortVolume....788
Modelica.Media.Examples.Tests.Components.FixedMassFlowRate....789
Modelica.Media.Examples.Tests.Components.FixedAmbient....789
Modelica.Media.Examples.Tests.Components.ShortPipe....790
Modelica.Media.Examples.Tests.Components.PartialTestModel....790
Modelica.Media.Examples.Tests.Components.PartialTestModel2....791
Modelica.Media.Examples.Tests.MediaTestModels....791
Modelica.Media.Examples.Tests.MediaTestModels.Air....791
Modelica.Media.Examples.Tests.MediaTestModels.Air.SimpleAir....792
Modelica.Media.Examples.Tests.MediaTestModels.Air.DryAirNasa....792
Modelica.Media.Examples.Tests.MediaTestModels.Air.MoistAir....793

Modelica.Media.Examples.Tests.MediaTestModels.IdealGases....793
Modelica.Media.Examples.Tests.MediaTestModels.IdealGases.Air....793
Modelica.Media.Examples.Tests.MediaTestModels.IdealGases.Nitrogen....794
Modelica.Media.Examples.Tests.MediaTestModels.IdealGases.SimpleNaturalGas....795
Modelica.Media.Examples.Tests.MediaTestModels.IdealGases.SimpleNaturalGasFixedComposition...795
Modelica.Media.Examples.Tests.MediaTestModels.Incompressible....796
Modelica.Media.Examples.Tests.MediaTestModels.Incompressible.Glycol47....796
Modelica.Media.Examples.Tests.MediaTestModels.Incompressible.Essotherm650....796
Modelica.Media.Examples.Tests.MediaTestModels.Water....797
Modelica.Media.Examples.Tests.MediaTestModels.Water.ConstantPropertyLiquidWater....797
Modelica.Media.Examples.Tests.MediaTestModels.Water.IdealSteam....798
Modelica.Media.Examples.Tests.MediaTestModels.Water.WaterIF97OnePhase_ph....798
Modelica.Media.Examples.Tests.MediaTestModels.Water.WaterIF97_pT....799
Modelica.Media.Examples.Tests.MediaTestModels.Water.WaterIF97_ph....799
Modelica.Media.Examples.Tests.MediaTestModels.LinearFluid....800
Modelica.Media.Examples.Tests.MediaTestModels.LinearFluid.LinearColdWater....800
Modelica.Media.Examples.Tests.MediaTestModels.LinearFluid.LinearWater_pT....801
Modelica.Media.Examples.SolveOneNonlinearEquation....801
Modelica.Media.Examples.SolveOneNonlinearEquation.Inverse_sine....802
Modelica.Media.Examples.SolveOneNonlinearEquation.Inverse_sh_T....802
Modelica.Media.Examples.SolveOneNonlinearEquation.InverseIncompressible_sh_T....803
Modelica.Media.Examples.SolveOneNonlinearEquation.Inverse_sh_TX....803
Modelica.Media.Interfaces....803
Modelica.Media.Interfaces.TemplateMedium....804
Modelica.Media.Interfaces.TemplateMedium.BaseProperties....807
Modelica.Media.Interfaces.TemplateMedium.ThermodynamicState....807
Modelica.Media.Interfaces.TemplateMedium.dynamicViscosity....808
Modelica.Media.Interfaces.TemplateMedium.thermalConductivity....808
Modelica.Media.Interfaces.TemplateMedium.specificEntropy....808
Modelica.Media.Interfaces.TemplateMedium.specificHeatCapacityCp....809
Modelica.Media.Interfaces.TemplateMedium.specificHeatCapacityCv....809
Modelica.Media.Interfaces.TemplateMedium.isentropicExponent....809
Modelica.Media.Interfaces.TemplateMedium.velocityOfSound....810
Modelica.Media.Interfaces.PartialMedium....810
Modelica.Media.Interfaces.PartialMedium.FluidConstants....817
Modelica.Media.Interfaces.PartialMedium.ThermodynamicState....817
Modelica.Media.Interfaces.PartialMedium.BasePropertiesRecord....817
Modelica.Media.Interfaces.PartialMedium.BaseProperties....818
Modelica.Media.Interfaces.PartialMedium.setState_pTX....819
Modelica.Media.Interfaces.PartialMedium.setState_phX....819
Modelica.Media.Interfaces.PartialMedium.setState_psX....819
Modelica.Media.Interfaces.PartialMedium.setState_dTX....820
Modelica.Media.Interfaces.PartialMedium.dynamicViscosity....820
Modelica.Media.Interfaces.PartialMedium.thermalConductivity....821
Modelica.Media.Interfaces.PartialMedium.prandtlNumber....821
Modelica.Media.Interfaces.PartialMedium.pressure....821
Modelica.Media.Interfaces.PartialMedium.temperature....821
Modelica.Media.Interfaces.PartialMedium.density....822
Modelica.Media.Interfaces.PartialMedium.specificEnthalpy....822
Modelica.Media.Interfaces.PartialMedium.specificInternalEnergy....822
Modelica.Media.Interfaces.PartialMedium.specificEntropy....823
Modelica.Media.Interfaces.PartialMedium.specificGibbsEnergy....823
Modelica.Media.Interfaces.PartialMedium.specificHelmholtzEnergy....823
Modelica.Media.Interfaces.PartialMedium.specificHeatCapacityCp....824
Modelica.Media.Interfaces.PartialMedium.heatCapacity_cp....824
Modelica.Media.Interfaces.PartialMedium.specificHeatCapacityCv....824
Modelica.Media.Interfaces.PartialMedium.heatCapacity_cv....825
Modelica.Media.Interfaces.PartialMedium.isentropicExponent....825

Modelica.Media.Interfaces.PartialMedium.isentropicEnthalpy....825
Modelica.Media.Interfaces.PartialMedium.velocityOfSound....826
Modelica.Media.Interfaces.PartialMedium.isobaricExpansionCoefficient....826
Modelica.Media.Interfaces.PartialMedium.beta....826
Modelica.Media.Interfaces.PartialMedium.isothermalCompressibility....827
Modelica.Media.Interfaces.PartialMedium.kappa....827
Modelica.Media.Interfaces.PartialMedium.density_derp_h....827
Modelica.Media.Interfaces.PartialMedium.density_derh_p....828
Modelica.Media.Interfaces.PartialMedium.density_derp_T....828
Modelica.Media.Interfaces.PartialMedium.density_derT_p....828
Modelica.Media.Interfaces.PartialMedium.density_derX....829
Modelica.Media.Interfaces.PartialMedium.molarMass....829
Modelica.Media.Interfaces.PartialMedium.specificEnthalpy_pTX....829
Modelica.Media.Interfaces.PartialMedium.density_pTX....830
Modelica.Media.Interfaces.PartialMedium.temperature_phX....830
Modelica.Media.Interfaces.PartialMedium.density_phX....830
Modelica.Media.Interfaces.PartialMedium.temperature_psX....831
Modelica.Media.Interfaces.PartialMedium.density_psX....831
Modelica.Media.Interfaces.PartialMedium.specificEnthalpy_psX....832
Modelica.Media.Interfaces.PartialMedium.Choices....832
Modelica.Media.Interfaces.PartialMedium.Choices.Init....832
Modelica.Media.Interfaces.PartialMedium.Choices.ReferenceEnthalpy....833
Modelica.Media.Interfaces.PartialMedium.Choices.ReferenceEntropy....834
Modelica.Media.Interfaces.PartialMedium.Choices.pd....834
Modelica.Media.Interfaces.PartialMedium.Choices.Th....835
Modelica.Media.Interfaces.PartialMedium.Choices.Explicit....835
Modelica.Media.Interfaces.PartialPureSubstance....836
Modelica.Media.Interfaces.PartialPureSubstance.setState_pT....839
Modelica.Media.Interfaces.PartialPureSubstance.setState_ph....839
Modelica.Media.Interfaces.PartialPureSubstance.setState_ps....840
Modelica.Media.Interfaces.PartialPureSubstance.setState_dT....840
Modelica.Media.Interfaces.PartialPureSubstance.density_ph....840
Modelica.Media.Interfaces.PartialPureSubstance.temperature_ph....841
Modelica.Media.Interfaces.PartialPureSubstance.pressure_dT....841
Modelica.Media.Interfaces.PartialPureSubstance.specificEnthalpy_dT....841
Modelica.Media.Interfaces.PartialPureSubstance.specificEnthalpy_ps....842
Modelica.Media.Interfaces.PartialPureSubstance.temperature_ps....842
Modelica.Media.Interfaces.PartialPureSubstance.density_ps....842
Modelica.Media.Interfaces.PartialPureSubstance.specificEnthalpy_pT....842
Modelica.Media.Interfaces.PartialPureSubstance.density_pT....843
Modelica.Media.Interfaces.PartialLinearFluid....843
Modelica.Media.Interfaces.PartialLinearFluid.ThermodynamicState....848
Modelica.Media.Interfaces.PartialLinearFluid.BaseProperties....848
Modelica.Media.Interfaces.PartialLinearFluid.setState_pTX....849
Modelica.Media.Interfaces.PartialLinearFluid.setState_phX....849
Modelica.Media.Interfaces.PartialLinearFluid.setState_dTX....849
Modelica.Media.Interfaces.PartialLinearFluid.setState_psX....850
Modelica.Media.Interfaces.PartialLinearFluid.pressure....850
Modelica.Media.Interfaces.PartialLinearFluid.temperature....850
Modelica.Media.Interfaces.PartialLinearFluid.density....851
Modelica.Media.Interfaces.PartialLinearFluid.specificEnthalpy....851
Modelica.Media.Interfaces.PartialLinearFluid.specificEntropy....851
Modelica.Media.Interfaces.PartialLinearFluid.specificInternalEnergy....851
Modelica.Media.Interfaces.PartialLinearFluid.specificGibbsEnergy....852
Modelica.Media.Interfaces.PartialLinearFluid.specificHelmholtzEnergy....852
Modelica.Media.Interfaces.PartialLinearFluid.velocityOfSound....852
Modelica.Media.Interfaces.PartialLinearFluid.isentropicExponent....853
Modelica.Media.Interfaces.PartialLinearFluid.isentropicEnthalpy....853

Modelica.Media.Interfaces.PartialLinearFluid.specificHeatCapacityCp....853
Modelica.Media.Interfaces.PartialLinearFluid.specificHeatCapacityCv....854
Modelica.Media.Interfaces.PartialLinearFluid.isothermalCompressibility....854
Modelica.Media.Interfaces.PartialLinearFluid.isobaricExpansionCoefficient....854
Modelica.Media.Interfaces.PartialLinearFluid.density_derp_h....854
Modelica.Media.Interfaces.PartialLinearFluid.density_derh_p....855
Modelica.Media.Interfaces.PartialLinearFluid.density_derp_T....855
Modelica.Media.Interfaces.PartialLinearFluid.density_derT_p....855
Modelica.Media.Interfaces.PartialLinearFluid.molarMass....856
Modelica.Media.Interfaces.PartialLinearFluid.T_ph....856
Modelica.Media.Interfaces.PartialLinearFluid.T_ps....856
Modelica.Media.Interfaces.PartialMixtureMedium....856
Modelica.Media.Interfaces.PartialMixtureMedium.ThermodynamicState....860
Modelica.Media.Interfaces.PartialMixtureMedium.FluidConstants....860
Modelica.Media.Interfaces.PartialMixtureMedium.gasConstant....861
Modelica.Media.Interfaces.PartialMixtureMedium.moleToMassFractions....861
Modelica.Media.Interfaces.PartialMixtureMedium.massToMoleFractions....861
Modelica.Media.Interfaces.PartialCondensingGases....862
Modelica.Media.Interfaces.PartialCondensingGases.saturationPressure....865
Modelica.Media.Interfaces.PartialCondensingGases.enthalpyOfVaporization....865
Modelica.Media.Interfaces.PartialCondensingGases.enthalpyOfLiquid....865
Modelica.Media.Interfaces.PartialCondensingGases.enthalpyOfGas....866
Modelica.Media.Interfaces.PartialCondensingGases.enthalpyOfCondensingGas....866
Modelica.Media.Interfaces.PartialTwoPhaseMedium....866
Modelica.Media.Interfaces.PartialTwoPhaseMedium.FluidLimits....871
Modelica.Media.Interfaces.PartialTwoPhaseMedium.FluidConstants....871
Modelica.Media.Interfaces.PartialTwoPhaseMedium.ThermodynamicState....872
Modelica.Media.Interfaces.PartialTwoPhaseMedium.SaturationProperties....872
Modelica.Media.Interfaces.PartialTwoPhaseMedium.BaseProperties....873
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setDewState....873
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setBubbleState....873
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_dTX....874
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_phX....874
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_psX....874
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_pTX....875
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setSat_T....875
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setSat_p....876
Modelica.Media.Interfaces.PartialTwoPhaseMedium.bubbleEnthalpy....876
Modelica.Media.Interfaces.PartialTwoPhaseMedium.dewEnthalpy....876
Modelica.Media.Interfaces.PartialTwoPhaseMedium.bubbleEntropy....877
Modelica.Media.Interfaces.PartialTwoPhaseMedium.dewEntropy....877
Modelica.Media.Interfaces.PartialTwoPhaseMedium.bubbleDensity....877
Modelica.Media.Interfaces.PartialTwoPhaseMedium.dewDensity....878
Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationPressure....878
Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationTemperature....878
Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationPressure_sat....879
Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationTemperature_sat....879
Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationTemperature_derp....879
Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationTemperature_derp_sat....880
Modelica.Media.Interfaces.PartialTwoPhaseMedium.surfaceTension....880
Modelica.Media.Interfaces.PartialTwoPhaseMedium.molarMass....880
Modelica.Media.Interfaces.PartialTwoPhaseMedium.dBubbleDensity_dPressure....881
Modelica.Media.Interfaces.PartialTwoPhaseMedium.dDewDensity_dPressure....881
Modelica.Media.Interfaces.PartialTwoPhaseMedium.dBubbleEnthalpy_dPressure....881
Modelica.Media.Interfaces.PartialTwoPhaseMedium.dDewEnthalpy_dPressure....882
Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_pTX....882
Modelica.Media.Interfaces.PartialTwoPhaseMedium.temperature_phX....882
Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_phX....883

Modelica.Media.Interfaces.PartialTwoPhaseMedium.temperature_psX....883
Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_psX....883
Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_psX....884
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_pT....884
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_ph....885
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_ps....885
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_dT....885
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_px....886
Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_Tx....886
Modelica.Media.Interfaces.PartialTwoPhaseMedium.vapourQuality....887
Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_ph....887
Modelica.Media.Interfaces.PartialTwoPhaseMedium.temperature_ph....887
Modelica.Media.Interfaces.PartialTwoPhaseMedium.pressure_dT....888
Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_dT....888
Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_ps....888
Modelica.Media.Interfaces.PartialTwoPhaseMedium.temperature_ps....889
Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_ps....889
Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_pT....890
Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_pT....890
Modelica.Media.Interfaces.PartialSimpleMedium....890
Modelica.Media.Interfaces.PartialSimpleMedium.ThermodynamicState....894
Modelica.Media.Interfaces.PartialSimpleMedium.BaseProperties....895
Modelica.Media.Interfaces.PartialSimpleMedium.setState_pTX....895
Modelica.Media.Interfaces.PartialSimpleMedium.setState_phX....895
Modelica.Media.Interfaces.PartialSimpleMedium.setState_psX....896
Modelica.Media.Interfaces.PartialSimpleMedium.setState_dTX....896
Modelica.Media.Interfaces.PartialSimpleMedium.dynamicViscosity....897
Modelica.Media.Interfaces.PartialSimpleMedium.thermalConductivity....897
Modelica.Media.Interfaces.PartialSimpleMedium.specificHeatCapacityCp....897
Modelica.Media.Interfaces.PartialSimpleMedium.specificHeatCapacityCv....898
Modelica.Media.Interfaces.PartialSimpleMedium.isentropicExponent....898
Modelica.Media.Interfaces.PartialSimpleMedium.velocityOfSound....898
Modelica.Media.Interfaces.PartialSimpleMedium.specificEnthalpy_pTX....899
Modelica.Media.Interfaces.PartialSimpleMedium.temperature_phX....899
Modelica.Media.Interfaces.PartialSimpleMedium.density_phX....899
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium....900
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.ThermodynamicState....904
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.BaseProperties....904
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.setState_pTX....904
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.setState_phX....905
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.setState_psX....905
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.setState_dTX....905
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.pressure....906
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.temperature....906
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.density....906
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificEnthalpy....907
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificInternalEnergy....907
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificEntropy....907
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificGibbsEnergy....908
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificHelmholtzEnergy....908
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.dynamicViscosity....908
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.thermalConductivity....909
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificHeatCapacityCp....909
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificHeatCapacityCv....909
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.isentropicExponent....910
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.velocityOfSound....910
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificEnthalpy_pTX....910
Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.temperature_phX....911

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.density_phX....911
Modelica.Media.Common....911
Modelica.Media.Common.SaturationProperties....917
Modelica.Media.Common.SaturationBoundaryProperties....918
Modelica.Media.Common.IF97BaseTwoPhase....918
Modelica.Media.Common.IF97PhaseBoundaryProperties....919
Modelica.Media.Common.GibbsDerivs....919
Modelica.Media.Common.HelmholtzDerivs....920
Modelica.Media.Common.TwoPhaseTransportProps....920
Modelica.Media.Common.PhaseBoundaryProperties....920
Modelica.Media.Common.NewtonDerivatives_ph....921
Modelica.Media.Common.NewtonDerivatives_ps....921
Modelica.Media.Common.NewtonDerivatives_pT....922
Modelica.Media.Common.ExtraDerivatives....922
Modelica.Media.Common.BridgmansTables....922
Modelica.Media.Common.gibbsToBridgmansTables....925
Modelica.Media.Common.helmholtzToBridgmansTables....925
Modelica.Media.Common.gibbsToBoundaryProps....926
Modelica.Media.Common.helmholtzToBoundaryProps....926
Modelica.Media.Common.cv2Phase....926
Modelica.Media.Common.cvdpT2Phase....927
Modelica.Media.Common.gibbsToExtraDerivs....927
Modelica.Media.Common.helmholtzToExtraDerivs....927
Modelica.Media.Common.Helmholtz_ph....928
Modelica.Media.Common.Helmholtz_pT....928
Modelica.Media.Common.Helmholtz_ps....928
Modelica.Media.Common.OneNonLinearEquation....929
Modelica.Media.Common.OneNonLinearEquation.f_nonlinear_Data....930
Modelica.Media.Common.OneNonLinearEquation.f_nonlinear....930
Modelica.Media.Common.OneNonLinearEquation.solve....930
Modelica.Media.Air....931
Modelica.Media.Air.SimpleAir....931
Modelica.Media.Air.DryAirNasa....936
Modelica.Media.Air.DryAirNasa.dynamicViscosity....940
Modelica.Media.Air.DryAirNasa.thermalConductivity....941
Modelica.Media.Air.MoistAir....941
Modelica.Media.Air.MoistAir.BaseProperties....948
Modelica.Media.Air.MoistAir.setState_pTX....948
Modelica.Media.Air.MoistAir.setState_phX....949
Modelica.Media.Air.MoistAir.setState_dTX....949
Modelica.Media.Air.MoistAir.Xsaturation....950
Modelica.Media.Air.MoistAir.xsaturation....950
Modelica.Media.Air.MoistAir.xsaturation_pT....950
Modelica.Media.Air.MoistAir.massFraction_pTphi....951
Modelica.Media.Air.MoistAir.relativeHumidity_pTX....951
Modelica.Media.Air.MoistAir.relativeHumidity....952
Modelica.Media.Air.MoistAir.gasConstant....952
Modelica.Media.Air.MoistAir.gasConstant_X....952
Modelica.Media.Air.MoistAir.saturationPressureLiquid....953
Modelica.Media.Air.MoistAir.saturationPressureLiquid_der....953
Modelica.Media.Air.MoistAir.sublimationPressureIce....954
Modelica.Media.Air.MoistAir.sublimationPressureIce_der....954
Modelica.Media.Air.MoistAir.saturationPressure....954
Modelica.Media.Air.MoistAir.saturationPressure_der....955
Modelica.Media.Air.MoistAir.saturationTemperature....955
Modelica.Media.Air.MoistAir.enthalpyOfVaporization....956
Modelica.Media.Air.MoistAir.HeatCapacityOfWater....956
Modelica.Media.Air.MoistAir.enthalpyOfLiquid....956

Modelica.Media.Air.MoistAir.enthalpyOfGas 957
Modelica.Media.Air.MoistAir.enthalpyOfCondensingGas 957
Modelica.Media.Air.MoistAir.enthalpyOfWater 957
Modelica.Media.Air.MoistAir.enthalpyOfWater_der 958
Modelica.Media.Air.MoistAir.pressure 958
Modelica.Media.Air.MoistAir.temperature 959
Modelica.Media.Air.MoistAir.T_phX 959
Modelica.Media.Air.MoistAir.density 959
Modelica.Media.Air.MoistAir.specificEnthalpy 960
Modelica.Media.Air.MoistAir.h_pTX 960
Modelica.Media.Air.MoistAir.h_pTX_der 961
Modelica.Media.Air.MoistAir.specificInternalEnergy 961
Modelica.Media.Air.MoistAir.specificInternalEnergy_pTX 962
Modelica.Media.Air.MoistAir.specificInternalEnergy_pTX_der 962
Modelica.Media.Air.MoistAir.specificEntropy 962
Modelica.Media.Air.MoistAir.specificGibbsEnergy 963
Modelica.Media.Air.MoistAir.specificHelmholtzEnergy 963
Modelica.Media.Air.MoistAir.specificHeatCapacityCp 964
Modelica.Media.Air.MoistAir.specificHeatCapacityCv 964
Modelica.Media.Air.MoistAir.dynamicViscosity 964
Modelica.Media.Air.MoistAir.thermalConductivity 965
Modelica.Media.Air.MoistAir.Utilities 965
Modelica.Media.Air.MoistAir.Utilities.spliceFunction 965
Modelica.Media.Air.MoistAir.Utilities.spliceFunction_der 966
Modelica.Media.Air.MoistAir.PsychrometricData 966
Modelica.Media.CompressibleLiquids 968
Modelica.Media.CompressibleLiquids.Common 969
Modelica.Media.CompressibleLiquids.Common.LinearWater_pT 969
Modelica.Media.CompressibleLiquids.LinearColdWater 973
Modelica.Media.CompressibleLiquids.LinearColdWater.dynamicViscosity 977
Modelica.Media.CompressibleLiquids.LinearColdWater.thermalConductivity 977
Modelica.Media.CompressibleLiquids.LinearWater_pT_Ambient 977
Modelica.Media.IdealGases 977
Modelica.Media.IdealGases.Common 981
Modelica.Media.IdealGases.Common.DataRecord 982
Modelica.Media.IdealGases.Common.SingleGasNasa 983
Modelica.Media.IdealGases.Common.SingleGasNasa.ThermodynamicState 988
Modelica.Media.IdealGases.Common.SingleGasNasa.FluidConstants 988
Modelica.Media.IdealGases.Common.SingleGasNasa.BaseProperties 989
Modelica.Media.IdealGases.Common.SingleGasNasa.setState_pTX 990
Modelica.Media.IdealGases.Common.SingleGasNasa.setState_phX 990
Modelica.Media.IdealGases.Common.SingleGasNasa.setState_psX 990
Modelica.Media.IdealGases.Common.SingleGasNasa.setState_dTX 991
Modelica.Media.IdealGases.Common.SingleGasNasa.pressure 991
Modelica.Media.IdealGases.Common.SingleGasNasa.temperature 991
Modelica.Media.IdealGases.Common.SingleGasNasa.density 991
Modelica.Media.IdealGases.Common.SingleGasNasa.specificEnthalpy 992
Modelica.Media.IdealGases.Common.SingleGasNasa.specificInternalEnergy 992
Modelica.Media.IdealGases.Common.SingleGasNasa.specificEntropy 992
Modelica.Media.IdealGases.Common.SingleGasNasa.specificGibbsEnergy 993
Modelica.Media.IdealGases.Common.SingleGasNasa.specificHelmholtzEnergy 993
Modelica.Media.IdealGases.Common.SingleGasNasa.specificHeatCapacityCp 993
Modelica.Media.IdealGases.Common.SingleGasNasa.specificHeatCapacityCv 993
Modelica.Media.IdealGases.Common.SingleGasNasa.isentropicExponent 994
Modelica.Media.IdealGases.Common.SingleGasNasa.velocityOfSound 994
Modelica.Media.IdealGases.Common.SingleGasNasa.isentropicEnthalpyApproximation 994
Modelica.Media.IdealGases.Common.SingleGasNasa.isentropicEnthalpy 995
Modelica.Media.IdealGases.Common.SingleGasNasa.isobaricExpansionCoefficient 995

Modelica.Media.IdealGases.Common.SingleGasNasa.isothermalCompressibility....996
Modelica.Media.IdealGases.Common.SingleGasNasa.density_derp_T....996
Modelica.Media.IdealGases.Common.SingleGasNasa.density_derT_p....996
Modelica.Media.IdealGases.Common.SingleGasNasa.density_derX....997
Modelica.Media.IdealGases.Common.SingleGasNasa.cp_T....997
Modelica.Media.IdealGases.Common.SingleGasNasa.cp_Tlow....997
Modelica.Media.IdealGases.Common.SingleGasNasa.cp_Tlow_der....997
Modelica.Media.IdealGases.Common.SingleGasNasa.h_T....998
Modelica.Media.IdealGases.Common.SingleGasNasa.h_T_der....998
Modelica.Media.IdealGases.Common.SingleGasNasa.h_Tlow....999
Modelica.Media.IdealGases.Common.SingleGasNasa.h_Tlow_der....999
Modelica.Media.IdealGases.Common.SingleGasNasa.s0_T....1000
Modelica.Media.IdealGases.Common.SingleGasNasa.s0_Tlow....1000
Modelica.Media.IdealGases.Common.SingleGasNasa.dynamicViscosityLowPressure....1000
Modelica.Media.IdealGases.Common.SingleGasNasa.dynamicViscosity....1001
Modelica.Media.IdealGases.Common.SingleGasNasa.thermalConductivityEstimate....1001
Modelica.Media.IdealGases.Common.SingleGasNasa.thermalConductivity....1002
Modelica.Media.IdealGases.Common.SingleGasNasa.molarMass....1002
Modelica.Media.IdealGases.Common.SingleGasNasa.T_h....1002
Modelica.Media.IdealGases.Common.SingleGasNasa.T_ps....1003
Modelica.Media.IdealGases.Common.MixtureGasNasa....1003
Modelica.Media.IdealGases.Common.MixtureGasNasa.BaseProperties....1007
Modelica.Media.IdealGases.Common.MixtureGasNasa.setState_pTX....1007
Modelica.Media.IdealGases.Common.MixtureGasNasa.setState_phX....1008
Modelica.Media.IdealGases.Common.MixtureGasNasa.setState_psX....1008
Modelica.Media.IdealGases.Common.MixtureGasNasa.setState_dTX....1008
Modelica.Media.IdealGases.Common.MixtureGasNasa.pressure....1009
Modelica.Media.IdealGases.Common.MixtureGasNasa.temperature....1009
Modelica.Media.IdealGases.Common.MixtureGasNasa.density....1009
Modelica.Media.IdealGases.Common.MixtureGasNasa.specificEnthalpy....1010
Modelica.Media.IdealGases.Common.MixtureGasNasa.specificInternalEnergy....1010
Modelica.Media.IdealGases.Common.MixtureGasNasa.specificEntropy....1010
Modelica.Media.IdealGases.Common.MixtureGasNasa.specificGibbsEnergy....1010
Modelica.Media.IdealGases.Common.MixtureGasNasa.specificHelmholtzEnergy....1011
Modelica.Media.IdealGases.Common.MixtureGasNasa.h_TX....1011
Modelica.Media.IdealGases.Common.MixtureGasNasa.h_TX_der....1012
Modelica.Media.IdealGases.Common.MixtureGasNasa.gasConstant....1012
Modelica.Media.IdealGases.Common.MixtureGasNasa.specificHeatCapacityCp....1012
Modelica.Media.IdealGases.Common.MixtureGasNasa.specificHeatCapacityCv....1013
Modelica.Media.IdealGases.Common.MixtureGasNasa.MixEntropy....1013
Modelica.Media.IdealGases.Common.MixtureGasNasa.s_TX....1013
Modelica.Media.IdealGases.Common.MixtureGasNasa.isentropicExponent....1014
Modelica.Media.IdealGases.Common.MixtureGasNasa.velocityOfSound....1014
Modelica.Media.IdealGases.Common.MixtureGasNasa.isentropicEnthalpyApproximation....1014
Modelica.Media.IdealGases.Common.MixtureGasNasa.isentropicEnthalpy....1014
Modelica.Media.IdealGases.Common.MixtureGasNasa.gasMixtureViscosity....1015
Modelica.Media.IdealGases.Common.MixtureGasNasa.dynamicViscosity....1015
Modelica.Media.IdealGases.Common.MixtureGasNasa.mixtureViscosityChung....1016
Modelica.Media.IdealGases.Common.MixtureGasNasa.lowPressureThermalConductivity....1017
Modelica.Media.IdealGases.Common.MixtureGasNasa.thermalConductivity....1017
Modelica.Media.IdealGases.Common.MixtureGasNasa.isobaricExpansionCoefficient....1018
Modelica.Media.IdealGases.Common.MixtureGasNasa.isothermalCompressibility....1018
Modelica.Media.IdealGases.Common.MixtureGasNasa.density_derp_T....1018
Modelica.Media.IdealGases.Common.MixtureGasNasa.density_derT_p....1019
Modelica.Media.IdealGases.Common.MixtureGasNasa.density_derX....1019
Modelica.Media.IdealGases.Common.MixtureGasNasa.molarMass....1019
Modelica.Media.IdealGases.Common.MixtureGasNasa.T_hX....1019
Modelica.Media.IdealGases.Common.MixtureGasNasa.T_psX....1020

Modelica.Media.IdealGases.Common.FluidData....1020
Modelica.Media.IdealGases.Common.SingleGasesData....1033
Modelica.Media.IdealGases.SingleGases....1037
Modelica.Media.IdealGases.SingleGases.Ar....1038
Modelica.Media.IdealGases.SingleGases.CH4....1039
Modelica.Media.IdealGases.SingleGases.CH3OH....1039
Modelica.Media.IdealGases.SingleGases.CO....1040
Modelica.Media.IdealGases.SingleGases.CO2....1040
Modelica.Media.IdealGases.SingleGases.C2H2_vinylidene....1041
Modelica.Media.IdealGases.SingleGases.C2H4....1041
Modelica.Media.IdealGases.SingleGases.C2H5OH....1042
Modelica.Media.IdealGases.SingleGases.C2H6....1042
Modelica.Media.IdealGases.SingleGases.C3H6_propylene....1043
Modelica.Media.IdealGases.SingleGases.C3H8....1043
Modelica.Media.IdealGases.SingleGases.C3H8O_1propanol....1044
Modelica.Media.IdealGases.SingleGases.C4H8_1_butene....1044
Modelica.Media.IdealGases.SingleGases.C4H10_n_butane....1045
Modelica.Media.IdealGases.SingleGases.C5H10_1_pentene....1045
Modelica.Media.IdealGases.SingleGases.C5H12_n_pentane....1046
Modelica.Media.IdealGases.SingleGases.C6H6....1046
Modelica.Media.IdealGases.SingleGases.C6H12_1_hexene....1047
Modelica.Media.IdealGases.SingleGases.C6H14_n_hexane....1047
Modelica.Media.IdealGases.SingleGases.C7H14_1_heptene....1048
Modelica.Media.IdealGases.SingleGases.C7H16_n_heptane....1048
Modelica.Media.IdealGases.SingleGases.C8H10_ethylbenz....1049
Modelica.Media.IdealGases.SingleGases.C8H18_n_octane....1049
Modelica.Media.IdealGases.SingleGases.CL2....1050
Modelica.Media.IdealGases.SingleGases.F2....1050
Modelica.Media.IdealGases.SingleGases.H2....1051
Modelica.Media.IdealGases.SingleGases.H2O....1051
Modelica.Media.IdealGases.SingleGases.He....1052
Modelica.Media.IdealGases.SingleGases.NH3....1052
Modelica.Media.IdealGases.SingleGases.NO....1053
Modelica.Media.IdealGases.SingleGases.NO2....1053
Modelica.Media.IdealGases.SingleGases.N2....1054
Modelica.Media.IdealGases.SingleGases.N2O....1054
Modelica.Media.IdealGases.SingleGases.Ne....1055
Modelica.Media.IdealGases.SingleGases.O2....1055
Modelica.Media.IdealGases.SingleGases.SO2....1056
Modelica.Media.IdealGases.SingleGases.SO3....1056
Modelica.Media.IdealGases.MixtureGases....1057
Modelica.Media.IdealGases.MixtureGases.CombustionAir....1057
Modelica.Media.IdealGases.MixtureGases.AirSteam....1058
Modelica.Media.IdealGases.MixtureGases.FlueGasLambdaOnePlus....1058
Modelica.Media.IdealGases.MixtureGases.FlueGasSixComponents....1058
Modelica.Media.IdealGases.MixtureGases.SimpleNaturalGas....1058
Modelica.Media.IdealGases.MixtureGases.SimpleNaturalGasFixedComposition....1058
Modelica.Media.Incompressible....1058
Modelica.Media.Incompressible.Common....1059
Modelica.Media.Incompressible.Common.BaseProps_Tpoly....1059
Modelica.Media.Incompressible.TableBased....1060
Modelica.Media.Incompressible.TableBased.invertTemp....1066
Modelica.Media.Incompressible.TableBased.BaseProperties....1066
Modelica.Media.Incompressible.TableBased.setState_pTX....1067
Modelica.Media.Incompressible.TableBased.setState_dTX....1067
Modelica.Media.Incompressible.TableBased.setState_pT....1067
Modelica.Media.Incompressible.TableBased.setState_phX....1068
Modelica.Media.Incompressible.TableBased.setState_ph....1068

Modelica.Media.Incompressible.TableBased.setState_psX....1068
Modelica.Media.Incompressible.TableBased.setState_ps....1069
Modelica.Media.Incompressible.TableBased.specificHeatCapacityCv....1069
Modelica.Media.Incompressible.TableBased.specificHeatCapacityCp....1069
Modelica.Media.Incompressible.TableBased.dynamicViscosity....1070
Modelica.Media.Incompressible.TableBased.thermalConductivity....1070
Modelica.Media.Incompressible.TableBased.s_T....1070
Modelica.Media.Incompressible.TableBased.specificEntropy....1071
Modelica.Media.Incompressible.TableBased.h_T....1071
Modelica.Media.Incompressible.TableBased.h_T_der....1071
Modelica.Media.Incompressible.TableBased.h_pT....1071
Modelica.Media.Incompressible.TableBased.density_T....1072
Modelica.Media.Incompressible.TableBased.temperature....1072
Modelica.Media.Incompressible.TableBased.pressure....1072
Modelica.Media.Incompressible.TableBased.density....1073
Modelica.Media.Incompressible.TableBased.specificEnthalpy....1073
Modelica.Media.Incompressible.TableBased.specificInternalEnergy....1073
Modelica.Media.Incompressible.TableBased.T_ph....1073
Modelica.Media.Incompressible.TableBased.T_ps....1074
Modelica.Media.Incompressible.TableBased.Polynomials_Temp....1074
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.evaluate....1075
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.derivative....1075
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.derivativeValue....1075
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.secondDerivativeValue....1076
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.integral....1076
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.integralValue....1076
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.fitting....1077
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.evaluate_der....1077
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.integralValue_der....1077
Modelica.Media.Incompressible.TableBased.Polynomials_Temp.derivativeValue_der....1078
Modelica.Media.Incompressible.Examples....1078
Modelica.Media.Incompressible.Examples.Glycol47....1078
Modelica.Media.Incompressible.Examples.Essotherm650....1079
Modelica.Media.Incompressible.Examples.TestGlycol....1079
Modelica.Media.Water....1079
Modelica.Media.Water.ConstantPropertyLiquidWater....1083
Modelica.Media.Water.IdealSteam....1083
Modelica.Media.Water.WaterIF97OnePhase_ph....1083
Modelica.Media.Water.WaterIF97_ph....1083
Modelica.Media.Water.WaterIF97_base....1083
Modelica.Media.Water.WaterIF97_base.ThermodynamicState....1088
Modelica.Media.Water.WaterIF97_base.BaseProperties....1088
Modelica.Media.Water.WaterIF97_base.density_ph....1089
Modelica.Media.Water.WaterIF97_base.temperature_ph....1089
Modelica.Media.Water.WaterIF97_base.temperature_ps....1089
Modelica.Media.Water.WaterIF97_base.density_ps....1090
Modelica.Media.Water.WaterIF97_base.pressure_dT....1090
Modelica.Media.Water.WaterIF97_base.specificEnthalpy_dT....1090
Modelica.Media.Water.WaterIF97_base.specificEnthalpy_pT....1091
Modelica.Media.Water.WaterIF97_base.specificEnthalpy_ps....1091
Modelica.Media.Water.WaterIF97_base.density_pT....1091
Modelica.Media.Water.WaterIF97_base.setDewState....1092
Modelica.Media.Water.WaterIF97_base.setBubbleState....1092
Modelica.Media.Water.WaterIF97_base.dynamicViscosity....1092
Modelica.Media.Water.WaterIF97_base.thermalConductivity....1093
Modelica.Media.Water.WaterIF97_base.surfaceTension....1093
Modelica.Media.Water.WaterIF97_base.pressure....1093
Modelica.Media.Water.WaterIF97_base.temperature....1093

Modelica.Media.Water.WaterIF97_base.density 1094
Modelica.Media.Water.WaterIF97_base.specificEnthalpy 1094
Modelica.Media.Water.WaterIF97_base.specificInternalEnergy 1094
Modelica.Media.Water.WaterIF97_base.specificGibbsEnergy 1095
Modelica.Media.Water.WaterIF97_base.specificHelmholtzEnergy 1095
Modelica.Media.Water.WaterIF97_base.specificEntropy 1095
Modelica.Media.Water.WaterIF97_base.specificHeatCapacityCp 1095
Modelica.Media.Water.WaterIF97_base.specificHeatCapacityCv 1096
Modelica.Media.Water.WaterIF97_base.isentropicExponent 1096
Modelica.Media.Water.WaterIF97_base.isothermalCompressibility 1096
Modelica.Media.Water.WaterIF97_base.isobaricExpansionCoefficient 1097
Modelica.Media.Water.WaterIF97_base.velocityOfSound 1097
Modelica.Media.Water.WaterIF97_base.isentropicEnthalpy 1097
Modelica.Media.Water.WaterIF97_base.density_derh_p 1098
Modelica.Media.Water.WaterIF97_base.density_derp_h 1098
Modelica.Media.Water.WaterIF97_base.bubbleEnthalpy 1098
Modelica.Media.Water.WaterIF97_base.dewEnthalpy 1098
Modelica.Media.Water.WaterIF97_base.bubbleEntropy 1099
Modelica.Media.Water.WaterIF97_base.dewEntropy 1099
Modelica.Media.Water.WaterIF97_base.bubbleDensity 1099
Modelica.Media.Water.WaterIF97_base.dewDensity 1100
Modelica.Media.Water.WaterIF97_base.saturationTemperature 1100
Modelica.Media.Water.WaterIF97_base.saturationTemperature_derp 1100
Modelica.Media.Water.WaterIF97_base.saturationPressure 1100
Modelica.Media.Water.WaterIF97_base.dBubbleDensity_dPressure 1101
Modelica.Media.Water.WaterIF97_base.dDewDensity_dPressure 1101
Modelica.Media.Water.WaterIF97_base.dBubbleEnthalpy_dPressure 1101
Modelica.Media.Water.WaterIF97_base.dDewEnthalpy_dPressure 1102
Modelica.Media.Water.WaterIF97_base.setState_dTX 1102
Modelica.Media.Water.WaterIF97_base.setState_phX 1102
Modelica.Media.Water.WaterIF97_base.setState_psX 1103
Modelica.Media.Water.WaterIF97_base.setState_pTX 1103
Modelica.Media.Water.WaterIF97_fixedregion 1103
Modelica.Media.Water.WaterIF97_fixedregion.ThermodynamicState 1108
Modelica.Media.Water.WaterIF97_fixedregion.BaseProperties 1108
Modelica.Media.Water.WaterIF97_fixedregion.density_ph 1109
Modelica.Media.Water.WaterIF97_fixedregion.temperature_ph 1109
Modelica.Media.Water.WaterIF97_fixedregion.temperature_ps 1109
Modelica.Media.Water.WaterIF97_fixedregion.density_ps 1110
Modelica.Media.Water.WaterIF97_fixedregion.pressure_dT 1110
Modelica.Media.Water.WaterIF97_fixedregion.specificEnthalpy_dT 1111
Modelica.Media.Water.WaterIF97_fixedregion.specificEnthalpy_pT 1111
Modelica.Media.Water.WaterIF97_fixedregion.specificEnthalpy_ps 1111
Modelica.Media.Water.WaterIF97_fixedregion.density_pT 1112
Modelica.Media.Water.WaterIF97_fixedregion.setDewState 1112
Modelica.Media.Water.WaterIF97_fixedregion.setBubbleState 1112
Modelica.Media.Water.WaterIF97_fixedregion.dynamicViscosity 1113
Modelica.Media.Water.WaterIF97_fixedregion.thermalConductivity 1113
Modelica.Media.Water.WaterIF97_fixedregion.surfaceTension 1113
Modelica.Media.Water.WaterIF97_fixedregion.pressure 1113
Modelica.Media.Water.WaterIF97_fixedregion.temperature 1114
Modelica.Media.Water.WaterIF97_fixedregion.density 1114
Modelica.Media.Water.WaterIF97_fixedregion.specificEnthalpy 1114
Modelica.Media.Water.WaterIF97_fixedregion.specificInternalEnergy 1115
Modelica.Media.Water.WaterIF97_fixedregion.specificGibbsEnergy 1115
Modelica.Media.Water.WaterIF97_fixedregion.specificHelmholtzEnergy 1115
Modelica.Media.Water.WaterIF97_fixedregion.specificEntropy 1115
Modelica.Media.Water.WaterIF97_fixedregion.specificHeatCapacityCp 1116

Modelica.Media.Water.WaterIF97_fixedregion.specificHeatCapacityCv....1116
Modelica.Media.Water.WaterIF97_fixedregion.isentropicExponent....1116
Modelica.Media.Water.WaterIF97_fixedregion.isothermalCompressibility....1117
Modelica.Media.Water.WaterIF97_fixedregion.isobaricExpansionCoefficient....1117
Modelica.Media.Water.WaterIF97_fixedregion.velocityOfSound....1117
Modelica.Media.Water.WaterIF97_fixedregion.isentropicEnthalpy....1117
Modelica.Media.Water.WaterIF97_fixedregion.density_derh_p....1118
Modelica.Media.Water.WaterIF97_fixedregion.density_derp_h....1118
Modelica.Media.Water.WaterIF97_fixedregion.bubbleEnthalpy....1118
Modelica.Media.Water.WaterIF97_fixedregion.dewEnthalpy....1119
Modelica.Media.Water.WaterIF97_fixedregion.bubbleEntropy....1119
Modelica.Media.Water.WaterIF97_fixedregion.dewEntropy....1119
Modelica.Media.Water.WaterIF97_fixedregion.bubbleDensity....1120
Modelica.Media.Water.WaterIF97_fixedregion.dewDensity....1120
Modelica.Media.Water.WaterIF97_fixedregion.saturationTemperature....1120
Modelica.Media.Water.WaterIF97_fixedregion.saturationTemperature_derp....1120
Modelica.Media.Water.WaterIF97_fixedregion.saturationPressure....1121
Modelica.Media.Water.WaterIF97_fixedregion.dBubbleDensity_dPressure....1121
Modelica.Media.Water.WaterIF97_fixedregion.dDewDensity_dPressure....1121
Modelica.Media.Water.WaterIF97_fixedregion.dBubbleEnthalpy_dPressure....1122
Modelica.Media.Water.WaterIF97_fixedregion.dDewEnthalpy_dPressure....1122
Modelica.Media.Water.WaterIF97_fixedregion.setState_dTX....1122
Modelica.Media.Water.WaterIF97_fixedregion.setState_phX....1123
Modelica.Media.Water.WaterIF97_fixedregion.setState_psX....1123
Modelica.Media.Water.WaterIF97_fixedregion.setState_pTX....1123
Modelica.Media.Water.WaterIF97_R1ph....1124
Modelica.Media.Water.WaterIF97_R2ph....1124
Modelica.Media.Water.WaterIF97_R3ph....1124
Modelica.Media.Water.WaterIF97_R4ph....1124
Modelica.Media.Water.WaterIF97_R5ph....1124
Modelica.Media.Water.WaterIF97_R1pT....1124
Modelica.Media.Water.WaterIF97_R2pT....1124
Modelica.Media.Water.IF97_Utilities....1125
Modelica.Media.Water.IF97_Utilities.BaseIF97....1129
Modelica.Media.Water.IF97_Utilities.BaseIF97.IterationData....1133
Modelica.Media.Water.IF97_Utilities.BaseIF97.data....1133
Modelica.Media.Water.IF97_Utilities.BaseIF97.getTstar....1135
Modelica.Media.Water.IF97_Utilities.BaseIF97.getpstar....1135
Modelica.Media.Water.IF97_Utilities.BaseIF97.critical....1135
Modelica.Media.Water.IF97_Utilities.BaseIF97.triple....1136
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions....1136
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.boundary23ofT....1140
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.boundary23ofp....1140
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hlowerofp5....1140
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hupperofp5....1141
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.slowerofp5....1141
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.supperofp5....1141
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hlowerofp1....1141
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hupperofp1....1142
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.slowerofp1....1142
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.supperofp1....1142
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hlowerofp2....1143
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hupperofp2....1143
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.slowerofp2....1143
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.supperofp2....1144
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.d1n....1144
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.d2n....1144
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dhot1ofp....1144

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dupper1ofT....1145
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hl_p_R4b....1145
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hv_p_R4b....1145
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.sl_p_R4b....1146
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.sv_p_R4b....1146
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhol_p_R4b....1146
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhov_p_R4b....1147
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.boilingcurve_p....1147
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dewcurve_p....1147
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hvl_p....1147
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hl_p....1148
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hv_p....1148
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hvl_p_der....1148
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhovl_p....1149
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhol_p....1149
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhov_p....1149
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhovl_p_der....1149
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.sl_p....1150
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.sv_p....1150
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhol_T....1150
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhov_T....1151
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.region_ph....1151
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.region_ps....1151
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.region_pT....1152
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.region_dT....1152
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hvl_dp....1152
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dhl_dp....1153
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dhv_dp....1153
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.drhovl_dp....1153
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.drhol_dp....1153
Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.drhov_dp....1154
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic....1154
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g1....1156
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g2....1157
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g2metastable....1157
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.f3....1157
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g5....1157
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.gibbs....1158
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g1pitau....1158
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g2pitau....1158
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g5pitau....1159
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.f3deltatau....1159
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tph1....1160
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps1....1160
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tph2....1160
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps2a....1161
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps2b....1161
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps2c....1161
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps2....1161
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tsat....1162
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.dtsatofp....1162
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tsat_der....1162
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.psat....1163
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.dptofT....1163
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.psat_der....1163
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.p1_hs....1164
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.h2ab_s....1164
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.p2a_hs....1165

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.p2b_hs....1165
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.p2c_hs....1166
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.h3ab_p....1166
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.T3a_ph....1167
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.T3b_ph....1167
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.v3a_ph....1168
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.v3b_ph....1168
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.T3a_ps....1169
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.T3b_ps....1169
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.v3a_ps....1170
Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.v3b_ps....1170
Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries....1171
Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries.pmIceI_T....1171
Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries.pmIceIII_T....1171
Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries.pmIceV_T....1172
Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries.sublimationPressure_T....1172
Modelica.Media.Water.IF97_Utilities.BaseIF97.Transport....1173
Modelica.Media.Water.IF97_Utilities.BaseIF97.Transport.visc_dTp....1174
Modelica.Media.Water.IF97_Utilities.BaseIF97.Transport.cond_dTp....1174
Modelica.Media.Water.IF97_Utilities.BaseIF97.Transport.surfaceTension....1174
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic....1175
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofpT1....1176
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.handsofpT1....1176
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofps1....1177
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofpT2....1177
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.handsofpT2....1177
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofps2....1177
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofdT3....1178
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofps3....1178
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofpsdt3....1178
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofps4....1179
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofpT5....1179
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.water_hisentropic....1179
Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.water_hisentropic_dyn....1180
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses....1180
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.fixdT....1181
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dofp13....1182
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dofp23....1182
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dofpt3....1182
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dtofph3....1183
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dtofps3....1183
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dtofpsdt3....1183
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.pofdt125....1184
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.tofph5....1184
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.tofps5....1185
Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.tofpst5....1185
Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion....1185
Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion.waterR1_pT....1186
Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion.waterR2_pT....1186
Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion.waterR3_dT....1187
Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion.waterR5_pT....1187
Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase....1187
Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterLiq_p....1188
Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterVap_p....1188
Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterSat_ph....1189
Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterR4_ph....1189
Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterR4_dT....1189
Modelica.Media.Water.IF97_Utilities.BaseIF97.extraDerivs_ph....1190

Modelica.Media.Water.IF97_Utilities.BaseIF97.extraDerivs_pT ..1190
Modelica.Media.Water.IF97_Utilities.iter ..1190
Modelica.Media.Water.IF97_Utilities.waterBaseProp_ph ..1190
Modelica.Media.Water.IF97_Utilities.waterBaseProp_ps ..1191
Modelica.Media.Water.IF97_Utilities.rho_props_ps ..1191
Modelica.Media.Water.IF97_Utilities.rho_ps ..1192
Modelica.Media.Water.IF97_Utilities.T_props_ps ..1192
Modelica.Media.Water.IF97_Utilities.T_ps ..1192
Modelica.Media.Water.IF97_Utilities.h_props_ps ..1193
Modelica.Media.Water.IF97_Utilities.h_ps ..1193
Modelica.Media.Water.IF97_Utilities.phase_ps ..1193
Modelica.Media.Water.IF97_Utilities.phase_ph ..1194
Modelica.Media.Water.IF97_Utilities.phase_dT ..1194
Modelica.Media.Water.IF97_Utilities.rho_props_ph ..1194
Modelica.Media.Water.IF97_Utilities.rho_ph ..1194
Modelica.Media.Water.IF97_Utilities.rho_ph_der ..1195
Modelica.Media.Water.IF97_Utilities.T_props_ph ..1195
Modelica.Media.Water.IF97_Utilities.T_ph ..1196
Modelica.Media.Water.IF97_Utilities.T_ph_der ..1196
Modelica.Media.Water.IF97_Utilities.s_props_ph ..1196
Modelica.Media.Water.IF97_Utilities.s_ph ..1197
Modelica.Media.Water.IF97_Utilities.s_ph_der ..1197
Modelica.Media.Water.IF97_Utilities.cv_props_ph ..1197
Modelica.Media.Water.IF97_Utilities.cv_ph ..1198
Modelica.Media.Water.IF97_Utilities.regionAssertReal ..1198
Modelica.Media.Water.IF97_Utilities.cp_props_ph ..1198
Modelica.Media.Water.IF97_Utilities.cp_ph ..1199
Modelica.Media.Water.IF97_Utilities.beta_props_ph ..1199
Modelica.Media.Water.IF97_Utilities.beta_ph ..1199
Modelica.Media.Water.IF97_Utilities.kappa_props_ph ..1200
Modelica.Media.Water.IF97_Utilities.kappa_ph ..1200
Modelica.Media.Water.IF97_Utilities.velocityOfSound_props_ph ..1200
Modelica.Media.Water.IF97_Utilities.velocityOfSound_ph ..1201
Modelica.Media.Water.IF97_Utilities.isentropicExponent_props_ph ..1201
Modelica.Media.Water.IF97_Utilities.isentropicExponent_ph ..1201
Modelica.Media.Water.IF97_Utilities.ddph_props ..1202
Modelica.Media.Water.IF97_Utilities.ddph ..1202
Modelica.Media.Water.IF97_Utilities.ddhp_props ..1202
Modelica.Media.Water.IF97_Utilities.ddhp ..1203
Modelica.Media.Water.IF97_Utilities.waterBaseProp_pT ..1203
Modelica.Media.Water.IF97_Utilities.rho_props_pT ..1203
Modelica.Media.Water.IF97_Utilities.rho_pT ..1204
Modelica.Media.Water.IF97_Utilities.h_props_pT ..1204
Modelica.Media.Water.IF97_Utilities.h_pT ..1204
Modelica.Media.Water.IF97_Utilities.h_pT_der ..1205
Modelica.Media.Water.IF97_Utilities.rho_pT_der ..1205
Modelica.Media.Water.IF97_Utilities.s_props_pT ..1206
Modelica.Media.Water.IF97_Utilities.s_pT ..1206
Modelica.Media.Water.IF97_Utilities.cv_props_pT ..1206
Modelica.Media.Water.IF97_Utilities.cv_pT ..1207
Modelica.Media.Water.IF97_Utilities.cp_props_pT ..1207
Modelica.Media.Water.IF97_Utilities.cp_pT ..1207
Modelica.Media.Water.IF97_Utilities.beta_props_pT ..1208
Modelica.Media.Water.IF97_Utilities.beta_pT ..1208
Modelica.Media.Water.IF97_Utilities.kappa_props_pT ..1208
Modelica.Media.Water.IF97_Utilities.kappa_pT ..1209
Modelica.Media.Water.IF97_Utilities.velocityOfSound_props_pT ..1209
Modelica.Media.Water.IF97_Utilities.velocityOfSound_pT ..1209

Modelica.Media.Water.IF97_Utilities.isentropicExponent_props_pT....1210
Modelica.Media.Water.IF97_Utilities.isentropicExponent_pT....1210
Modelica.Media.Water.IF97_Utilities.waterBaseProp_dT....1210
Modelica.Media.Water.IF97_Utilities.h_props_dT....1211
Modelica.Media.Water.IF97_Utilities.h_dT....1211
Modelica.Media.Water.IF97_Utilities.h_dT_der....1211
Modelica.Media.Water.IF97_Utilities.p_props_dT....1212
Modelica.Media.Water.IF97_Utilities.p_dT....1212
Modelica.Media.Water.IF97_Utilities.p_dT_der....1212
Modelica.Media.Water.IF97_Utilities.s_props_dT....1213
Modelica.Media.Water.IF97_Utilities.s_dT....1213
Modelica.Media.Water.IF97_Utilities.cv_props_dT....1213
Modelica.Media.Water.IF97_Utilities.cv_dT....1214
Modelica.Media.Water.IF97_Utilities.cp_props_dT....1214
Modelica.Media.Water.IF97_Utilities.cp_dT....1214
Modelica.Media.Water.IF97_Utilities.beta_props_dT....1215
Modelica.Media.Water.IF97_Utilities.beta_dT....1215
Modelica.Media.Water.IF97_Utilities.kappa_props_dT....1216
Modelica.Media.Water.IF97_Utilities.kappa_dT....1216
Modelica.Media.Water.IF97_Utilities.velocityOfSound_props_dT....1216
Modelica.Media.Water.IF97_Utilities.velocityOfSound_dT....1217
Modelica.Media.Water.IF97_Utilities.isentropicExponent_props_dT....1217
Modelica.Media.Water.IF97_Utilities.isentropicExponent_dT....1217
Modelica.Media.Water.IF97_Utilities.hl_p....1218
Modelica.Media.Water.IF97_Utilities.hv_p....1218
Modelica.Media.Water.IF97_Utilities.sl_p....1218
Modelica.Media.Water.IF97_Utilities.sv_p....1218
Modelica.Media.Water.IF97_Utilities.rhol_T....1219
Modelica.Media.Water.IF97_Utilities.rhov_T....1219
Modelica.Media.Water.IF97_Utilities.rhol_p....1219
Modelica.Media.Water.IF97_Utilities.rhov_p....1220
Modelica.Media.Water.IF97_Utilities.dynamicViscosity....1220
Modelica.Media.Water.IF97_Utilities.thermalConductivity....1220
Modelica.Media.Water.IF97_Utilities.surfaceTension....1221
Modelica.Media.Water.IF97_Utilities.isentropicEnthalpy....1221
Modelica.Media.Water.IF97_Utilities.isentropicEnthalpy_props....1221
Modelica.Media.Water.IF97_Utilities.isentropicEnthalpy_der....1222
Modelica.Media.Water.IF97_Utilities.dynamicIsentropicEnthalpy....1222
Modelica.SIunits....1222
Modelica.SIunits.UsersGuide....1258
Modelica.SIunits.UsersGuide.HowToUseSIunits....1258
Modelica.SIunits.UsersGuide.Conventions....1260
Modelica.SIunits.UsersGuide.Literature....1260
Modelica.SIunits.UsersGuide.Contact....1261
Modelica.SIunits.Conversions....1261
Modelica.SIunits.Conversions.NonSIunits....1262
Modelica.SIunits.Conversions.to_degC....1264
Modelica.SIunits.Conversions.from_degC....1264
Modelica.SIunits.Conversions.to_degF....1264
Modelica.SIunits.Conversions.from_degF....1264
Modelica.SIunits.Conversions.to_degRk....1265
Modelica.SIunits.Conversions.from_degRk....1265
Modelica.SIunits.Conversions.to_deg....1265
Modelica.SIunits.Conversions.from_deg....1266
Modelica.SIunits.Conversions.to_rpm....1266
Modelica.SIunits.Conversions.from_rpm....1266
Modelica.SIunits.Conversions.to_kmh....1266
Modelica.SIunits.Conversions.from_kmh....1267

Modelica.SIunits.Conversions.to_day 1267
Modelica.SIunits.Conversions.from_day 1267
Modelica.SIunits.Conversions.to_hour 1268
Modelica.SIunits.Conversions.from_hour 1268
Modelica.SIunits.Conversions.to_minute 1268
Modelica.SIunits.Conversions.from_minute 1268
Modelica.SIunits.Conversions.to_litre 1269
Modelica.SIunits.Conversions.from_litre 1269
Modelica.SIunits.Conversions.to_kWh 1269
Modelica.SIunits.Conversions.from_kWh 1270
Modelica.SIunits.Conversions.to_bar 1270
Modelica.SIunits.Conversions.from_bar 1270
Modelica.SIunits.Conversions.to_gps 1270
Modelica.SIunits.Conversions.from_gps 1271
Modelica.SIunits.Conversions.ConversionIcon 1271
Modelica.StateGraph 1271
Modelica.StateGraph.UsersGuide 1273
Modelica.StateGraph.UsersGuide.OverView 1273
Modelica.StateGraph.UsersGuide.FirstExample 1279
Modelica.StateGraph.UsersGuide.ApplicationExample 1279
Modelica.StateGraph.UsersGuide.ReleaseNotes 1282
Modelica.StateGraph.UsersGuide.Literature 1282
Modelica.StateGraph.UsersGuide.Contact 1283
Modelica.StateGraph.Examples 1283
Modelica.StateGraph.Examples.FirstExample 1284
Modelica.StateGraph.Examples.FirstExample_Variant2 1284
Modelica.StateGraph.Examples.FirstExample_Variant3 1284
Modelica.StateGraph.Examples.ExecutionPaths 1285
Modelica.StateGraph.Examples.ShowCompositeStep 1285
Modelica.StateGraph.Examples.ShowExceptions 1286
Modelica.StateGraph.Examples.ControlledTanks 1286
Modelica.StateGraph.Examples.Utilities 1287
Modelica.StateGraph.Examples.Utilities.TankController 1288
Modelica.StateGraph.Examples.Utilities.MakeProduct 1288
Modelica.StateGraph.Examples.Utilities.inflow 1289
Modelica.StateGraph.Examples.Utilities.outflow 1289
Modelica.StateGraph.Examples.Utilities.valve 1289
Modelica.StateGraph.Examples.Utilities.Tank 1289
Modelica.StateGraph.Examples.Utilities.Source 1290
Modelica.StateGraph.Examples.Utilities.CompositeStep 1290
Modelica.StateGraph.Examples.Utilities.CompositeStep1 1290
Modelica.StateGraph.Examples.Utilities.CompositeStep2 1291
Modelica.StateGraph.Interfaces 1291
Modelica.StateGraph.Interfaces.Step_in 1292
Modelica.StateGraph.Interfaces.Step_out 1292
Modelica.StateGraph.Interfaces.Transition_in 1292
Modelica.StateGraph.Interfaces.Transition_out 1293
Modelica.StateGraph.Interfaces.CompositeStep_resume 1293
Modelica.StateGraph.Interfaces.CompositeStep_suspend 1293
Modelica.StateGraph.Interfaces.CompositeStepStatePort_in 1293
Modelica.StateGraph.Interfaces.CompositeStepStatePort_out 1294
Modelica.StateGraph.Interfaces.PartialStep 1294
Modelica.StateGraph.Interfaces.PartialTransition 1294
Modelica.StateGraph.Interfaces.PartialStateGraphIcon 1295
Modelica.StateGraph.Interfaces.CompositeStepState 1295
Modelica.StateGraph.InitialStep 1295
Modelica.StateGraph.InitialStepWithSignal 1296
Modelica.StateGraph.Step 1296

Modelica.StateGraph.StepWithSignal....1296
Modelica.StateGraph.Transition....1297
Modelica.StateGraph.TransitionWithSignal....1297
Modelica.StateGraph.Alternative....1298
Modelica.StateGraph.Parallel....1298
Modelica.StateGraph.PartialCompositeStep....1299
Modelica.StateGraph.StateGraphRoot....1299
Modelica.StateGraph.Temporary....1299
Modelica.StateGraph.Temporary.anyTrue....1300
Modelica.StateGraph.Temporary.allTrue....1300
Modelica.StateGraph.Temporary.RadioButton....1301
Modelica.StateGraph.Temporary.NumericValue....1301
Modelica.StateGraph.Temporary.IndicatorLamp....1301
Modelica.Thermal....1302
Modelica.Thermal.FluidHeatFlow....1302
Modelica.Thermal.FluidHeatFlow.Examples....1303
Modelica.Thermal.FluidHeatFlow.Examples.SimpleCooling....1304
Modelica.Thermal.FluidHeatFlow.Examples.ParallelCooling....1305
Modelica.Thermal.FluidHeatFlow.Examples.IndirectCooling....1306
Modelica.Thermal.FluidHeatFlow.Examples.PumpAndValve....1307
Modelica.Thermal.FluidHeatFlow.Examples.PumpDropOut....1307
Modelica.Thermal.FluidHeatFlow.Examples.ParallelPumpDropOut....1308
Modelica.Thermal.FluidHeatFlow.Examples.OneMass....1309
Modelica.Thermal.FluidHeatFlow.Examples.TwoMass....1309
Modelica.Thermal.FluidHeatFlow.Examples.Utilities....1310
Modelica.Thermal.FluidHeatFlow.Examples.Utilities.DoubleRamp....1311
Modelica.Thermal.FluidHeatFlow.Components....1311
Modelica.Thermal.FluidHeatFlow.Components.IsolatedPipe....1312
Modelica.Thermal.FluidHeatFlow.Components.HeatedPipe....1313
Modelica.Thermal.FluidHeatFlow.Components.Valve....1314
Modelica.Thermal.FluidHeatFlow.Interfaces....1314
Modelica.Thermal.FluidHeatFlow.Interfaces.FlowPort....1315
Modelica.Thermal.FluidHeatFlow.Interfaces.FlowPort_a....1316
Modelica.Thermal.FluidHeatFlow.Interfaces.FlowPort_b....1316
Modelica.Thermal.FluidHeatFlow.Interfaces.Partials....1317
Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.SimpleFriction....1317
Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.TwoPort....1318
Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.Ambient....1318
Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.AbsoluteSensor....1319
Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.RelativeSensor....1319
Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.FlowSensor....1320
Modelica.Thermal.FluidHeatFlow.Media....1320
Modelica.Thermal.FluidHeatFlow.Media.Medium....1321
Modelica.Thermal.FluidHeatFlow.Media.Air_30degC....1321
Modelica.Thermal.FluidHeatFlow.Media.Air_70degC....1322
Modelica.Thermal.FluidHeatFlow.Media.Water....1322
Modelica.Thermal.FluidHeatFlow.Sensors....1323
Modelica.Thermal.FluidHeatFlow.Sensors.pSensor....1324
Modelica.Thermal.FluidHeatFlow.Sensors.TSensor....1324
Modelica.Thermal.FluidHeatFlow.Sensors.dpSensor....1325
Modelica.Thermal.FluidHeatFlow.Sensors.dTSensor....1325
Modelica.Thermal.FluidHeatFlow.Sensors.m_flowSensor....1326
Modelica.Thermal.FluidHeatFlow.Sensors.V_flowSensor....1326
Modelica.Thermal.FluidHeatFlow.Sensors.H_flowSensor....1326
Modelica.Thermal.FluidHeatFlow.Sources....1327
Modelica.Thermal.FluidHeatFlow.Sources.Ambient....1328
Modelica.Thermal.FluidHeatFlow.Sources.PrescribedAmbient....1328
Modelica.Thermal.FluidHeatFlow.Sources.AbsolutePressure....1329

Modelica.Thermal.FluidHeatFlow.Sources.ConstantVolumeFlow....1329
Modelica.Thermal.FluidHeatFlow.Sources.PrescribedVolumeFlow....1329
Modelica.Thermal.FluidHeatFlow.Sources.ConstantPressureIncrease....1330
Modelica.Thermal.FluidHeatFlow.Sources.PrescribedPressureIncrease....1331
Modelica.Thermal.FluidHeatFlow.Sources.IdealPump....1331
Modelica.Thermal.HeatTransfer....1332
Modelica.Thermal.HeatTransfer.Examples....1334
Modelica.Thermal.HeatTransfer.Examples.TwoMasses....1335
Modelica.Thermal.HeatTransfer.Examples.ControlledTemperature....1335
Modelica.Thermal.HeatTransfer.Examples.Motor....1336
Modelica.Thermal.HeatTransfer.Interfaces....1337
Modelica.Thermal.HeatTransfer.Interfaces.HeatPort....1337
Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a....1338
Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b....1338
Modelica.Thermal.HeatTransfer.Interfaces.Element1D....1338
Modelica.Thermal.HeatTransfer.HeatCapacitor....1339
Modelica.Thermal.HeatTransfer.ThermalConductor....1340
Modelica.Thermal.HeatTransfer.Convection....1341
Modelica.Thermal.HeatTransfer.BodyRadiation....1342
Modelica.Thermal.HeatTransfer.FixedTemperature....1343
Modelica.Thermal.HeatTransfer.PrescribedTemperature....1343
Modelica.Thermal.HeatTransfer.FixedHeatFlow....1344
Modelica.Thermal.HeatTransfer.PrescribedHeatFlow....1344
Modelica.Thermal.HeatTransfer.TemperatureSensor....1345
Modelica.Thermal.HeatTransfer.RelTemperatureSensor....1345
Modelica.Thermal.HeatTransfer.HeatFlowSensor....1345
Modelica.Thermal.HeatTransfer.Celsius....1346
Modelica.Thermal.HeatTransfer.Celsius.ToKelvin....1346
Modelica.Thermal.HeatTransfer.Celsius.FromKelvin....1346
Modelica.Thermal.HeatTransfer.Celsius.FixedTemperature....1347
Modelica.Thermal.HeatTransfer.Celsius.PrescribedTemperature....1347
Modelica.Thermal.HeatTransfer.Celsius.TemperatureSensor....1347
Modelica.Thermal.HeatTransfer.Fahrenheit....1348
Modelica.Thermal.HeatTransfer.Fahrenheit.ToKelvin....1348
Modelica.Thermal.HeatTransfer.Fahrenheit.FromKelvin....1348
Modelica.Thermal.HeatTransfer.Fahrenheit.FixedTemperature....1349
Modelica.Thermal.HeatTransfer.Fahrenheit.PrescribedTemperature....1349
Modelica.Thermal.HeatTransfer.Fahrenheit.TemperatureSensor....1350
Modelica.Thermal.HeatTransfer.Rankine....1350
Modelica.Thermal.HeatTransfer.Rankine.ToKelvin....1350
Modelica.Thermal.HeatTransfer.Rankine.FromKelvin....1351
Modelica.Thermal.HeatTransfer.Rankine.FixedTemperature....1351
Modelica.Thermal.HeatTransfer.Rankine.PrescribedTemperature....1352
Modelica.Thermal.HeatTransfer.Rankine.TemperatureSensor....1352
Modelica.Utilities....1352
Modelica.Utilities.UsersGuide....1353
Modelica.Utilities.UsersGuide.ImplementationNotes....1353
Modelica.Utilities.UsersGuide.ReleaseNotes....1354
Modelica.Utilities.UsersGuide.Contact....1355
Modelica.Utilities.Examples....1355
Modelica.Utilities.Examples.calculator....1356
Modelica.Utilities.Examples.expression....1356
Modelica.Utilities.Examples.readRealParameter....1358
Modelica.Utilities.Examples.readRealParameterModel....1359
Modelica.Utilities.Files....1359
Modelica.Utilities.Files.list....1360
Modelica.Utilities.Files.copy....1361
Modelica.Utilities.Files.move....1361

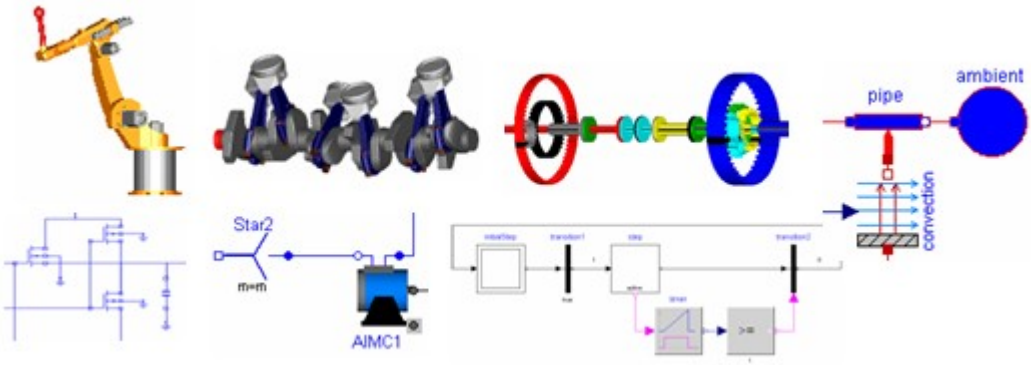
Modelica.Utilities.Files.remove....1362
Modelica.Utilities.Files.removeFile....1362
Modelica.Utilities.Files.createDirectory....1363
Modelica.Utilities.Files.exist....1363
Modelica.Utilities.Files.assertNew....1364
Modelica.Utilities.Files.fullPathName....1364
Modelica.Utilities.Files.splitPathName....1365
Modelica.Utilities.Files.temporaryFileName....1366
Modelica.Utilities.Streams....1366
Modelica.Utilities.Streams.print....1367
Modelica.Utilities.Streams.readFile....1368
Modelica.Utilities.Streams.readLine....1368
Modelica.Utilities.Streams.countLines....1369
Modelica.Utilities.Streams.error....1369
Modelica.Utilities.Streams.close....1370
Modelica.Utilities.Strings....1370
Modelica.Utilities.Strings.length....1372
Modelica.Utilities.Strings.substring....1372
Modelica.Utilities.Strings.repeat....1373
Modelica.Utilities.Strings.compare....1374
Modelica.Utilities.Strings.isEqual....1374
Modelica.Utilities.Strings.count....1375
Modelica.Utilities.Strings.find....1375
Modelica.Utilities.Strings.findLast....1376
Modelica.Utilities.Strings.replace....1377
Modelica.Utilities.Strings.sort....1377
Modelica.Utilities.Strings.scanToken....1378
Modelica.Utilities.Strings.scanReal....1380
Modelica.Utilities.Strings.scanInteger....1380
Modelica.Utilities.Strings.scanBoolean....1381
Modelica.Utilities.Strings.scanString....1382
Modelica.Utilities.Strings.scanIdentifier....1382
Modelica.Utilities.Strings.scanDelimiter....1383
Modelica.Utilities.Strings.scanNoToken....1384
Modelica.Utilities.Strings.syntaxError....1384
Modelica.Utilities.Strings.Advanced....1385
Modelica.Utilities.Strings.Advanced.scanReal....1386
Modelica.Utilities.Strings.Advanced.scanInteger....1387
Modelica.Utilities.Strings.Advanced.scanString....1388
Modelica.Utilities.Strings.Advanced.scanIdentifier....1388
Modelica.Utilities.Strings.Advanced.skipWhiteSpace....1389
Modelica.Utilities.Strings.Advanced.skipLineComments....1390
Modelica.Utilities.System....1390
Modelica.Utilities.System.getWorkDirectory....1391
Modelica.Utilities.System.setWorkDirectory....1391
Modelica.Utilities.System.getEnvironmentVariable....1391
Modelica.Utilities.System.setEnvironmentVariable....1391
Modelica.Utilities.System.command....1392
Modelica.Utilities.System.exit....1392
Modelica.Utilities.Types....1392
Modelica.Utilities.Types.Compare....1393
Modelica.Utilities.Types.FileType....1393
Modelica.Utilities.Types.TokenType....1394
Modelica.Utilities.Types.TokenValue....1395

Modelica

Modelica Standard Library

Information

Package **Modelica** is a **standardized** and **free** package that is developed together with the Modelica language from the Modelica Association, see http://www.Modelica.org. It is also called **Modelica Standard Library**. It provides model components in many domains that are based on standardized interface definitions. Some typical examples are shown in the next figure:



For an introduction, have especially a look at:

- Overview provides an overview of the Modelica Standard Library inside the User's Guide.
- Release Notes summarizes the changes of new versions of this package.
- Contact lists the contributors of the Modelica Standard Library.
- ModelicaStandardLibrary.pdf is the complete documentation of the library in pdf format.
- The **Examples** packages in the various libraries, demonstrate how to use the components of the corresponding sublibrary.

This version of the Modelica Standard Library consists of

- **739** models and blocks, and
- **539** functions

that are directly usable (= number of public, non-partial classes).

Copyright © 1998-2007, Modelica Association.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
UsersGuide	User's Guide of Modelica library
Blocks	Library of basic input/output control blocks (continuous, discrete, logical, table blocks)
Constants	Library of mathematical constants and constants of nature (e.g., pi, eps, R, sigma)
Electrical	Library of electrical models (analog, digital, machines, multi-phase)
Icons	Library of icons
Math	Library of mathematical functions (e.g., sin, cos) and of functions operating on vectors and matrices
Mechanics	Library of 1-dim. and 3-dim. mechanical components (multi-body, rotational, translational)

Media	Library of media property models
SIunits	Library of type and unit definitions based on SI units according to ISO 31-1992
StateGraph	Library of hierarchical state machine components to model discrete event and reactive systems
Thermal	Library of thermal system components to model heat transfer and simple thermo-fluid pipe flow
Utilities	Library of utility functions dedicated to scripting (operating on files, streams, strings, system)

Modelica.UsersGuide

User's Guide of Modelica library

Package **Modelica** is a **standardized** and **pre-defined** package that is developed together with the Modelica language from the Modelica Association, see http://www.Modelica.org. It is also called **Modelica Standard Library**. It provides constants, types, connectors, partial models and model components in various disciplines.

This is a short **User's Guide** for the overall library. Some of the main sublibraries have their own User's Guides that can be accessed by the following links:

Digital	Library for digital electrical components based on the VHDL standard (2-,3-,4-,9-valued logic)
MultiBody	Library to model 3-dimensional mechanical systems
Rotational	Library to model 1-dimensional mechanical systems
Media	Property models of media
SIunits	Type definitions based on SI units according to ISO 31-1992
StateGraph	Library to model discrete event and reactive systems by hierarchical state machines
Utilities	Utility functions especially for scripting (Files, Streams, Strings, System)

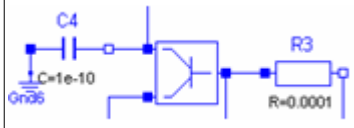
Package Content

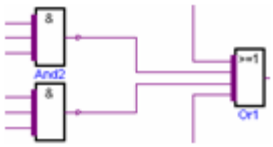
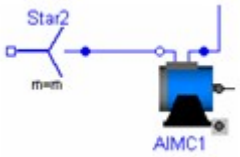
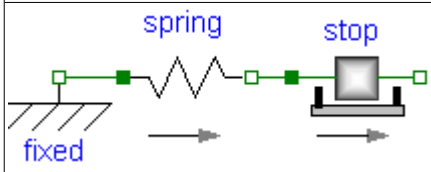
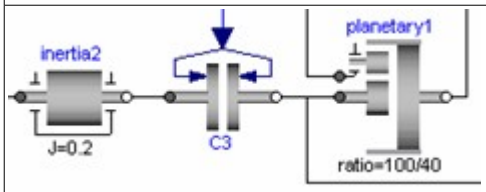
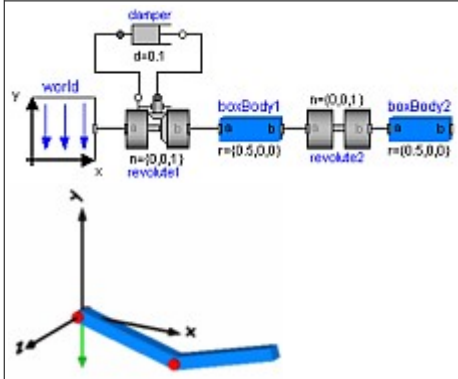
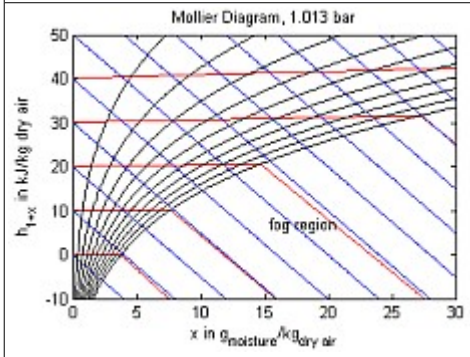
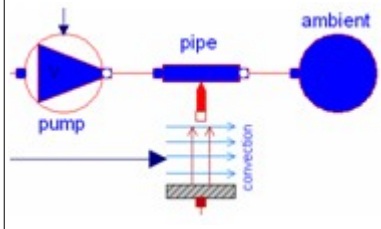
Name	Description
Overview	Overview of Modelica Library
Connectors	Connectors
Conventions	Conventions
ReleaseNotes	Release notes
Contact	Contact
ModelicaLicense	Modelica License (Version 1.1 of June 30, 2000)

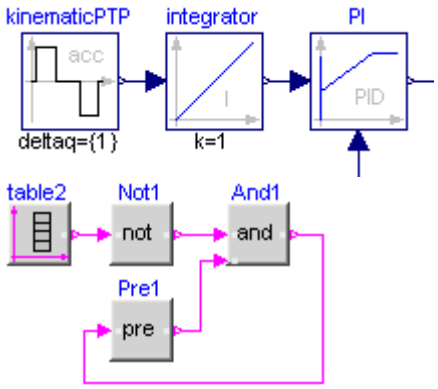
Modelica.UsersGuide.Overview

Overview of Modelica Library

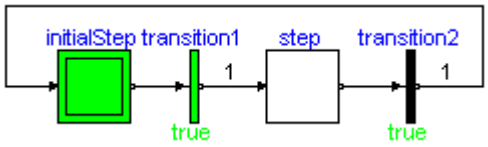
The Modelica Standard Library consists of the following main sub-libraries:

Library Components	Description
	Analog Analog electric and electronic components, such as resistor, capacitor, transformers, diodes, transistors, transmission lines, switches, sources, sensors.

	Digital Digital electrical components based on the VHDL standard, like basic logic blocks with 9-value logic, delays, gates, sources, converters between 2-, 3-, 4-, and 9-valued logic.
	Machines Electrical asynchronous-, synchronous-, and DC-machines (motors and generators) as well as 3-phase transformers.
	Translational 1-dim. mechanical, translational systems, e.g., sliding mass, mass with stops, spring, damper.
	Rotational 1-dim. mechanical, rotational systems, e.g., inertias, gears, planetary gears, convenient definition of speed/torque dependent friction (clutches, brakes, bearings, ..)
	MultiBody 3-dim. mechanical systems consisting of joints, bodies, force and sensor elements. Joints can be driven by drive trains defined by 1-dim. mechanical system library (Rotational). Every component has a default animation. Components can be arbitrarily connected together.
	Media Large media library providing models and functions to compute media properties, such as h = h(p,T), d = d(p,T), for the following media: • 1240 gases and mixtures between these gases. • incompressible, table based liquids (h = h(T), etc.). • compressible liquids • dry and moist air • high precision model for water (IF97).
	FluidHeatFlow, HeatTransfer Simple thermo-fluid pipe flow, especially to model cooling of machines with air or water (pipes, pumps, valves, ambient, sensors, sources) and lumped heat transfer with heat capacitors, thermal conductors, convection, body radiation, sources and sensors.



Blocks
Input/output blocks to model block diagrams and logical networks, e.g.,
integerator, PI, PID, transfer function, linear state space system, sampler, unit delay, discrete transfer function, and/or blocks, timer, hysteresis, nonlinear and routing blocks, sources, tables.



StateGraph
Hierarchical state machines with the same modeling power as Statecharts.
Modelica is used as synchronous action language, i.e. deterministic
behavior is guaranteed (not the case for Statecharts)

```
A = [1,2,3;
     3,4,5;
     2,1,4];
b = {10,22,12};
x = Matrices.solve(A,b);
Matrices.eigenValues(A);
```

Math, Utilities
Functions operating on vectors and matrices, such as for solving

linear systems, eigen and singular values etc., and functions operating on strings, streams, files, e.g.,
to copy and remove a file or sort a vector of strings.

Modelica.UsersGuide.Connectors

Connectors

The Modelica standard library defines the most important **elementary connectors** in various domains. If any possible, a user should utilize these connectors in order that components from the Modelica Standard Library and from other libraries can be combined without problems. The following elementary connectors are defined (potential variables are connector variables without the flow attribute, flow variables are connector variables that have the flow attribute):

domain	pot. variables	flow variables	connector definition	icons
electrical analog	electrical potential	electrical current	Modelica.Electrical.Analog.Interfaces Pin, PositivePin, NegativePin	
electrical multi-phase	vector of electrical pins		Modelica.Electrical.MultiPhase.Interfaces Plug, PositivePlug, NegativePlug	
electrical sphace phasor	2 electrical potentials	2 electrical currents	Modelica.Electrical.Machines.Interfaces SpacePhasor	
electrical digital	Integer (1..9)	---	Modelica.Electrical.Digital.Interfaces DigitalSignal, DigitalInput, DigitalOutput	
translational	distance	cut-force	Modelica.Mechanics.Translational.Interfaces Flange_a, Flange_b	
rotational	angle	cut-torque	Modelica.Mechanics.Rotational.Interfaces Flange_a, Flange_b	
3-dim. mechanics	position vector orientation object	cut-force vector cut-torque vector	Modelica.Mechanics.MultiBody.Interfaces Frame, Frame_a, Frame_b, Frame_resolve	

simple fluid flow	pressure specific enthalpy	mass flow rate enthalpy flow rate	Modelica.Thermal.FluidHeatFlow.Interfaces FlowPort, FlowPort_a, FlowPort_b	
heat transfer	temperature	heat flow rate	Modelica.Thermal.HeatTransfer.Interfaces HeatPort, HeatPort_a, HeatPort_b	
block diagram	Real variable Integer variable Boolean variable	---	Modelica.Blocks.Interfaces RealSignal, RealInput, RealOutput IntegerSignal, IntegerInput, IntegerOutput BooleanSignal, BooleanInput, BooleanOutput	
state machine	Boolean variables (occupied, set, available, reset)	---	Modelica.StateGraph.Interfaces Step_in, Step_out, Transition_in, Transition_out	
Connectors from libraries that will be included in one of the next releases of package Modelica				
thermo fluid flow	pressure specific enthalpy mass fractions	mass flow rate enthalpy flow rate subst. mass flow rates	Modelica_Fluid.Interfaces FluidPort, FluidPort_a, FluidPort_b	
magnetic	magnetic potential	magnetic flux	Magnetic.Interfaces MagneticPort, PositiveMagneticPort, NegativeMagneticPort	
Connectors from other libraries				
hydraulic	pressure	volume flow rate	HyLibLight.Interfaces Port_A, Port_b	
pneumatic	pressure	mass flow rate	PneuLibLight.Interfaces Port_1, Port_2	

In all domains, usually 2 connectors are defined. The variable declarations are **identical**, only the icons are different in order that it is easy to distinguish connectors of the same domain that are attached at the same component.

Modelica supports also hierarchical connectors, in a similar way as hierarchical models. As a result, it is, e.g., possible, to collect elementary connectors together. For example, an electrical plug consisting of two electrical pins can be defined as:

```
connector Plug
   import Modelica.Electrical.Analog.Interfaces;
   Interfaces.PositivePin phase;
   Interfaces.NegativePin ground;
end Plug;
```

With one connect(..) equation, either two plugs can be connected (and therefore implicitly also the phase and ground pins) or a Pin connector can be directly connected to the phase or ground of a Plug connector, such as "connect(resistor.p, plug.phase)".

Modelica.UsersGuide.Conventions

Conventions

Note, in the html documentation of any Modelica library, the headings "h1, h2, h3" should not be used, because they are utilized from the automatically generated documentation/headings.

Additional headings in the html documentation should start with "h4".

In the Modelica package the following conventions are used:

1. Class and instance names are written in upper and lower case letters, e.g., "ElectricCurrent". An underscore is only used at the end of a name to characterize a lower or upper index, e.g., "pin_a".

2. **Class names** start always with an upper case letter.

3. **Instance names**, i.e., names of component instances and of variables (with the exception of constants), start usually with a lower case letter with only a few exceptions if this is common sense (such as "T" for a temperature variable).

4. **Constant names**, i.e., names of variables declared with the "constant" prefix, follow the usual naming conventions (= upper and lower case letters) and start usually with an upper case letter, e.g. UniformGravity, SteadyState.

5. The two connectors of a domain that have identical declarations and different icons are usually distinguished by "_a", "_b" or "_p", "_n", e.g., Flange_a/Flange_b, HeatPort_a, HeatPort_b.

6. The **instance name** of a component is always displayed in its icon (= text string "%name") in **blue color**. A connector class has the instance name definition in the diagram layer and not in the icon layer. **Parameter** values, e.g., resistance, mass, gear ratio, are displayed in the icon in **black color** in a smaller font size as the instance name.

7. A main package has usually the following subpackages:
   - **UsersGuide** containing an overall description of the library and how to use it.
   - **Examples** containing models demonstrating the usage of the library.
   - **Interfaces** containing connectors and partial models.
   - **Types** containing type, enumeration and choice definitions.

Enumerations are defined in the Modelica language since release 2.0. However, they are not yet supported in the most important Modelica simulation environment Dymola. For this reason, this language element is not used in the Modelica standard library. Instead, enumerations are emulated with packages and constants. For example, the enumeration

```
type Init = enumeration (NoInit, InitializeStates, SteadyState);

parameter Init initType = Init.NoInit "Type of initialization";
```

is emulated in the following way:

```
package  Init "Enumeration emulation"
   extends  Modelica.Icons.Enumeration;

   constant  Integer NoInit=1;
   constant  Integer InitializeStates=2;
   constant  Integer SteadyState=3;

   type  Temp
      extends  Modelica.Icons.TypeInteger;
      annotation  (choices(
          choice=Init.NoInit           "NoInit (no initialization)",
          choice=Init.InitializeStates "InitializeStates (initialize
states)",
          choice=Init.SteadyState      "SteadyState (initialize in steady
state)"));
   end  Temp;
end  Init;
```

```
parameter Init.Temp initType = Init.NoInit;
```

Modelica.UsersGuide.ReleaseNotes

Release notes

This section summarizes the changes that have been performed on the Modelica standard library.

- Version 2.2.2 (Aug. 12, 2007)
- Version 2.2.1 (March 24, 2006)
- Version 2.2 (April 6, 2005)
- Version 2.1 (Nov. 11, 2004)
- Version 1.6 (June 21, 2004)
- Version 1.5 (Dec. 16, 2002)
- Version 1.4 (June 28, 2001 and previous versions)

Maintenance of the Modelica Standard Library is performed with three branches on the subversion server of the Modelica Association:

Released branch

Example: "Modelica/tags/V2_2_1/Modelica"

This branch contains the released Modelica versions (e.g. version 2.2.1), where all available test cases and compatibility checks with other Modelica libraries have been performed on the respective release. This version is usually shipped with a Modelica modelling and simulation environment and utilized by a Modelica user.

Development branch

Example: "Modelica/trunk/Modelica"

This branch contains the actual development version, i.e., all bug fixes and new features based on the last Modelica release. New features should have been tested before including them. However, the exhaustive tests for a new version are (usually) not performed. This version is usually only be used by the developers of the Modelica Standard Library and is not utilized by Modelica users.

Maintenance branch

Example: "Modelica/branches/maintenance/2.2.1/Modelica"

This branch contains the released Modelica version (e.g. version 2.2.1) where all bug fixes since this release date are included (up to a new release, when becoming available; i.e., after a new release, the previous maintenance versions are no longer changed). These bug fixes might be not yet tested with all test cases or with other Modelica libraries. The goal is that a vendor may take this version at any time for a new release of its software, in order to incorporate the latest bug fixes, without changing the version number of the Modelica Standard Library.

Incorporation of bug fixes (subversion "commit") shall be performed in the following way:

- One person is fixing the bug and another person is checking whether the fix is fine.
- It is up to the library developer, whether he opens a new branch for testing and then merges it with the "head" maintenance branch or not.
- Every change to the maintenance branch has to be done at the development branch (see above) as well.
- Every change to the maintenance branch requires introducing a description of the bug fix under Modelica.UsersGuide.ReleaseNotes.<release-number>_bugFixes.

- Every change to the maintenance branch requires changing the date of the Modelica library in the version annotation.
  When including the library in a distribution, the vendor has to add the subversion build number of the corresponding release in the version annotation. Example:

  ```
  annotation(version="2.2.1", versionBuild="436", versionDate="2007-05-13")
  ```

  The goal is to include the version build and version date information automatically from the subversion server, but this is not yet the case.
- If time does not permit, a vendor makes the bug fix in its local version and then has to include it in the maintenance version. It would be best to make these changes at a new branch in order to get a unique release number.

A valid "commit" to the maintenance branch may contain one or more of the following changes.

- Correcting an equation.
- Correcting attributes quantity/unit/defaultUnit in a declaration.
- Improving/fixing the documentation.
- Introducing a new name in the public section of a class (model, package, ...) or in any section of a partial class is **not** allowed. Since otherwise, a user might use this new name and when storing its model and loading it with an older build-version, an error would occur.
- Introducing a new name in the protected section of a non-partial class should only be done if absolutely necessary to fix a bug. The problem is that this might be non-backward compatible, because a user might already extend from this class and already using the same name.

Package Content

Name	Description
Version_2_2_2	Version 2.2.2 (Aug. 12, 2007)
Version_2_2_1	Version 2.2.1 (March 24, 2006)
Version_2_2	Version 2.2 (April 6, 2005)
Version_2_1	Version 2.1 (Nov. 11, 2004)
Version_1_6	Version 1.6 (June 21, 2004)
Version_1_5	Version 1.5 (Dec. 16, 2002)
Version_1_4	Version 1.4 (June 28, 2001)

Modelica.UsersGuide.ReleaseNotes.Version_2_2_2

Version 2.2.2 (Aug. 12, 2007)

Version 2.2.2 is backward compatible to version 2.2.1 and 2.2 with the following exceptions:

- Removal of package Modelica.Media.Interfaces.PartialTwoPhaseMediumWithCache (this was not yet utilized).
- Removal of the media packages in Modelica.Media.IdealGases.SingleGases that are not type compatible to Modelica.Media.Interfaces.PartialMedium, because a FluidConstants record definition is missing (for details, see Modelica.Media.IdealGases)

An overview of the differences between version 2.2.2 and the previous version 2.2.1 is given below. The exact differences (but without differences in the documentation) are available here. This comparison file was generated automatically with Dymolas ModelManagement.compare function.

In this version, **no** new libraries have been added. The **documentation** of the whole library was improved. Especially, the documentation is now also available as one pdf file.

The following **new components** have been added to **existing** libraries:

Blocks.Logical.	
TerminateSimulation	Terminate a simulation by a given condition.
Blocks.Routing.	
RealPassThrough IntegerPassThrough BooleanPassThrough	Pass a signal from input to output (useful in combination with a bus due to restrictions of expandable connectors).
Blocks.Sources.	
KinematicPTP2	Directly gives q,qd,qdd as output (and not just qdd as KinematicPTP).
Electrical.Machines.Examples.	
TransformerTestbench	Transformer Testbench
Rectifier6pulse	6-pulse rectifier with 1 transformer
Rectifier12pulse	12-pulse rectifier with 2 transformers
AIMC_Steinmetz	Asynchronous induction machine squirrel cage with Steinmetz connection
Electrical.Machines.BasicMachines.Components.	
BasicAIM	Partial model for asynchronous induction machine
BasicSM	Partial model for synchronous induction machine
PartialAirGap	Partial airgap model
BasicDCMachine	Partial model for DC machine
PartialAirGapDC	Partial airgap model of a DC machine
BasicTransformer	Partial model of threephase transformer
PartialCore	Partial model of transformer core with 3 windings
IdealCore	Ideal transformer with 3 windings
Electrical.Machines.BasicMachines.	
Transformers	Sub-Library for technical 3phase transformers
Electrical.Machines.Interfaces.	
Adapter	Adapter to model housing of electrical machine
Math.	
Vectors	New library of functions operating on vectors
atan3	Four quadrant inverse tangens (select solution that is closest to given angle y0)
asinh	Inverse of sinh (area hyperbolic sine)
acosh	Inverse of cosh (area hyperbolic cosine)
Math.Vectors	
isEqual	Determine if two Real vectors are numerically identical
norm	Return the p-norm of a vector
length	Return length of a vector (better as norm(), if further symbolic processing is performed)
normalize	Return normalized vector such that length = 1 and prevent zero-division for zero vector
reverse	Reverse vector elements (e.g. v[1] becomes last element)
sort	Sort elements of vector in ascending or descending order
Math.Matrices	
solve2	Solve real system of linear equations A*X=B with a B matrix (Gaussian elemination with partial pivoting)
LU_solve2	Solve real system of linear equations P*L*U*X=B with a B matrix and an LU decomposition (from LU(..))

Mechanics.Rotational.	
InitializeFlange	Initialize a flange according to given signals (useful if initialization signals are provided by a signal bus).
Media.Interfaces.PartialMedium.	
density_pTX	Return density from p, T, and X or Xi
Media.Interfaces.PartialTwoPhaseMedium.	
BaseProperties	Base properties (p, d, T, h, u, R, MM, x) of a two phase medium
molarMass	Return the molar mass of the medium
saturationPressure_sat	Return saturation pressure
saturationTemperature_sat	Return saturation temperature
saturationTemperature_derp_sat	Return derivative of saturation temperature w.r.t. pressure
setState_px	Return thermodynamic state from pressure and vapour quality
setState_Tx	Return thermodynamic state from temperature and vapour quality
vapourQuality	Return vapour quality
Media.Interfaces.	
PartialLinearFluid	Generic pure liquid model with constant cp, compressibility and thermal expansion coefficients
Media.Air.MoistAir.	
massFraction_pTphi	Return the steam mass fraction from relative humidity and T
saturationTemperature	Return saturation temperature from (partial) pressure via numerical inversion of function saturationPressure
enthalpyOfWater	Return specific enthalpy of water (solid/liquid) near atmospheric pressure from temperature
enthalpyOfWater_der	Return derivative of enthalpyOfWater()" function
PsychrometricData	Model to generate plot data for psychrometric chart
Media.CompressibleLiquids. New sub-library for simple compressible liquid models	
LinearColdWater	Cold water model with linear compressibility
LinerWater_pT_Ambient	Liquid, linear compressibility water model at 1.01325 bar and 25 degree Celsius
SIunits.	
TemperatureDifference	Type for temperature difference

The following **existing components** have been **improved**:

Blocks.Examples.	
BusUsage	Example changed from the "old" to the "new" bus concept with expandable connectors.
Blocks.Discrete.	
ZeroOrderHold	Sample output ySample moved from "protected" to "public" section with new attributes (start=0, fixed=true).
TransferFunction	Discrete state x with new attributes (each start=0, each fixed=0).
Electrical.	
Analog MultiPhase	Improved some icons.
Electrical.Digital.Interfaces.	
MISO	Removed "algorithm" from this partial block.
Electrical.Digital.Delay.	
DelayParams	Removed "algorithm" from this partial block.

Electrical.Digital.Delay.	
DelayParams	Removed "algorithm" from this partial block.
TransportDelay	If delay time is zero, an infinitely small delay is introduced via pre(x) (previously "x" was used).
Electrical.Digital.Sources.	
Clock Step	Changed if-conditions from "xxx < time" to "time >= xxx" (according to the Modelica specification, in the second case a time event should be triggered, i.e., this change leads potentially to a faster simulation).
Electrical.Digital.Converters.	
BooleanToLogic LogicToBoolean RealToLogic LogicToReal	Changed from "algorithm" to "equation" section to allow better symbolic preprocessing
Electrical.	
Machines	Slightly improved documentation, typos in documentation corrected
Electrical.Machines.Examples.	
AIMS_start	Changed QuadraticLoadTorque1(TorqueDirection=true) to QuadraticLoadTorque1(TorqueDirection=false) since more realistic
Electrical.Machines.Interfaces.	
PartialBasicMachine	Introduced support flange to model the reaction torque to the housing
Electrical.Machines.Sensors.	
Rotorangle	Introduced support flange to model the reaction torque to the housing
Mechanics.MultiBody.Examples.Elementary.	
PointMassesWithGravity	Added two point masses connected by a line force to demonstrate additionally how this works. Connections of point masses with 3D-elements are demonstrated in the new example PointMassesWithGravity (there is the difficulty that the orientation is not defined in a PointMass object and therefore some special handling is needed in case of a connection with 3D-elements, where the orientation of the point mass is not determined by these elements.
Mechanics.MultiBody.Examples.Systems.	
RobotR3	Changed from the "old" to the "new" bus concept with expandable connectors. Replaced the non-standard Modelica function "constrain()" by standard Modelica components. As a result, the non-standard function constrain() is no longer used in the Modelica Standard Library.
Mechanics.MultiBody.Frames.Orientation.	
equalityConstraint	Use a better residual for the equalityConstraint function. As a result, the non-linear equation system of a kinematic loop is formulated in a better way (the range where the desired result is a unique solution of the non-linear system of equations becomes much larger).
Mechanics.MultiBody.	
Visualizers.	Removed (misleading) annotation "structurallyIncomplete" in the models of this sub-library
Mechanics.Rotational.	
Examples	For all models in this sub-library: • Included a housing object in all examples to compute all support torques. • Replaced initialization by modifiers via the initialization menu parameters of Inertia components. • Removed "encapsulated" and unnecessary "import".

	• Included "StopTime" in the annotations.
Mechanics.Rotational.Interfaces.	
FrictionBase	Introduced "fixed=true" for Boolean variables startForward, startBackward, mode.
Mechanics.Translational.Interfaces.	
FrictionBase	Introduced "fixed=true" for Boolean variables startForward, startBackward, mode.
Media.UsersGuide.MediumUsage.	
TwoPhase	Improved documentation and demonstrating the newly introduced functions
Media.Examples.	
WaterIF97	Provided (missing) units for variables V, dV, H_flow_ext, m, U.
Media.Interfaces.	
PartialMedium	Final modifiers are removed from nX and nXi, to allow customized medium models such as mixtures of refrigerants with oil, etc.
PartialCondensingGases	Included attributes "min=1, max=2" for input argument FixedPhase for functions setDewState and setBubbleState (in order to guarantee that input arguments are correct).
Media.Interfaces.PartialMedium.	
BaseProperties	New Boolean parameter "standardOrderComponents". If true, last element vector X is computed from 1-sum(Xi) (= default) otherwise, no equation is provided for it in PartialMedium.
IsentropicExponent	"max" value changed from 1.7 to 500000
setState_pTX setState_phX setState_psX setState_dTX specificEnthalpy_pTX temperature_phX density_phX temperature_psX density_psX specificEnthalpy_psX	Introduced defaut value "reference_X" for input argument "X".
Media.Interfaces.PartialSimpleMedium.	
setState_pTX setState_phX setState_psX setState_dTX	Introduced defaut value "reference_X" for input argument "X".
Media.Interfaces.PartialSimpleIdealGasMedium.	
setState_pTX setState_phX setState_psX setState_dTX	Introduced defaut value "reference_X" for input argument "X".
Media.Air.MoistAir.	
setState_pTX setState_phX setState_dTX	Introduced defaut value "reference_X" for input argument "X".
Media.IdealGases.Common.SingleGasNasa.	
setState_pTX setState_phX setState_psX	Introduced defaut value "reference_X" for input argument "X".

setState_dTX	
Media.IdealGases.Common.MixtureGasNasa.	
setState_pTX setState_phX setState_psX setState_dTX h_TX	Introduced defaut value "reference_X" for input argument "X".
Media.Common.	
IF97PhaseBoundaryProperties gibbsToBridgmansTables	Introduced unit for variables vt, vp.
SaturationProperties	Introduced unit for variable dpT.
BridgmansTables	Introduced unit for dfs, dgs.
Media.Common.ThermoFluidSpecial.	
gibbsToProps_ph gibbsToProps_ph gibbsToBoundaryProps gibbsToProps_dT gibbsToProps_pT	Introduced unit for variables vt, vp.
TwoPhaseToProps_ph	Introduced unit for variables dht, dhd, detph.
Media.	
MoistAir	Documentation of moist air model significantly improved.
Media.MoistAir.	
enthalpyOfVaporization	Replaced by linear correlation since simpler and more accurate in the entire region.
Media.Water.IF97_Utilities.BaseIF97.Regions.	
drhovl_dp	Introduced unit for variable dd_dp.
Thermal.	
FluidHeatFlow	Introduced new parameter tapT (0..1) to define the temperature of the HeatPort as linear combination of the flowPort_a (tapT=0) and flowPort_b (tapT=1) temperatures.

The following **critical errors** have been fixed (i.e. errors that can lead to wrong simulation results):

Electrical.Machines.BasicMachines.Components.	
ElectricalExcitation	Excitation voltage ve is calculated as "spacePhasor_r.v_[1]*TurnsRatio*3/2" instead of "spacePhasor_r.v_[1]*TurnsRatio
Mechanics.MultiBody.Parts.	
FixedRotation	Bug corrected that the torque balance was wrong in the following cases (since vector r was not transformed from frame_a to frame_b): • frame_b is in the spanning tree closer to the root (usually this is frame_a). • vector r from frame_a to frame_b is not zero.
PointMass	If a PointMass model is connected so that no equations are present to compute its orientation object, the orientation was arbitrarily set to a unit rotation. In some cases this can lead to a wrong overall model, depending on how the PointMass model is used. For this reason, such cases lead now to an error (via an assert(..)) with an explanation how to fix this.
Media.Interfaces.PartialPureSubstance.	
pressure_dT	Changed wrong call from "setState_pTX" to "setState_dTX"

specificEnthalpy_dT	
Media.Interfaces.PartialTwoPhaseMedium.	
pressure_dT specificEnthalpy_dT	Changed wrong call from "setState_pTX" to "setState_dTX"
Media.Common.ThermoFluidSpecial.	
gibbsToProps_dT helmholtzToProps_ph helmholtzToProps_pT helmholtzToProps_dT	Bugs in equations corrected
Media.Common.	
helmholtzToBridgmansTables helmholtzToExtraDerivs	Bugs in equations corrected
Media.Water.IF97_Utilities.	
BaseIF97.Inverses.dtofps3 isentropicExponent_props_ph isentropicExponent_props_pT isentropicExponent_props_dT	Bugs in equations corrected
Media.Air.MoistAir.	
h_pTX	Bug in setState_phX due to wrong vector size in h_pTX corrected. Furthermore, syntactical errors corrected: • In function massFractionpTphi an equation sign is used in an algorithm. • Two consecutive semicolons removed
Media.Water.	
waterConstants	Bug in equation of criticalMolarVolume corrected.
Media.Water.IF97_Utilities.BaseIF97.Regions.	
region_ph region_ps	Bug in region determination corrected.
boilingcurve_p dewcurve_p	ug in equation of plim corrected.

The following **uncritical errors** have been fixed (i.e. errors that do **not** lead to wrong simulation results, but, e.g., units are wrong or errors in documentation):

Blocks.	
Examples	Corrected typos in description texts of bus example models.
Blocks.Continuous.	
LimIntegrator	removed incorrect smooth(0,..) because expression might be discontinuous.
Blocks.Math.UnitConversions.	
block_To_kWh block_From_kWh	Corrected unit from "kWh" to (syntactically correct) "kW.h".
Electrical.Analog.Examples.	
HeatingNPN_OrGate	Included start values, so that initialization is successful
Electrical.Analog.Lines.	
OLine	Corrected unit from "Siemens/m" to "S/m".
TLine2	Changed wrong type of parameter NL (normalized length) from

	SIunits.Length to Real.
Electrical.Digital.Delay.	
TransportDelay	Syntax error corrected (":=" in equation section is converted by Dymola silently to "=").
Electrical.Digital	
Converters	Syntax error corrected (":=" in equation section is converted by Dymola silently to "=").
Electrical.MultiPhase.Basic.	
Conductor	Changed wrong type of parameter G from SIunits.Resistance to SIunits.Conductance.
Electrical.MultiPhase.Interfaces.	
Plug	Made used "pin" connectors non-graphical (otherwise, there are difficulties to connect to Plug).
Electrical.MultiPhase.Sources.	
SineCurrent	Changed wrong type of parameter offset from SIunits.Voltage to SIunits.Current.
Mechanics.MultiBody.Examples.Loops.	
EngineV6	Corrected wrong crankAngleOffset of some cylinders and improved the example.
Mechanics.MultiBody.Examples.Loops.Utilities.	
GasForce	Wrong units corrected: "SIunitsPosition x,y" to "Real x,y"; "SIunits.Pressure press" to "SIunits.Conversions.NonSIunits.Pressure_bar"
GasForce2	Wrong unit corrected: "SIunits.Position x" to "Real x".
EngineV6_analytic	Corrected wrong crankAngleOffset of some cylinders.
Mechanics.MultiBody.Interfaces.	
PartialLineForce	Corrected wrong unit: "SIunits.Position eRod_a" to "Real eRod_a";
FlangeWithBearingAdaptor	If includeBearingConnector = false, connector "frame" was not removed. As long as the connecting element to "frame" determines the non-flow variables, this is fine. In the corrected version, "frame" is conditionally removed in this case.
Mechanics.MultiBody.Forces.	
ForceAndTorque	Corrected wrong unit: "SIunits.Force t_b_0" to "SIunits.Torque t_b_0".
LineForceWithTwoMasses	Corrected wrong unit: "SIunits.Position e_rel_0" to "Real e_rel_0".
Mechanics.MultiBody.Frames.	
axisRotation	Corrected wrong unit: "SIunits.Angle der_angle" to "SIunits.AngularVelocity der_angle".
Mechanics.MultiBody.Joints.Assemblies.	
JointUSP JointSSP	Corrected wrong unit: "SIunits.Position aux" to "Real"
Mechanics.MultiBody.Sensors.	
AbsoluteSensor	Corrected wrong units: "SIunits.Acceleration angles" to "SIunits.Angle angles" and "SIunits.Velocity w_abs_0" to "SIunits.AngularVelocity w_abs_0"
RelativeSensor	Corrected wrong units: "SIunits.Acceleration angles" to "SIunits.Angle angles"
Distance	Corrected wrong units: "SIunits.Length L2" to "SIunits.Area L2" and SIunits.Length s_small2" to "SIunits.Area s_small2"
Mechanics.MultiBody.Visualizers.Advanced.	
Shape	Changed "MultiBody.Types.Color color" to "Real color[3]", since "Types.Color" is "Integer color[3]" and there have been backward compatibility problems with models using "color" before it was changed to

	"Types.Color".
Mechanics.Rotational.Interfaces.	
FrictionBase	Rewrote equations with new variables "unitAngularAcceleration" and "unitTorque" in order that the equations are correct with respect to units (previously, variable "s" can be both a torque and an angular acceleration and this lead to unit incompatibilities)
Mechanics.Rotational.	
OneWayClutch LossyGear	Rewrote equations with new variables "unitAngularAcceleration" and "unitTorque" in order that the equations are correct with respect to units (previously, variable "s" can be both a torque and an angular acceleration and this lead to unit incompatibilities)
Mechanics.Translational.Interfaces.	
FrictionBase	Rewrote equations with new variables "unitAngularAcceleration" and "unitTorque" in order that the equations are correct with respect to units (previously, variable "s" can be both a torque and an angular acceleration and this lead to unit incompatibilities)
Mechanics.Translational.	
Speed	Corrected unit of v_ref from SIunits.Position to SIunits.Velocity
Media.Examples.Tests.Components.	
PartialTestModel PartialTestModel2	Corrected unit of h_start from "SIunits.Density" to "SIunits.SpecificEnthalpy"
Media.Examples.SolveOneNonlinearEquation.	
Inverse_sh_T InverseIncompressible_sh_T Inverse_sh_TX	Rewrote equations so that dimensional (unit) analysis is correct"
Media.Incompressible.Examples.	
TestGlycol	Rewrote equations so that dimensional (unit) analysis is correct"
Media.Interfaces.PartialTwoPhaseMedium.	
dBubbleDensity_dPressure dDewDensity_dPressure	Changed wrong type of ddldp from "DerDensityByEnthalpy" to "DerDensityByPressure".
Media.Common.ThermoFluidSpecial.	
ThermoProperties	Changed wrong units: "SIunits.DerEnergyByPressure dupT" to "Real dupT" and "SIunits.DerEnergyByDensity dudT" to "Real dudT"
ThermoProperties_ph	Changed wrong unit from "SIunits.DerEnergyByPressure duph" to "Real duph"
ThermoProperties_pT	Changed wrong unit from "SIunits.DerEnergyByPressure dupT" to "Real dupT"
ThermoProperties_dT	Changed wrong unit from "SIunits.DerEnergyByDensity dudT" to "Real dudT"
Media.IdealGases.Common.SingleGasNasa.	
cp_Tlow_der	Changed wrong unit from "SIunits.Temperature dT" to "Real dT".
Media.Water.IF97_Utilities.BaseIF97.Basic.	
p1_hs h2ab_s p2a_hs p2b_hs p2c_hs h3ab_p T3a_ph T3b_ph v3a_ph v3b_ph	Changed wrong unit of variables h/hstar, s, sstar from "SIunits.Enthalpy" to "SIunits.SpecificEnthalpy", "SIunits.SpecificEntropy", "SIunits.SpecificEntropy

T3a_ps T3b_ps v3a_ps v3b_ps	
Media.Water.IF97_Utilities.BaseIF97.Transport.	
cond_dTp	Changed wrong unit of TREL, rhoREL, lambdaREL from "SIunits.Temperature", "SIunit.Density", "SIunits.ThermalConductivity" to "Real".
Media.Water.IF97_Utilities.BaseIF97.Inverses.	
tofps5 tofpst5	Changed wrong unit of pros from "SIunits.SpecificEnthalpy" to "SIunits.SpecificEntropy".
Media.Water.IF97_Utilities.	
waterBaseProp_ph	Improved calculation at the limits of the validity.
Thermal.	
FluidHeatFlow HeatTransfer	Corrected wrong unit "SIunits.Temperature" of difference temperature variables with "SIunits.TemperatureDifference".

Modelica.UsersGuide.ReleaseNotes.Version_2_2_1

Version 2.2.1 (March 24, 2006)

Version 2.2.1 is backward compatible to version 2.2.

In this version, **no** new libraries have been added. The following major improvements have been made:

- The **Documentation** of the Modelica standard library was considerably improved:
  In Dymola 6, the new feature was introduced to automatically add tables for class content and component interface definitions (parameters and connectors) to the info layer. For this reason, the corresponding (partial) tables previously present in the Modelica Standard Library have been removed. The new feature of Dymola 6 has the significant advantage that all tables are now guaranteed to be up-to-date.
  Additionally, the documentation has been improved by adding appropriate description texts to parameters, connector instances, function input and output arguments etc., in order that the automatically generated tables do not have empty entries. Also new User's Guides for sublibraries Rotational and SIunits have been added and the User's Guide on top level (Modelica.UsersGuide) has been improved.

- Initialization options have been added to the Modelica.Blocks.**Continuous** blocks (NoInit, SteadyState, InitialState, InitialOutput). If InitialOutput is selected, the block output is provided as initial condition. The states of the block are then initialized as close as possible to steady state. Furthermore, the Continuous.LimPID block has been significantly improved and much better documented.

- The Modelica.**Media** library has been significantly improved:
  New functions setState_pTX, setState_phX, setState_psX, setState_dTX have been added to PartialMedium to compute the independent medium variables (= state of medium) from p,T,X, or from p,h,X or from p,s,X or from d,T,X. Then functions are provided for all interesting medium variables to compute them from its medium state. All these functions are implemented in a robust way for all media (with a few exceptions, if the generic function does not make sense for a particular medium).

The following **new components** have been added to **existing** libraries:

Modelica.Blocks.Examples.	
PID_Controller	Example to demonstrate the usage of the Blocks.Continuous.LimPID block.

Modelica.Blocks.Math.	
UnitConversions.*	New package that provides blocks for unit conversions. UnitConversions.ConvertAllBlocks allows to select all available conversions from a menu.
Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines.	
SM_ElectricalExcitedDamperCage	Electrical excited synchronous induction machine with damper cage
Modelica.Electrical.Machines.BasicMachines.Components.	
ElectricalExcitation	Electrical excitation for electrical excited synchronous induction machines
DamperCage	Unsymmetrical damper cage for electrical excited synchronous induction machines. At least the user has to specify the dampers resistance and stray inductance in d-axis; if he omits the parameters of the q-axis, the same values as for the d.axis are used, assuming a symmetrical damper.
Modelica.Electrical.Machines.Examples.	
SMEE_Gen	Test example 7: ElectricalExcitedSynchronousInductionMachine as Generator
Utilities.TerminalBox	Terminal box for three-phase induction machines to choose either star (wye) ? or delta ? connection
Modelica.Math.Matrices.	
equalityLeastSquares	Solve a linear equality constrained least squares problem: min|A*x-a|^2 subject to B*x=b
Modelica.Mechanics.MultiBody.	
Parts.PointMass	Point mass, i.e., body where inertia tensor is neglected.
Interfaces.FlangeWithBearing	Connector consisting of 1-dim. rotational flange and its 3-dim. bearing frame.
Interfaces.FlangeWithBearingAdaptor	Adaptor to allow direct connections to the sub-connectors of FlangeWithBearing.
Types.SpecularCoefficient	New type to define a specular coefficient.
Types.ShapeExtra	New type to define the extra data for visual shape objects and to have a central place for the documentation of this data.
Modelica.Mechanics.MultiBody.Examples.Elementary	
PointGravityWithPointMasses	Example of two point masses in a central gravity field.
Modelica.Mechanics.Rotational.	
UsersGuide	A User's Guide has been added by using the documentation previously present in the package documentation of Rotational.
Sensors.PowerSensor	New component to measure the energy flow between two connectors of the Rotational library.
Modelica.Mechanics.Translational.	
Speed	New component to move a translational flange according to a reference speed
Modelica.Media.Interfaces.PartialMedium.	
specificEnthalpy_pTX	New function to compute specific enthalpy from pressure, temperature and mass fractions.
temperature_phX	New function to compute temperature from pressure, specific enthalpy, and mass fractions.
Modelica.Icons.	
SignalBus	Icon for signal bus
SignalSubBus	Icon for signal sub-bus

Modelica.SIunits.	
UsersGuide	A User's Guide has been added that describes unit handling.
Resistance Conductance	Attribute 'min=0' removed from these types.
Modelica.Thermal.FluidHeatFlow.	
Components.Valve	Simple controlled valve with either linear or exponential characteristic.
Sources. IdealPump	Simple ideal pump (resp. fan) dependent on the shaft?s speed; pressure increase versus volume flow is defined as a linear function. Torque * Speed = Pressure invrease * Volume flow (without losses).
Examples.PumpAndValve	Test example for valves.
Examples.PumpDropOut	Drop out of 1 pump to test behavior of semiLinear.
Examples.ParallelPumpDropOut	Drop out of 2 parallel pumps to test behavior of semiLinear.
Examples.OneMass	Cooling of 1 hot mass to test behavior of semiLinear.
Examples.TwoMass	Cooling of 2 hot masses to test behavior of semiLinear.

The following **components** have been improved:

Modelica.	
UsersGuide	User's Guide and package description of Modelica Standard Library improved.
Modelica.Blocks.Interfaces.	
RealInput BooleanInput IntegerInput	When dragging one of these connectors the width and height is a factor of 2 larger as a standard icon. Previously, these connectors have been dragged and then manually enlarged by a factor of 2 in the Modelica standard library.
Modelica.Blocks.	
Continuous.*	Initialization options added to all blocks (NoInit, SteadyState, InitialState, InitialOutput). New parameter limitsAtInit to switch off the limits of LimIntegrator or LimPID during initialization
Continuous.LimPID	Option to select P, PI, PD, PID controller. Documentation significantly improved.
Nonlinear.Limiter Nonlinear.VariableLimiter Nonlinear.Deadzone	New parameter limitsAtInit/deadZoneAtInit to switch off the limits or the dead zone during initialization
Modelica.Electrical.Analog.	
Sources	Icon improved (+/- added to voltage sources, arrow added to current sources).
Modelica.Electrical.Analog.Semiconductors.	
Diode	smooth() operator included to improve numerics.
Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines.	
SM_PermanentMagnetDamperCage SM_ElectricalExcitedDamperCage SM_ReluctanceRotorDamperCage	The user can choose "DamperCage = false" (default: true) to remove all equations for the damper cage from the model.
Modelica.Electrical.Machines.BasicMachines.AsynchronousInductionMachines.	
AIM_SlipRing	Easier parameterization: if the user selects "useTrunsRatio = false" (default: true, this is the same behavior as before), parameter TurnsRatio is calculated internally from Nominal stator voltage and Locked-rotor voltage.
Modelica.Math.Matrices.	
leastSquares	The A matrix in the least squares problem might be rank deficient. Previously, it was required that A has full rank.

Modelica.Mechanics.MultiBody.	
all models	• All components with animation information have a new variable **specularCoefficient** to define the reflection of ambient light. The default value is world.defaultSpecularCoefficient which has a default of 0.7. By changing world.defaultSpecularCoefficient, the specularCoefficient of all components is changed that are not explicitly set differently. Since specularCoefficient is a variable (and no parameter), it can be changed during simulation. Since annotation(Dialog) is set, this variable still appears in the parameter menus. Previously, a constant specularCoefficient of 0.7 was used for all components. • Variable **color** of all components is no longer a parameter but an input variable. Also all parameters in package **Visualizers**, with the exception of **shapeType** are no longer parameters but defined as input variables with annotation(Dialog). As a result, all these variables appear still in parameter menus, but they can be changed during simulation (e.g., color might be used to display the temperature of a part). • All menus have been changed to follow the Modelica 2.2 annotations "Dialog, group, tab, enable" (previously, a non-standard Dymola definition for menus was used). Also, the "enable" annotation is used in all menus to disable input fields if the input would be ignored. • All visual shapes are now defined with conditional declarations (to remove them, if animation is switched off). Previously, these (protected) objects have been defined by arrays with dimension 0 or 1.
Frames.resolveRelative	The derivative of this function added as function and defined via an annotation. In certain situations, tools had previously difficulties to differentiate the inlined function automatically.
Forces.*	The scaling factors N_to_m and Nm_to_m have no longer a default value of 1000 but a default value of world.defaultN_to_m (=1000) and world.defaultNm_to_m (=1000). This allows to change the scaling factors for all forces and torques in the world object.
Interfaces.Frame.a Interfaces.Frame.b Interfaces.Frame_resolve	The Frame connectors are now centered around the origin to ease the usage. The shape was changed, such that the icon is a factor of 1.6 larger as a standard icon (previously, the icon had a standard size when dragged and then the icon was manually enlarged by a factor of 1.5 in the y-direction in the MultiBody library; the height of 16 allows easy positioning on the standard grid size of 2). The double line width of the border in icon and diagram layer was changed to a single line width and when making a connection the connection line is dark grey and no longer black which looks better.
Joints.Assemblies.*	When dragging an assembly joint, the icon is a factor of 2 larger as a standard icon. Icon texts and connectors have a standard size in this enlarged icon (and are not a factor of 2 larger as previously).
Types.*	All types have a corresponding icon now to visualize the content in the package browser (previously, the types did not have an icon).
Modelica.Mechanics.Rotational.	
Inertia	Initialization and state selection added.
SpringDamper	Initialization and state selection added.
Move	New implementation based solely on Modelica 2.2 language

	(previously, the Dymola specific constrain(..) function was used).
Modelica.Mechanics.Translational.	
Move	New implementation based solely on Modelica 2.2 language (previously, the Dymola specific constrain(..) function was used).
Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.	
SimpleFriction	Calculates friction losses from pressure drop and volume flow.
Modelica.Thermal.FluidHeatFlow.Components.	
IsolatedPipe HeatedPipe	Added geodetic height as a source of pressure change; feeds friction losses as calculated by simple friction to the energy balance of the medium.
Modelica.Media.Interfaces.PartialMedium.FluidConstants.	
HCRIT0	Critical specifc enthalpy of the fundamental equation (base formulation of the fluid medium model).
SCRIT0	Critical specifc entropy of the fundamental equation (base formulation of the fluid medium model).
deltah	Enthalpy offset (default: 0) between the specific enthalpy of the fluid model and the user-visible specific enthalpy in the model: deltah = h_model - h_fundamentalEquation.
deltas	Entropy offset (default: 0) between the specific entropy of the fluid model and the user-visible specific entropy in the model: deltas = s_model - s_fundamentalEquation.
T_default	Default value for temperature of medium (for initialization)
h_default	Default value for specific enthalpy of medium (for initialization)
p_default	Default value for pressure of medium (for initialization)
X_default	Default value for mass fractions of medium (for initialization)

The following **errors** have been fixed:

Modelica.Blocks.Tables.	
CombiTable1D CombiTable1Ds CombiTable2D	Parameter "tableOnFile" determines now whether a table is read from file or used from parameter "table". Previously, if "fileName" was not "NoName", a table was always read from file "fileName", independently of the setting of "tableOnFile". This has been corrected. Furthermore, the initialization of a table is now performed in a when-clause and no longer in a parameter declaration, because some tools evaluate the parameter declaration in some situation more than once and then the table is unnecessarily read several times (and occupies also more memory).
Modelica.Blocks.Sources.	
CombiTimeTable	Same bug fix/improvement as for the tables from Modelica.Blocks.Tables as outlined above.
Modelica.Electrical.Analog.Semiconductors.	
PMOS NMOS HeatingPMOS HeatingNMOS	The Drain-Source-Resistance RDS had actually a resistance of RDS/v, with v=Beta*(W+dW)/(L+dL). The correct formula is without the divison by "v". This has now been corrected. This bug fix should not have an essential effect in most applications. In the default case (Beta=1e-5), the Drain-Source-Resistance was a factor of 1e5 too large and had in the default case the wrong value 1e12, although it should have the value 1e7. The effect was that this resistance had practically no effect.
Modelica.Media.IdealGases.Common.SingleGasNasa.	
dynamicViscosityLowPressure	Viscosity and thermal conductivity (which needs viscosity as input) were computed wrong for polar gases and gas mixtures (i.e. if dipole moment not 0.0). This has been fixed in version 2.2.1.

Modelica.Utilities.Streams.	
readLine	Depending on the C-implementation, the stream was not correctly closed. This has been corrected by adding a "Streams.close(..)" after reading the file content.

Modelica.UsersGuide.ReleaseNotes.Version_2_2

Version 2.2 (April 6, 2005)

Version 2.2 is backward compatible to version 2.1.

The following **new libraries** have been added:

Modelica.Media	Property models of liquids and gases, especially • 1241 detailed gas models, • moist air, • high precision water model (according to IAPWS/IF97 standard), • incompressible media defined by tables (cp(T), rho(t), eta(T), etc. are defined by tables). The user can conveniently define mixtures of gases between the 1241 gas models. The models are designed to work well in dynamic simulations. They are based on a new standard interface for media with single and multiple substances and one or multiple phases with the following features: • The independent variables of a medium model do not influence the definition of a fluid connector port or how the balance equations have to be implemented. Used independent variables: "p,T", "p,T,X", "p,h", "d,T". • Optional variables, e.g., dynamic viscosity, are only computed if needed. • The medium models are implemented with regards to efficient dynamic simulation.
Modelica.Thermal.FluidHeatFlow	Simple components for 1-dim., incompressible thermo-fluid flow to model coolant flows, e.g., of electrical machines. Components can be connected arbitrarily together (= ideal mixing at connection points) and fluid may reverse direction of flow.

The following **changes** have been performed in the **Modelica.Mechanics.MultiBody** library:

- Component MultiBody.World has a new parameter **driveTrainMechanics3D**. If set to **true**, 3D mechanical effects of MultiBody.Parts.Mounting1D/Rotor1D/BevelGear1D are taken into account. If set to **false** (= default), 3D mechanical effects in these elements are not taken into account and the frame connectors to connect to 3D parts are disabled (all connections to such a disabled connector are also disabled, due to the new feature of conditional declarations in Modelica language 2.2)
- All references to "MultiBody.xxx" have been changed to "Modelica.Mechanics.MultiBodys.xxx" in order that after copying of a component outside of the Modelica library, the references still remain valid.

Modelica.UsersGuide.ReleaseNotes.Version_2_1

Version 2.1 (Nov. 11, 2004)

This is a major change with respect to previous versions of the Modelica Standard Library, because **many new libraries** and components are included and because the input/output blocks (Modelica.Blocks) have been considerably simplified:

- An input/output connector is defined **without** a hierarchy (this is possible due to new features of the Modelica language). For example, the input signal of a block "FirstOrder" was previously accessed as "FirstOrder.inPort.signal[1]". Now it is accessed as "FirstOrder.u". This simplifies the understanding and usage especially for beginners.
- De-vectorized the **Modelica.Blocks** library. All blocks in the Modelica.Blocks library are now scalar blocks. As a result, the parameters of the Blocks are scalars and no vectors any more. For example, a parameter "amplitude" that might had a value of "{1}" previously, has now a value of "1". This simplifies the understanding and usage especially for beginners.
  If a vector of blocks is needed, this can be easily accomplished by adding a dimension to the instance. For example "Constant const[3](k={1,2,3}" defines three Constant blocks. An additional advantage of the new approach is that the implementation of Modelica.Blocks is much simpler and is easier to understand.

The discussed changes of Modelica.Blocks are not backward compatible. A script to **automatically** convert models to this new version is provided. There might be rare cases, where this script does not convert. In this case models have to be manually converted. In any case you should make a back-up copy of your model before automatic conversion is performed.

The following **new libraries** have been added:

Modelica.Electrical.Digital	Digital electrical components based on 2-,3-,4-, and 9-valued logic according to the VHDL standard
Modelica.Electrical.Machines	Asynchronous, synchronous and DC motor and generator models
Modelica.Math.Matrices	Functions operatinng on matrices such as solve() (A*x=b), leastSquares(), norm(), LU(), QR(), eigenValues(), singularValues(), exp(), ...
Modelica.StateGraph	Modeling of **discrete event** and **reactive** systems in a convenient way using **hierarchical state machines** and **Modelica** as **action language**. It is based on JGraphChart and Grafcet and has a similar modeling power as StateCharts. It avoids deficiences of usually used action languages. This library makes the ModelicaAdditions.PetriNets library obsolet.
Modelica.Utilities.Files	Functions to operate on files and directories (copy, move, remove files etc.)
Modelica.Utilities.Streams	Read from files and write to files (print, readLine, readFile, error, ...)
Modelica.Utilities.Strings	Operations on strings (substring, find, replace, sort, scanToken, ...)
Modelica.Utilities.System	Get/set current directory, get/set environment variable, execute shell command, etc.

The following existing libraries outside of the Modelica standard library have been improved and added as **new libraries** (models using the previous libraries are automatically converted to the new sublibraries inside package Modelica):

Modelica.Blocks.Discrete	Discrete input/output blocks with fixed sample period (from ModelicaAdditions.Blocks.Discrete)
Modelica.Blocks.Logical	Logical components with Boolean input and output signals (from ModelicaAdditions.Blocks.Logical)
Modelica.Blocks.Nonlinear	Discontinuous or non-differentiable algebraic control blocks such as variable limiter, fixed, variable and Pade delay etc. (from ModelicaAdditions.Blocks.Nonlinear)
Modelica.Blocks.Routing	Blocks to combine and extract signals, such as multiplexer (from ModelicaAdditions.Blocks.Multiplexer)
Modelica.Blocks.Tables	One and two-dimensional interpolation in tables. CombiTimeTable is available in Modelica.Blocks.Sources (from ModelicaAdditions.Tables)

Modelica.Mechanics.MultiBody	Components to model the movement of 3-dimensional mechanical systems. Contains body, joint, force and sensor components, analytic handling of kinematic loops, force elements with mass, series/parallel connection of 3D force elements etc. (from MultiBody 1.0 where the new signal connectors are used; makes the ModelicaAdditions.MultiBody library obsolet)

As a result, the ModelicaAdditions library is obsolet, because all components are either included in the Modelica library or are replaced by much more powerful libraries (MultiBody, StateGraph).

The following **new components** have been added to **existing** libraries.

Modelica.Blocks.Logical.	
Pre	y = pre(u): Breaks algebraic loops by an infinitesimal small time delay (event iteration continues until u = pre(u))
Edge	y = edge(u): Output y is true, if the input u has a rising edge
FallingEdge	y = edge(not u): Output y is true, if the input u has a falling edge
Change	y = change(u): Output y is true, if the input u has a rising or falling edge
GreaterEqual	Output y is true, if input u1 is greater or equal as input u2
Less	Output y is true, if input u1 is less as input u2
LessEqual	Output y is true, if input u1 is less or equal as input u2
Timer	Timer measuring the time from the time instant where the Boolean input became true
Modelica.Blocks.Math.	
BooleanToReal	Convert Boolean to Real signal
BooleanToInteger	Convert Boolean to Integer signal
RealToBoolean	Convert Real to Boolean signal
IntegerToBoolean	Convert Integer to Boolean signal
Modelica.Blocks.Sources.	
RealExpression	Set output signal to a time varying Real expression
IntegerExpression	Set output signal to a time varying Integer expression
BooleanExpression	Set output signal to a time varying Boolean expression
BooleanTable	Generate a Boolean output signal based on a vector of time instants
Modelica.Mechanics.MultiBody.	
Frames.from_T2	Return orientation object R from transformation matrix T and its derivative der(T)
Modelica.Mechanics.Rotational.	
LinearSpeedDependentTorque	Linear dependency of torque versus speed (acts as load torque)
QuadraticSpeedDependentTorque	Quadratic dependency of torque versus speed (acts as load torque)
ConstantTorque	Constant torque, not dependent on speed (acts as load torque)
ConstantSpeed	Constant speed, not dependent on torque (acts as load torque)
TorqueStep	Constant torque, not dependent on speed (acts as load torque)

The following **bugs** have been **corrected**:

Modelica.Mechanics.MultiBody.Forces.	
LineForceWithMass Spring	If mass of the line force or spring component is not zero, the mass was (implicitly) treated as "mass*mass" instead of as "mass"
Modelica.Mechanics.Rotational.	
Speed	If parameter exact=**false**, the filter was wrong (position was filtered and not the speed input signal)

Other changes:

- All connectors are now smaller in the diagram layer. This gives a nicer layout when connectors and components are used together in a diagram
- Default instance names are defined for all connectors, according to a new annotation introduced in Modelica 2.1. For example, when dragging connector "Flange_a" from the Rotational library to the diagram layer, the default connector instance name is "flange_a" and not "Flange_a1".
- The Modelica.Mechanics.Rotational connectors are changed from a square to a circle
- The Modelica.Mechanics.Translational connectors are changed from a green to a dark green color in order that connection lines can be better seen, especially when printed.
- The Modelica.Blocks connectors for Real signals are changed from blue to dark blue in order to distinguish them from electrical signals.

Modelica.UsersGuide.ReleaseNotes.Version_1_6

Version 1.6 (June 21, 2004)

Added 1 new library (Electrical.MultiPhase), 17 new components, improved 3 existing components in the Modelica.Electrical library and improved 3 types in the Modelica.SIunits library. Furthermore, this User's Guide has been started. The improvements in more detail:

New components

Modelica.Electrical.Analog.Basic.	
SaturatingInductor	Simple model of an inductor with saturation
VariableResistor	Ideal linear electrical resistor with variable resistance
VariableConductor	Ideal linear electrical conductor with variable conductance
VariableCapacitor	Ideal linear electrical capacitor with variable capacitance
VariableInductor	Ideal linear electrical inductor with variable inductance
Modelica.Electrical.Analog.Semiconductors.	
HeadingDiode	Simple diode with heating port
HeadingNMOS	Simple MOS Transistor with heating port
HeadingPMOS	Simple PMOS Transistor with heating port
HeadingNPN	Simple NPN BJT according to Ebers-Moll with heating port
HeadingPNP	Simple PNP BJT according to Ebers-Moll with heating port
Modelica.Electrical.MultiPhase A new library for multi-phase electrical circuits	

New examples

The following new examples have been added to Modelica.Electrical.Analog.Examples:

CharacteristicThyristors, CharacteristicIdealDiodes, HeatingNPN_OrGate, HeatingMOSInverter, HeatingRectifier, Rectifier, ShowSaturatingInductor ShowVariableResistor

Improved existing components

In the library Modelica.Electrical.Analog.Ideal, a knee voltage has been introduced for the components IdealThyristor, IdealGTOThyristor, IdealDiode in order that the approximation of these ideal elements is improved with not much computational effort.

In the Modelica.SIunits library, the following changes have been made:

Inductance	min=0 removed
SelfInductance	min=0 added
ThermodynamicTemperature	min=0 added

Modelica.UsersGuide.ReleaseNotes.Version_1_5

Version 1.5 (Dec. 16, 2002)

Added 55 new components. In particular, added new package **Thermal.HeatTransfer** for modeling of lumped heat transfer, added model **LossyGear** in Mechanics.Rotational to model gear efficiency and bearing friction according to a new theory in a robust way, added 10 new models in Electrical.Analog and added several other new models and improved existing models.

New components

Modelica.Blocks.	
Continuous.Der	Derivative of input (= analytic differentations)
Examples	Demonstration examples of the components of this package
Nonlinear.VariableLimiter	Limit the range of a signal with variable limits
Modelica.Blocks.Interfaces.	
RealPort	Real port (both input/output possible)
IntegerPort	Integer port (both input/output possible)
BooleanPort	Boolean port (both input/output possible)
SIMO	Single Input Multiple Output continuous control block
IntegerBlockIcon	Basic graphical layout of Integer block
IntegerMO	Multiple Integer Output continuous control block
IntegerSignalSource	Base class for continuous Integer signal source
IntegerMIBooleanMOs	Multiple Integer Input Multiple Boolean Output continuous control block with same number of inputs and outputs
BooleanMIMOs	Multiple Input Multiple Output continuous control block with same number of inputs and outputs of boolean type
BusAdaptors	Components to send signals to the bus or receive signals from the bus
Modelica.Blocks.Math.	
RealToInteger	Convert real to integer signals
IntegerToReal	Convert integer to real signals
Max	Pass through the largest signal
Min	Pass through the smallest signal
Edge	Indicates rising edge of boolean signal
BooleanChange	Indicates boolean signal changing
IntegerChange	Indicates integer signal changing
Modelica.Blocks.Sources.	
IntegerConstant	Generate constant signals of type Integer
IntegerStep	Generate step signals of type Integer
Modelica.Electrical.Analog.Basic.	
HeatingResistor	Temperature dependent electrical resistor
OpAmp	Simple nonideal model of an OpAmp with limitation
Modelica.Electrical.Analog.Ideal.	
IdealCommutingSwitch	Ideal commuting switch
IdealIntermediateSwitch	Ideal intermediate switch
ControlledIdealCommutingSwitch	Controlled ideal commuting switch
ControlledIdealIntermediateSwitch	Controlled ideal intermediate switch
IdealOpAmpLimited	Ideal operational amplifier with limitation
IdealOpeningSwitch	Ideal opener
IdealClosingSwitch	Ideal closer

ControlledIdealOpeningSwitch	Controlled ideal opener
ControlledIdealClosingSwitch	Controlled ideal closer
Modelica.Electrical.Analog.Lines.	
TLine1	Lossless transmission line (Z0, TD)
TLine2	Lossless transmission line (Z0, F, NL)
TLine2	Lossless transmission line (Z0, F)
Modelica.Icons.	
Function	Icon for a function
Record	Icon for a record
Enumeration	Icon for an enumeration
Modelica.Math.	
tempInterpol2	temporary routine for vectorized linear interpolation (will be removed)
Modelica.Mechanics.Rotational.	
Examples.LossyGearDemo1	Example to show that gear efficiency may lead to stuck motion
Examples.LossyGearDemo2	Example to show combination of LossyGear and BearingFriction
LossyGear	Gear with mesh efficiency and bearing friction (stuck/rolling possible)
Gear2	Realistic model of a gearbox (based on LossyGear)
Modelica.SIunits.	
Conversions	Conversion functions to/from non SI units and type definitions of non SI units
EnergyFlowRate	Same definition as *Power*
EnthalpyFlowRate	`Real (final quantity="EnthalpyFlowRate", final unit="W")`
Modelica.	
Thermal.HeatTransfer	1-dimensional heat transfer with lumped elements
ModelicaAdditions.Blocks.Discrete.	
TriggeredSampler	Triggered sampling of continuous signals
TriggeredMax	Compute maximum, absolute value of continuous signal at trigger instants
ModelicaAdditions.Blocks.Logical.Interfaces.	
BooleanMIRealMOs	Multiple Boolean Input Multiple Real Output continuous control block with same number of inputs and outputs
RealMIBooleanMOs	Multiple Real Input Multiple Boolean Output continuous control block with same number of inputs and outputs
ModelicaAdditions.Blocks.Logical.	
TriggeredTrapezoid	Triggered trapezoid generator
Hysteresis	Transform Real to Boolean with Hysteresis
OnOffController	On-off controller
Compare	True, if signal of inPort1 is larger than signal of inPort2
ZeroCrossing	Trigger zero crossing of input signal
ModelicaAdditions.	
Blocks.Multiplexer.Extractor	Extract scalar signal out of signal vector dependent on IntegerRealInput index
Tables.CombiTable1Ds	Table look-up in one dimension (matrix/file) with only single input

Package-specific Changes

- All example models made **encapsulated**
- Upper case constants changed to lower case (cf. Modelica.Constants)

- Introduced Modelica.SIunits.Wavelength due to typo "Wavelenght"
- Introduced ModelicaAdditions.Blocks.Logical.Interfaces.Comparison due to typo "Comparision"
- Changed these components of *.Blocks to `block` class, which have not been already of block type
- Changed *.Interfaces.RelativeSensor to `partial` models

Class-specific Changes

Modelica.SIunits

Removed `final` from quantity attribute for *Mass* and *MassFlowRate*.

Modelica.Blocks.Math.Sum

Implemented avoiding algorithm section, which would lead to expensive function calls.

Modelica.Blocks.Sources.Step

```
  block Step "Generate step signals of type Real"
    parameter Real height[:]={1} "Heights of steps";
 // parameter Real offset[:]={0} "Offsets of output signals";
 // parameter SIunits.Time startTime[:]={0} "Output = offset for time <
startTime";
 // extends Interfaces.MO          (final nout=max([size(height, 1);
size(offset, 1); size(startTime, 1)]));
    extends Interfaces.SignalSource(final nout=max([size(height, 1);
size(offset, 1); size(startTime, 1)]));
```

Modelica.Blocks.Sources.Exponentials

Replaced usage of built-in function `exp` by Modelica.Math.exp.

Modelica.Blocks.Sources.TimeTable

Interface definition changed from

```
    parameter Real table[:, :]=[0, 0; 1, 1; 2, 4] "Table matrix (time = first
column)";
```

to

```
    parameter Real table[:, 2]=[0, 0; 1, 1; 2, 4] "Table matrix (time = first
column)";
```

Did the same for subfunction *getInterpolationCoefficients*.

Bug in *getInterpolationCoefficients* for startTime <> 0 fixed:

```
        ...
        end if;
      end if;
      // Take into account startTime "a*(time - startTime) + b"
      b := b - a*startTime;
    end getInterpolationCoefficients;
```

Modelica.Blocks.Sources.BooleanStep

```
  block BooleanStep "Generate step signals of type Boolean"
    parameter SIunits.Time startTime[:]={0} "Time instants of steps";
    parameter Boolean startValue[size(startTime, 1)]=fill(false, size(startTime,
1)) "Output before startTime";
    extends Interfaces.BooleanSignalSource(final nout=size(startTime, 1));
  equation
    for i in 1:nout loop
 //   outPort.signal[i] = time >= startTime[i];
```

```
      outPort.signal[i] = if time >= startTime[i] then not startValue[i] else
startValue[i];
    end for;
  end BooleanStep;
```

Modelica.Electrical.Analog

Corrected table of values and default for Beta by dividing them by 1000 (consistent with the values used in the NAND-example model):

- Semiconductors.PMOS
- Semiconductors.NMOS

Corrected parameter defaults, unit and description for TrapezoidCurrent. This makes the parameters consistent with their use in the model. Models specifying parameter values are not changed. Models not specifying parameter values did not generate trapezoids previously.

Icon layer background changed from transparent to white:

- Basic.Gyrator
- Basic.EMF
- Ideal.Idle
- Ideal.Short

Basic.Transformer: Replaced invalid escape characters '\ ' and '\[newline]' in documentation by '|'.

Modelica.Mechanics.Rotational

Removed arrows and names documentation from flanges in diagram layer

Modelica.Mechanics.Rotational.Interfaces.FrictionBase

Modelica.Mechanics.Rotational.Position

Replaced `reinit` by `initial equation`

Modelica.Mechanics.Rotational.RelativeStates

Bug corrected by using modifier `stateSelect = StateSelect.prefer` as implementation

Modelica.Mechanics.Translational.Interfaces.flange_b

Attribute **fillColor=7** added to Rectangle on Icon layer, i.e. it is now filled with white and not transparent any more.

Modelica.Mechanics.Translational.Position

Replaced `reinit` by `initial equation`

Modelica.Mechanics.Translational.RelativeStates

Bug corrected by using modifier `stateSelect = StateSelect.prefer` as implementation

Modelica.Mechanics.Translational.Stop

Use `stateSelect = StateSelect.prefer`.

Modelica.Mechanics.Translational.Examples.PreLoad

Improved documentation and coordinate system used for example.

ModelicaAdditions.Blocks.Nonlinear.PadeDelay

Replaced `reinit` by `initial equation`

ModelicaAdditions.HeatFlow1D.Interfaces

Definition of connectors *Surface_a* and *Surface_b*:
`flow SIunits.HeatFlux q;` changed to `flow SIunits.HeatFlowRate q;`

MultiBody.Parts.InertialSystem

Icon corrected.

Modelica.UsersGuide.ReleaseNotes.Version_1_4

Version 1.4 (June 28, 2001)

- Several minor bugs fixed.
- New models:
  Modelica.Blocks.Interfaces.IntegerRealInput/IntegerRealOutput,
  Modelica.Blocks.Math.TwoInputs/TwoOutputs
  Modelica.Electrical.Analog.Ideal.IdealOpAmp3Pin,
  Modelica.Mechanics.Rotational.Move,
  Modelica.Mechanics.Translational.Move.

Version 1.4.1beta1 (February 12, 2001)

Adapted to Modelica 1.4

Version 1.3.2beta2 (June 20, 2000)

- New subpackage Modelica.Mechanics.**Translational**
- Changes to Modelica.Mechanics.**Rotational**:
  New elements:

```
   IdealGearR2T    Ideal gear transforming rotational in translational
motion.
   Position        Forced movement of a flange with a reference angle
                   given as input signal
   RelativeStates  Definition of relative state variables
```

- Changes to Modelica.**SIunits**:
  Introduced new types:
  type Temperature = ThermodynamicTemperature;
  types DerDensityByEnthalpy, DerDensityByPressure, DerDensityByTemperature, DerEnthalpyByPressure, DerEnergyByDensity, DerEnergyByPressure
  Attribute "final" removed from min and max values in order that these values can still be changed to narrow the allowed range of values.
  Quantity="Stress" removed from type "Stress", in order that a type "Stress" can be connected to a type "Pressure".
- Changes to Modelica.**Icons**:
  New icons for motors and gearboxes.
- Changes to Modelica.**Blocks.Interfaces**:
  Introduced a replaceable signal type into Blocks.Interfaces.RealInput/RealOutput:

```
   replaceable type SignalType = Real
```

  in order that the type of the signal of an input/output block can be changed to a physical type, for example:

```
   Sine sin1(outPort(redeclare type SignalType=Modelica.SIunits.Torque))
```

Version 1.3.1 (Dec. 13, 1999)

First official release of the library.

Modelica.UsersGuide.Contact

Contact

The development of the Modelica package is organized by

Martin Otter
Deutsches Zentrum für Luft und Raumfahrt e.V. (DLR)
Institut für Robotik und Mechatronik
Abteilung für Entwurfsorientierte Regelungstechnik
Postfach 1116
D-82230 Wessling
Germany
email: Martin.Otter@dlr.de

This library is developed by several people from the Modelica Association, see http://www.Modelica.org. In particular, the following people have directly contributed (many more people have contributed to the design):

Peter Beater University of Paderborn, Germany	Modelica.Mechanics.Translational
Dag Brück Dynasim AB, Lund, Sweden	Modelica.Utilities
Francesco Casella Politecnico di Milano, Milano, Italy	Modelica.Media
Christoph Clauss Fraunhofer Institute for Integrated Circuits, Dresden, Germany	Modelica.Electrical.Analog Modelica.Electrical.Digital
Jonas Eborn Modelon AB, Lund, Sweden	Modelica.Media
Hilding Elmqvist Dynasim AB, Lund, Sweden	Modelica.Mechanics.MultiBody Modelica.Media Modelica.StateGraph Modelica.Utilities Conversion from 1.6 to 2.0
Rüdiger Franke ABB Corporate Research, Ladenburg, German	Modelica.Media
Anton Haumer Consultant, St.Andrae-Woerdern, Austria	Modelica.Electrical.Machines Modelica.Electrical.Multiphase Modelica.Mechanics.Rotational Modelica.Thermal.FluidHeatFlow
Hans-Dieter Joos Institute of Robotics and Mechatronics DLR, German Aerospace Center, Oberpfaffenhofen, Germany	Modelica.Math.Matrices
Christian Kral arsenal research, Vienna, Austria	Modelica.Electrical.Machines Modelica.Thermal.FluidHeatFlow
Sven Erik Mattsson Dynasim AB, Lund, Sweden	Modelica.Mechanics.MultiBody
Hans Olsson Dynasim AB, Lund, Sweden	Modelica.Blocks Modelica.Math.Matrices Modelica.Utilities Conversion from 1.6 to 2.0
Martin Otter Institute of Robotics and Mechatronics DLR, German Aerospace Center, Oberpfaffenhofen, Germany	Modelica.Blocks Modelica.Mechanics.MultiBody Modelica.Mechanics.Rotational Modelica.Math Modelica.Media

	Modelica.SIunits Modelica.StateGraph Modelica.Thermal Modelica.Utilities ModelicaReference Conversion from 1.6 to 2.0
Katrin Prölß Department of Technical Thermodynamics, Technical University Hamburg-Harburg, Germany	Modelica.Media
Christoph C. Richter Institut für Thermodynamik, Technische Universität Braunschweig, Germany	Modelica.Media
André Schneider Fraunhofer Institute for Integrated Circuits, Dresden, Germany	Modelica.Electrical.Analog Modelica.Electrical.Digital
Christian Schweiger Institute of Robotics and Mechatronics, DLR, German Aerospace Center, Oberpfaffenhofen, Germany	Modelica.Mechanics.Rotational ModelicaReference Conversion from 1.6 to 2.0
Michael Tiller Emmeskay, Inc., Dearborn, MI, U.S.A, (previously Ford Motor Company, Dearborn)	Modelica.Media Modelica.Thermal
Hubertus Tummescheit Modelon AB, Lund, Sweden	Modelica.Media Modelica.Thermal
Nico Walter Master thesis at HTWK Leipzig (Prof. R. Müller) and DLR Oberpfaffenhofen, Germany	Modelica.Math.Matrices

Modelica.UsersGuide.ModelicaLicense

Modelica License (Version 1.1 of June 30, 2000)

Redistribution and use in source and binary forms, with or without modification are permitted, provided that the following conditions are met:

1. The author and copyright notices in the source files, these license conditions and the disclaimer below are (a) retained and (b) reproduced in the documentation provided with the distribution.
2. Modifications of the original source files are allowed, provided that a prominent notice is inserted in each changed file and the accompanying documentation, stating how and when the file was modified, and provided that the conditions under (1) are met.
3. It is not allowed to charge a fee for the original version or a modified version of the software, besides a reasonable fee for distribution and support. Distribution in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution is permitted, provided that it is not advertised as a product of your own.

Disclaimer

The software (sources, binaries, etc.) in their original or in a modified form are provided "as is" and the copyright holders assume no responsibility for its contents what so ever. Any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are **disclaimed**. **In no event** shall the copyright holders, or any party who modify and/or redistribute the package, **be liable** for any direct, indirect, incidental, special, exemplary, or consequential damages, arising in any way out of the use of this software, even if advised of the possibility of such damage.

Modelica.Blocks

Library of basic input/output control blocks (continuous, discrete, logical, table blocks)

Information

This library contains input/output blocks to build up block diagrams.

Main Author:
Martin Otter
Deutsches Zentrum für Luft und Raumfahrt e. V. (DLR)
Oberpfaffenhofen
Postfach 1116
D-82230 Wessling
email: Martin.Otter@dlr.de

Copyright © 1998-2007, Modelica Association and DLR.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Examples	Library of examples to demonstrate the usage of package Blocks
Continuous	Library of continuous control blocks with internal states
Discrete	Library of discrete input/output blocks with fixed sample period
Interfaces	Library of connectors and partial models for input/output blocks
Logical	Library of components with Boolean input and output signals
Math	Library of mathematical functions as input/output blocks
Nonlinear	Library of discontinuous or non-differentiable algebraic control blocks
Routing	Library of blocks to combine and extract signals
Sources	Library of signal source blocks generating Real and Boolean signals
Tables	Library of blocks to interpolate in one and two-dimensional tables
Types	Library of constants and types with choices, especially to build menus

---

Modelica.Blocks.Examples

Library of examples to demonstrate the usage of package Blocks

Information

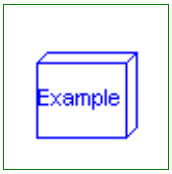
This package contains example models to demonstrate the usage of package blocks.

Package Content

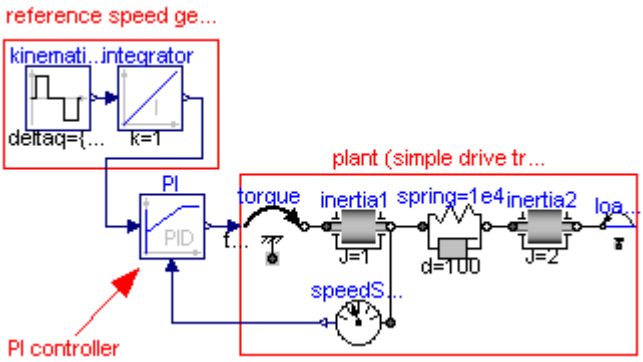
Name	Description
PID_Controller	Demonstrate usage of the Continuous.LimPID controller
ShowLogicalSources	Show logical sources and demonstrate their diagram animation
LogicalNetwork1	Example for a network of logical blocks
BusUsage	Demonstration of signal bus usage
BusUsage_Utilities	Utility models and connectors for the demonstration example

	Modelica.Blocks.Examples.BusUsage

Modelica.Blocks.Examples.PID_Controller



Demonstrate usage of the Continuous.LimPID controller



Information

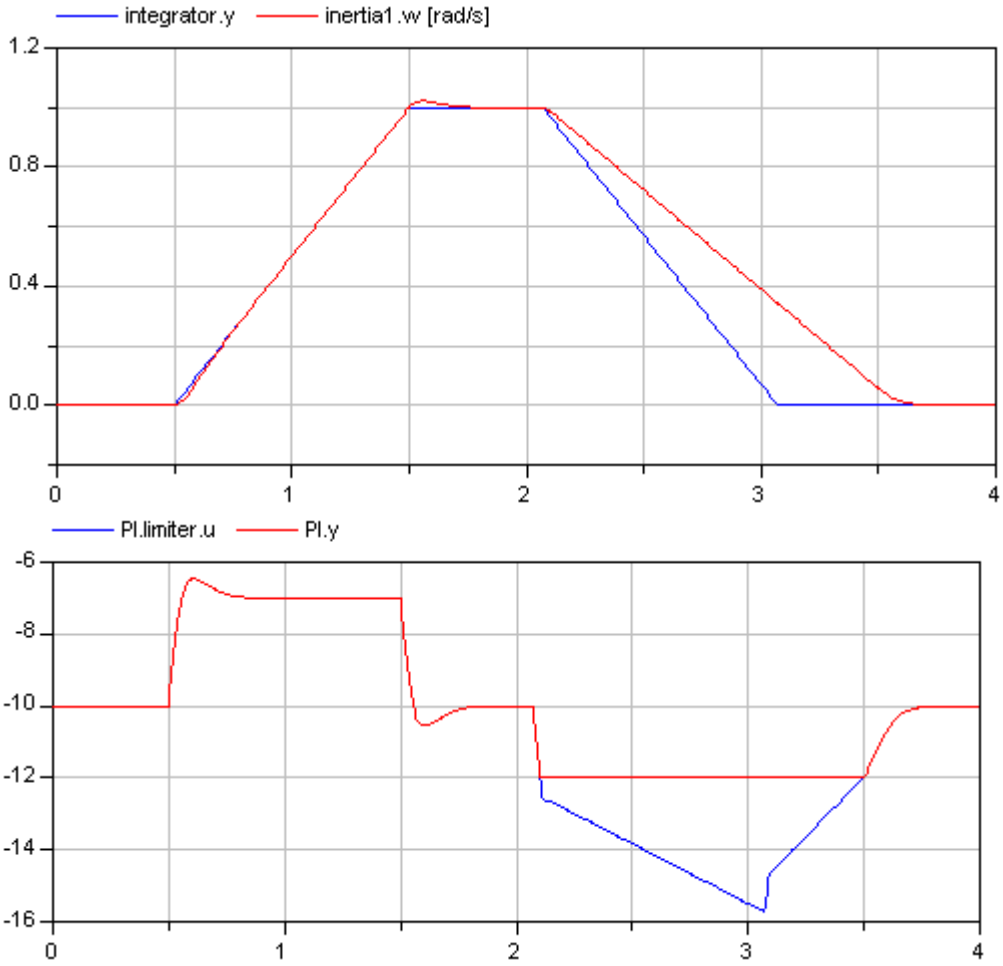
This is a simple drive train controlled by a PID controller:

- The two blocks "kinematic_PTP" and "integrator" are used to generate the reference speed (= constant acceleration phase, constant speed phase, constant deceleration phase until inertia is at rest). To check whether the system starts in steady state, the reference speed is zero until time = 0.5 s and then follows the sketched trajectory.
- The block "PI" is an instance of "Blocks.Continuous.LimPID" which is a PID controller where several practical important aspects, such as anti-windup-compensation has been added. In this case, the control block is used as PI controller.
- The output of the controller is a torque that drives a motor inertia "inertia1". Via a complöiant spring/damper component, the load inertia "inertia2" is attached. A constant external torque of 10 Nm is acting on the load inertia.

The PI controller settings included "limitAtInit=false", in order that the controller output limits of 12 Nm are removed from the initialization problem.

The PI controller is initialized in steady state (initType=SteadyState) and the drive shall also be initialized in steady state. However, it is not possible to initialize "inertia1" in SteadyState, because "der(inertia1.phi)=inertia1.w=0" is an input to the PI controller that defines that the derivative of the integrator state is zero (= the same condition that was already defined by option SteadyState of the PI controller). Furthermore, one initial condition is missing, because the absolute position of inertia1 or inertia2 is not defined. The solution shown in this examples is to initialize the angle and the angular acceleration of "inertia1".

In the following figure, results of a typical simulation are shown:



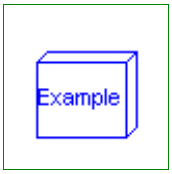
In the upper figure the reference speed (= integrator.y) and the actual speed (= inertia1.w) are shown. As can be seen, the system initializes in steady state, since no transients are present. The inertia follows the reference speed quite good until the end of the constant speed phase. Then there is a deviation. In the lower figure the reason can be seen: The output of the controller (PI.y) is in its limits. The anti-windup compensation works reasonably, since the input to the limiter (PI.limiter.u) is forced back to its limit after a transient phase.

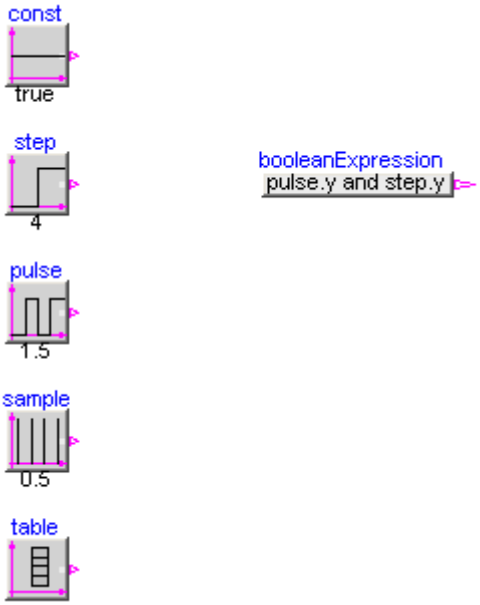
Parameters

Type	Name	Default	Description
Angle	driveAngle	1.57	Reference distance to move [rad]

Modelica.Blocks.Examples.ShowLogicalSources

Show logical sources and demonstrate their diagram animation

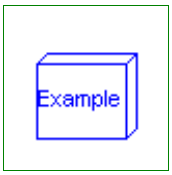




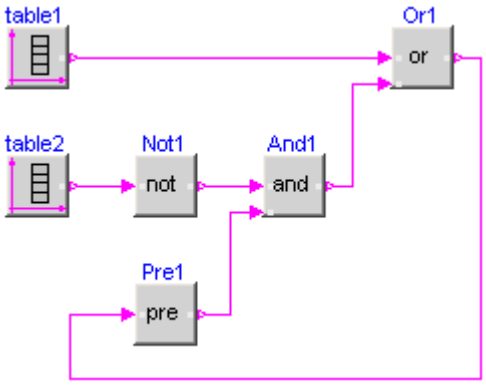
Information

This simple example demonstrates the logical sources in Modelica.Blocks.Sources and demonstrate their diagram animation (see "small circle" close to the output connector). The "booleanExpression" source shows how a logical expression can be defined in its parameter menu refering to variables available on this level of the model.

Modelica.Blocks.Examples.LogicalNetwork1



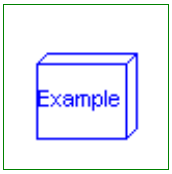
Example for a network of logical blocks



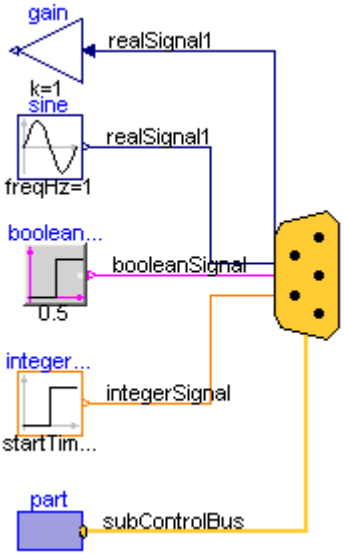
Information

This example demonstrates a network of logical blocks. Note, that the Boolean values of the input and output signals are visualized in the diagram animation, by the small "circles" close to the connectors. If a "circle" is "white", the signal is **false**. It a "circle" is "green", the signal is **true**.

Modelica.Blocks.Examples.BusUsage



Demonstration of signal bus usage

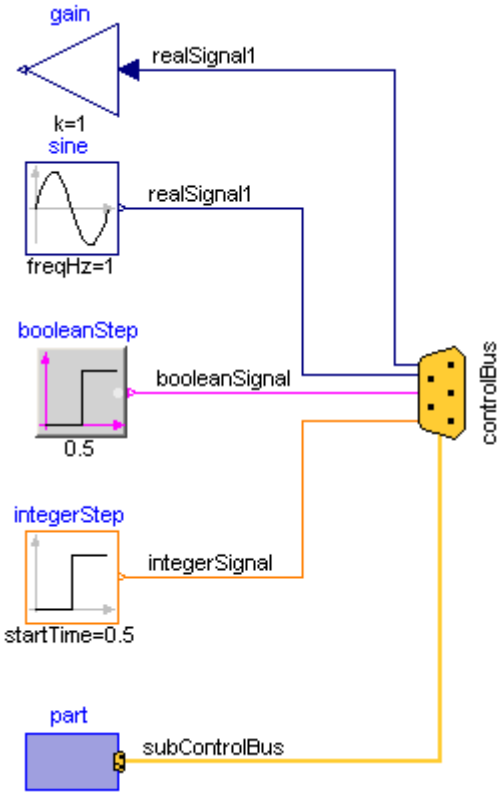


Information

Signal bus concept

In technical systems, such as vehicles, robots or satellites, many signals are exchanged between components. In a simulation system, these signals are usually modelled by signal connections of input/output blocks. Unfortunately, the signal connection structure may become very complicated, especially for hierarchical models.

The same is also true for real technical systems. To reduce complexity and get higher flexibility, many technical systems use data buses to exchange data between components. For the same reasons, it is often better to use a "signal bus" concept also in a Modelica model. This is demonstrated at hand of this model (Modelica.Blocks.Examples.BusUsage):



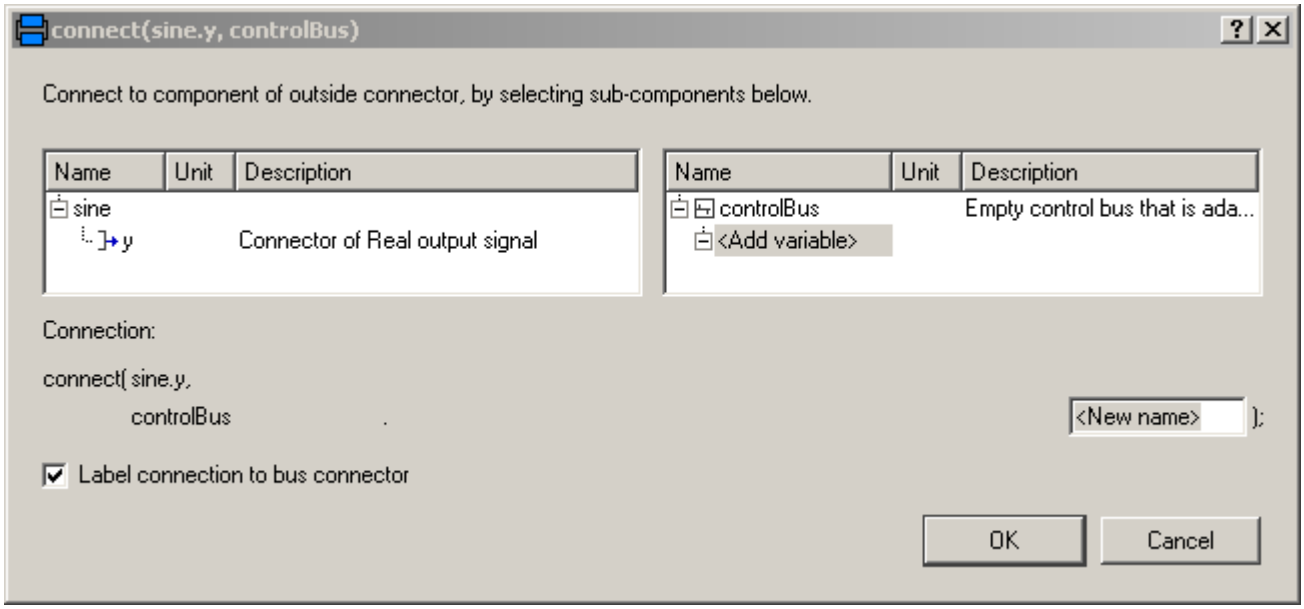
- Connector instance "controlBus" is a hierarchical connector that is used to exchange signals between different components. It is defined as "expandable connector" in order that **no** central definition of the connector is needed but is automatically constructed by the signals connected to it (see also Modelica specification 2.2.1).
- Input/output signals can be directly connected to the "controlBus". When connecting, it is optionally possible that a **label** is displayed at the connecting line, that contains the name of the variable on the controlBus to which the signal is connected.
- A component, such as "part", can be directly connected to the "controlBus", provided it has also a bus connector, or the "part" connector is a sub-connector contained in the "controlBus".

The control and sub-control bus icons are provided within Modelica.Icons. In Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces the buses for this example are defined. Both the "ControlBus" and the "SubControlBus" are **expandable** connectors that do not define any variable. For example, Interfaces.ControlBus is defined as:

```
expandable connector ControlBus
    extends Modelica.Icons.ControlBus;
    annotation (Icon(Rectangle(extent=[-20, 2; 22, -2],
                    style(rgbcolor={255,204,51}, thickness=2))));
end ControlBus;
```

Note, the "annotation" in the connector is important since the color and thickness of a connector line are taken from the first line element in the icon annotation of a connector class. Above, a small rectangle in the color of the bus is defined (and therefore this rectangle is not visible). As a result, when connecting from an instance of this connector to another connector instance, the connecting line has the color of the "ControlBus" with double width (due to "thickness=2").

An **expandable** connector is a connector where the content of the connector is constructed by the variables connected to instances of this connector. For example, if "sine.y" is connected to the "controlBus", the following menu pops-up in Dymola:



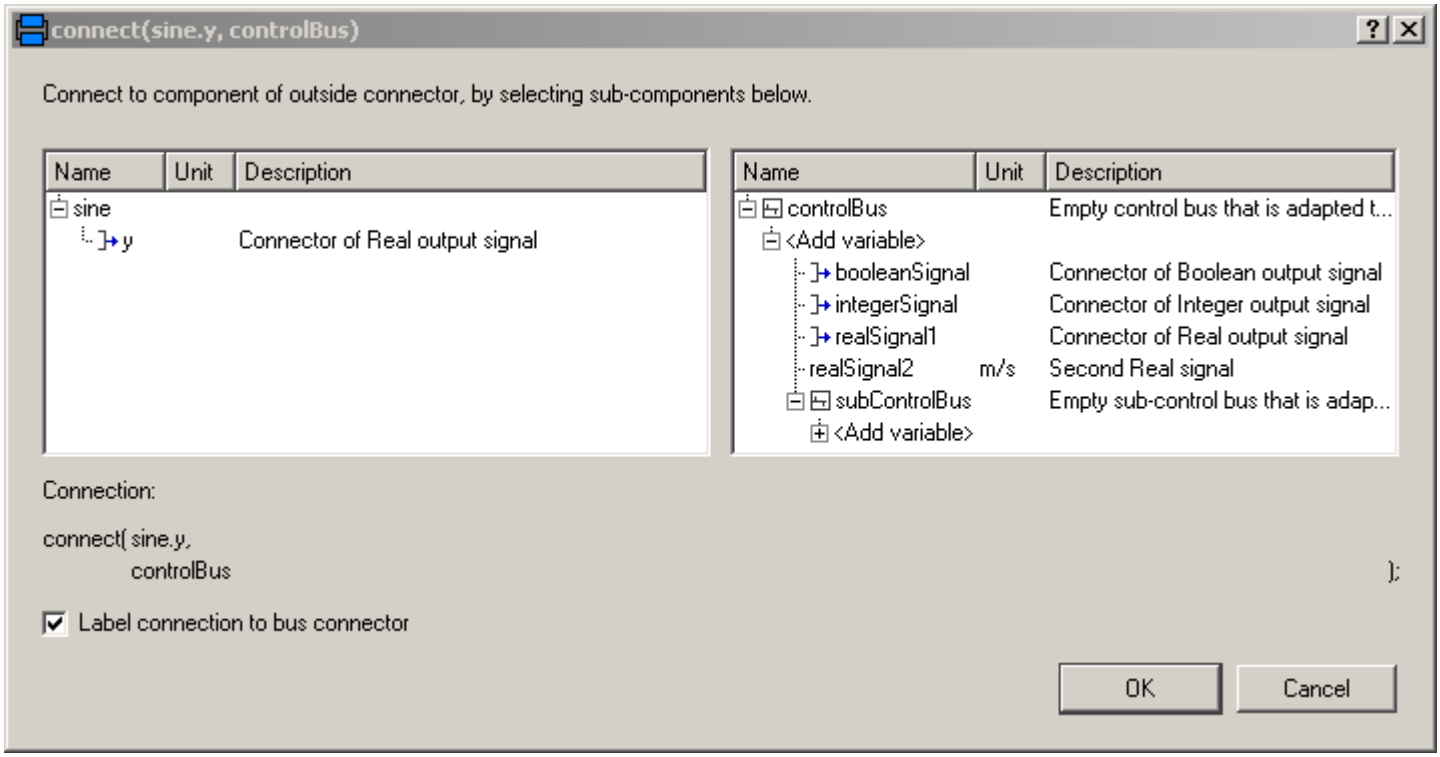
The "Add variable/New name" field allows the user to define the name of the signal on the "controlBus". When typing "realSignal1" as "New name", a connection of the form:

```
connect(sine.y, controlBus.realSignal1)
```

is generated and the "controlBus" contains the new signal "realSignal1". Modelica tools may give more support in order to list potential signals for a connection. For example, in Dymola all variables are listed in the menu that are contained in connectors which are derived by inheritance from "controlBus". Therefore, in BusUsage_Utilities.Interfaces.InternalConnectors the expected implementation of the "ControlBus" and of the "SubControlBus" are given. For example "Internal.ControlBus" is defined as:

```
expandable connector StandardControlBus
  extends BusUsage_Utilities.Interfaces.ControlBus;

  import SI = Modelica.SIunits;
  SI.AngularVelocity    realSignal1   "First Real signal";
  SI.Velocity           realSignal2   "Second Real signal";
  Integer               integerSignal "Integer signal";
  Boolean               booleanSignal "Boolean signal";
  StandardSubControlBus subControlBus "Combined signal";
end StandardControlBus;
```

Consequently, when connecting now from "sine.y" to "controlBus", the menu looks differently:



Note, even if the signals from "Internal.StandardControlBus" are listed, these are just potential signals. The user might still add different signal names.

Modelica.Blocks.Examples.BusUsage_Utilities

Utility models and connectors for the demonstration example Modelica.Blocks.Examples.BusUsage

Information

This package contains utility models and bus definitions needed for the BusUsage example.

Package Content

Name	Description
Interfaces	Interfaces specialised for this example
Part	Component with sub-control bus

Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces

Interfaces specialised for this example

Information

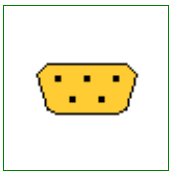
This package contains the bus definitions needed for the BusUsage example.

Package Content

Name	Description

ControlBus	Empty control bus that is adapted to the signals connected to it
SubControlBus	Empty sub-control bus that is adapted to the signals connected to it
InternalConnectors	Internal definitions that are usually not utilized (only indirectly via the expandable connectors)

Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.ControlBus

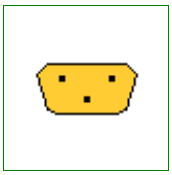


Empty control bus that is adapted to the signals connected to it

Information

This connector defines the "expandable connector" ControlBus that is used as bus in the BusUsage example. Note, this connector is "empty". When using it, the actual content is constructed by the signals connected to this bus.

Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.SubControlBus



Empty sub-control bus that is adapted to the signals connected to it

Information

This connector defines the "expandable connector" SubControlBus that is used as sub-bus in the BusUsage example. Note, this connector is "empty". When using it, the actual content is constructed by the signals connected to this bus.

Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.InternalConnectors

Internal definitions that are usually not utilized (only indirectly via the expandable connectors)

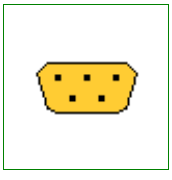
Information

This package contains the "actual" default bus definitions needed for the BusUsage example. The bus definitions in this package are the default definitions shown in the bus menu when connecting a signal to an expandable connector (here: ControlBus or SubControlBus). Usually, the connectors of this package should not be utilized by a user.

Package Content

Name	Description
StandardControlBus	Used to build up the standard control bus (do not use this connector)
StandardSubControlBus	Used to build up the standard sub-control bus (do not use this connector)

Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.InternalConnectors.StandardControlBus



Used to build up the standard control bus (do not use this connector)

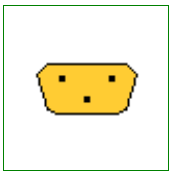
Information

This connector is used to show default signals that might be added to the ControlBus.

Contents

Type	Name	Description
AngularVelocity	realSignal1	First Real signal (angular velocity) [rad/s]
Velocity	realSignal2	Second Real signal [m/s]
Integer	integerSignal	Integer signal
Boolean	booleanSignal	Boolean signal
StandardSubControlBus	subControlBus	Combined signal

Modelica.Blocks.Examples.BusUsage_Utilities.Interfaces.InternalConnectors.StandardSubControlBus



Used to build up the standard sub-control bus (do not use this connector)

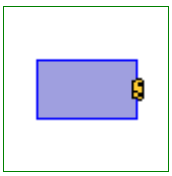
Information

This connector is used to show default signals that might be added to the SubControlBus.

Contents

Type	Name	Description
Real	myRealSignal	
Boolean	myBooleanSignal	

Modelica.Blocks.Examples.BusUsage_Utilities.Part



Component with sub-control bus

Information

This model is used to demonstrate the bus usage in example BusUsage.

Connectors

Type	Name	Description
SubControlBus	subControlBus	

Modelica.Blocks.Continuous

Library of continuous control blocks with internal states

Information

This package contains basic **continuous** input/output blocks described by differential equations.

All blocks of this package can be initialized in different ways controlled by parameter **initType**. The possible values of initType are defined in Modelica.Blocks.Types.Init:

Name	Description
Init.NoInit	no initialization (start values are used as guess values with fixed=false)
Init.SteadyState	steady state initialization (derivatives of states are zero)
Init.InitialState	Initialization with initial states
Init.InitialOutput	Initialization with initial outputs (and steady state of the states if possibles)

For backward compatibility reasons the default of all blocks is **Init.NoInit**, with the exception of Integrator and LimIntegrator where the default is **Init.InitialState** (this was the initialization defined in version 2.2 of the Modelica standard library).

In many cases, the most useful initial condition is **Init.SteadyState** because initial transients are then no longer present. The drawback is that in combination with a non-linear plant, non-linear algebraic equations occur that might be difficult to solve if appropriate guess values for the iteration variables are not provided (i.e. start values with fixed=false). However, it is often already useful to just initialize the linear blocks from the Continuous blocks library in SteadyState. This is uncritical, because only linear algebraic equations occur. If Init.NoInit is set, then the start values for the states are interpreted as **guess** values and are propagated to the states with fixed=**false**.

Note, initialization with Init.SteadyState is usually difficult for a block that contains an integrator (Integrator, LimIntegrator, PI, PID, LimPID). This is due to the basic equation of an integrator:

```
initial equation
   der(y) = 0;   // Init.SteadyState
equation
   der(y) = k*u;
```

The steady state equation leads to the condition that the input to the integrator is zero. If the input u is already (directly or indirectly) defined by another initial condition, then the initialization problem is **singular** (has none or infinitely many solutions). This situation occurs often for mechanical systems, where, e.g., u = desiredSpeed - measuredSpeed and since speed is both a state and a derivative, it is always defined by Init.InitialState or Init.SteadyState initializtion.

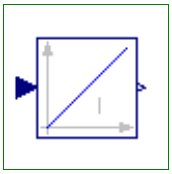
In such a case, **Init.NoInit** has to be selected for the integrator and an additional initial equation has to be added to the system to which the integrator is connected. E.g., useful initial conditions for a 1-dim. rotational inertia controlled by a PI controller are that **angle**, **speed**, and **acceleration** of the inertia are zero.

Package Content

Name	Description
Integrator	Output the integral of the input signal
LimIntegrator	Integrator with limited value of the output
Derivative	Approximated derivative block
FirstOrder	First order transfer function block (= 1 pole)
SecondOrder	Second order transfer function block (= 2 poles)
PI	Proportional-Integral controller
PID	PID-controller in additive description form
LimPID	P, PI, PD, and PID controller with limited output, anti-windup compensation and setpoint weighting
TransferFunction	Linear transfer function
StateSpace	Linear state space system
Der	Derivative of input (= analytic differentations)
LowpassButterworth	Output the input signal filtered with a low pass Butterworth filter of any order

CriticalDamping	Output the input signal filtered with an n-th order filter with critical damping

Modelica.Blocks.Continuous.Integrator



Output the integral of the input signal

Information

This blocks computes output **y** (element-wise) as *integral* of the input **u** multiplied with the gain *k*:

$$y = \frac{k}{s} u$$

It might be difficult to initialize the integrator in steady state. This is discussed in the description of package Continuous.

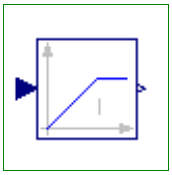
Parameters

Type	Name	Default	Description
Real	k	1	Integrator gain
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.I...	Type of initialization (InitialState and InitialOutput are identical)
Real	y_start	0	Initial or guess value of output (= state)
RealOutput	y.start	y_start	Connector of Real output signal

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal

Modelica.Blocks.Continuous.LimIntegrator



Integrator with limited value of the output

Information

This blocks computes **y** (element-wise) as *integral* of the input **u** multiplied with the gain *k*. If the integral reaches a given upper or lower *limit* and the input will drive the integral outside of this bound, the integration is halted and only restarted if the input drives the integral away from the bounds.

It might be difficult to initialize the integrator in steady state. This is discussed in the description of package Continuous.

If parameter **limitAtInit** = **false**, the limits of the integrator are removed from the initialization problem which leads to a much simpler equation system. After initialization has been performed, it is checked via an assert whether the output is in the defined limits. For backward compatibility reasons **limitAtInit** = **true**. In most cases it is best to use **limitAtInit** = **false**.

Parameters

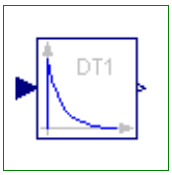
Type	Name	Default	Description

Real	k	1	Integrator gain
Real	outMax	1	Upper limit of output
Real	outMin	-outMax	Lower limit of output
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.I...	Type of initialization
Boolean	limitsAtInit	true	= false, if limits are ignored during initializiation (i.e., der(y)=k*u)
Real	y_start	0	Initial or guess value of output (must be in the limits outMin .. outMax)
RealOutput	y.start	y_start	Connector of Real output signal

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal

Modelica.Blocks.Continuous.Derivative



Approximated derivative block

Information

This blocks defines the transfer function between the input u and the output y (element-wise) as *approximated derivative*:

```
             k * s
y = ------------ * u
          T * s + 1
```

If you would like to be able to change easily between different transfer functions (FirstOrder, SecondOrder, ...) by changing parameters, use the general block **TransferFunction** instead and model a derivative block with parameters
b = {k,0}, a = {T, 1}.

If k=0, the block reduces to y=0.

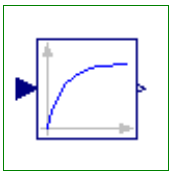
Parameters

Type	Name	Default	Description
Real	k	1	Gains
Time	T	0.01	Time constants (T>0 required; T=0 is ideal derivative block) [s]
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.N...	Type of initialization
Real	x_start	0	Initial or guess value of state
Real	y_start	0	Initial value of output (= state)

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Continuous.FirstOrder



First order transfer function block (= 1 pole)

Information

This blocks defines the transfer function between the input u and the output y (element-wise) as *first order* system:

```
               k
y = ------------ * u
      T * s + 1
```

If you would like to be able to change easily between different transfer functions (FirstOrder, SecondOrder, ...) by changing parameters, use the general block **TransferFunction** instead and model a first order SISO system with parameters
b = {k}, a = {T, 1}.

```
Example:
   parameter: k = 0.3, T = 0.4
   results in:
             0.3
      y = ----------- * u
          0.4 s + 1.0
```

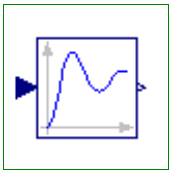
Parameters

Type	Name	Default	Description
Real	k	1	Gain
Time	T	1	Time Constant [s]
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.N...	Type of initialization (InitialState and InitialOutput are identical)
Real	y_start	0	Initial or guess value of output (= state)
RealOutput	y.start	y_start	Connector of Real output signal

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal

Modelica.Blocks.Continuous.SecondOrder



Second order transfer function block (= 2 poles)

Information

This blocks defines the transfer function between the input u and the output y (element-wise) as *second order* system:

```
                              k
y = ------------------------------------------ * u
       ( s / w )^2 + 2*D*( s / w ) + 1
```

If you would like to be able to change easily between different transfer functions (FirstOrder, SecondOrder, ...) by changing parameters, use the general model class **TransferFunction** instead and model a second order SISO system with parameters
b = {k}, a = {1/w^2, 2*D/w, 1}.

```
Example:

   parameter: k =  0.3,  w = 0.5,  D = 0.4
   results in:
                  0.3
      y = ------------------- * u
          4.0 s^2 + 1.6 s + 1
```

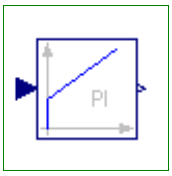
Parameters

Type	Name	Default	Description
Real	k	1	Gain
Real	w	1	Angular frequency
Real	D	1	Damping
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.N...	Type of initialization (InitialState and InitialOutput are identical)
Real	y_start	0	Initial or guess value of output (= state)
Real	yd_start	0	Initial or guess value of derivative of output (= state)
RealOutput	y.start	y_start	Connector of Real output signal

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal

Modelica.Blocks.Continuous.PI



Proportional-Integral controller

Information

This blocks defines the transfer function between the input u and the output y (element-wise) as *PI* system:

```
                1
y = k * (1 + ---) * u
              T*s
        T*s + 1
  = k * ------- * u
          T*s
```

If you would like to be able to change easily between different transfer functions (FirstOrder, SecondOrder, ...) by changing parameters, use the general model class **TransferFunction** instead and model a PI SISO system with parameters
b = {k*T, k}, a = {T, 0}.

```
Example:

   parameter: k = 0.3,  T = 0.4
```

```
results in:
                0.4 s + 1
   y = 0.3 ------------ * u
                 0.4 s
```

It might be difficult to initialize the PI component in steady state due to the integrator part. This is discussed in the description of package Continuous.

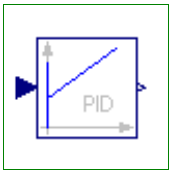
Parameters

Type	Name	Default	Description
Real	k	1	Gain
Time	T	1	Time Constant (T>0 required) [s]
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.N...	Type of initialization (SteadyState and InitialOutput are identical)
Real	x_start	0	Initial or guess value of state
Real	y_start	0	Initial value of output

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Continuous.PID



PID-controller in additive description form

Information

This is the text-book version of a PID-controller. For a more practically useful PID-controller, use block LimPID.

The PID block can be initialized in different ways controlled by parameter **initType**. The possible values of initType are defined in Modelica.Blocks.Types.InitPID. This type is identical to Types.Init, with the only exception that the additional option **DoNotUse_InitialIntegratorState** is added for backward compatibility reasons (= integrator is initialized with InitialState whereas differential part is initialized with NoInit which was the initialization in version 2.2 of the Modelica standard library).

Based on the setting of initType, the integrator (I) and derivative (D) blocks inside the PID controller are initialized according to the following table:

initType	**I.initType**	**D.initType**
NoInit	NoInit	NoInit
SteadyState	SteadyState	SteadyState
InitialState	InitialState	InitialState
InitialOutput and initial equation: y = y_start	NoInit	SteadyState
DoNotUse_InitialIntegratorState	InitialState	NoInit

In many cases, the most useful initial condition is **SteadyState** because initial transients are then no longer present. If initType = InitPID.SteadyState, then in some cases difficulties might occur. The reason is the equation of the integrator:

```
der(y) = k*u;
```

The steady state equation "der(x)=0" leads to the condition that the input u to the integrator is zero. If the input u is already (directly or indirectly) defined by another initial condition, then the initialization problem is **singular** (has none or infinitely many solutions). This situation occurs often for mechanical systems, where, e.g., u = desiredSpeed - measuredSpeed and since speed is both a state and a derivative, it is natural to initialize it with zero. As sketched this is, however, not possible. The solution is to not initialize u or the variable that is used to compute u by an algebraic equation.

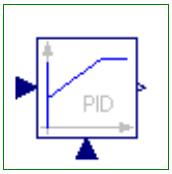
Parameters

Type	Name	Default	Description
Real	k	1	Gain
Time	Ti	0.5	Time Constant of Integrator [s]
Time	Td	0.1	Time Constant of Derivative block [s]
Real	Nd	10	The higher Nd, the more ideal the derivative block
Initialization			
Temp	initType	Modelica.Blocks.Types.InitPI...	Type of initialization
Real	xi_start	0	Initial or guess value value for integrator output (= integrator state)
Real	xd_start	0	Initial or guess value for state of derivative block
Real	y_start	0	Initial value of output

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Continuous.LimPID



P, PI, PD, and PID controller with limited output, anti-windup compensation and setpoint weighting

Information

Via parameter **controllerType** either **P**, **PI**, **PD**, or **PID** can be selected. If, e.g., PI is selected, all components belonging to the D-part are removed from the block (via conditional declarations). The example model Modelica.Blocks.Examples.PID_Controller demonstrates the usage of this controller. Several practical aspects of PID controller design are incorporated according to chapter 3 of the book:

Astroem K.J., and Haegglund T.:
PID Controllers: Theory, Design, and Tuning. Instrument Society of America, 2nd edition, 1995.
Information from: http://www.control.lth.se/publications/books/asthagg95.html

Besides the additive **proportional, integral** and **derivative** part of this controller, the following features are present:

- The output of this controller is limited. If the controller is in its limits, anti-windup compensation is activated to drive the integrator state to zero.
- The high-frequency gain of the derivative part is limited to avoid excessive amplification of measurement noise.
- Setpoint weighting is present, which allows to weight the setpoint in the proportional and the derivative part independantly from the measurement. The controller will respond to load disturbances

and measurement noise independantly of this setting (parameters wp, wd). However, setpoint changes will depend on this setting. For example, it is useful to set the setpoint weight wd for the derivative part to zero, if steps may occur in the setpoint signal.

The parameters of the controller can be manually adjusted by performing simulations of the closed loop system (= controller + plant connected together) and using the following strategy:

1. Set very large limits, e.g., yMax = Modelica.Constants.inf
2. Select a **P**-controller and manually enlarge parameter **k** (the total gain of the controller) until the closed-loop response cannot be improved any more.
3. Select a **PI**-controller and manually adjust parameters **k** and **Ti** (the time constant of the integrator). The first value of Ti can be selected, such that it is in the order of the time constant of the oscillations occuring with the P-controller. If, e.g., vibrations in the order of T=10 ms occur in the previous step, start with Ti=0.01 s.
4. If you want to make the reaction of the control loop faster (but probably less robust against disturbances and measurement noise) select a **PID**-Controller and manually adjust parameters **k**, **Ti**, **Td** (time constant of derivative block).
5. Set the limits yMax and yMin according to your specification.
6. Perform simulations such that the output of the PID controller goes in its limits. Tune **Ni** (Ni*Ti is the time constant of the anti-windup compensation) such that the input to the limiter block (= limiter.u) goes quickly enough back to its limits. If Ni is decreased, this happens faster. If Ni=infinity, the anti-windup compensation is switched off and the controller works bad.

Initialization

This block can be initialized in different ways controlled by parameter **initType**. The possible values of initType are defined in Modelica.Blocks.Types.InitPID. This type is identical to Types.Init, with the only exception that the additional option **DoNotUse_InitialIntegratorState** is added for backward compatibility reasons (= integrator is initialized with InitialState whereas differential part is initialized with NoInit which was the initialization in version 2.2 of the Modelica standard library).

Based on the setting of initType, the integrator (I) and derivative (D) blocks inside the PID controller are initialized according to the following table:

initType	**I.initType**	**D.initType**
NoInit	NoInit	NoInit
SteadyState	SteadyState	SteadyState
InitialState	InitialState	InitialState
InitialOutput and initial equation: y = y_start	NoInit	SteadyState
DoNotUse_InitialIntegratorState	InitialState	NoInit

In many cases, the most useful initial condition is **SteadyState** because initial transients are then no longer present. If initType = InitPID.SteadyState, then in some cases difficulties might occur. The reason is the equation of the integrator:

```
der(y) = k*u;
```

The steady state equation "der(x)=0" leads to the condition that the input u to the integrator is zero. If the input u is already (directly or indirectly) defined by another initial condition, then the initialization problem is **singular** (has none or infinitely many solutions). This situation occurs often for mechanical systems, where, e.g., u = desiredSpeed - measuredSpeed and since speed is both a state and a derivative, it is natural to initialize it with zero. As sketched this is, however, not possible. The solution is to not initialize u_m or the variable that is used to compute u_m by an algebraic equation.

If parameter **limitAtInit** = **false**, the limits at the output of this controller block are removed from the initialization problem which leads to a much simpler equation system. After initialization has been performed, it is checked via an assert whether the output is in the defined limits. For backward compatibility reasons **limitAtInit** = **true**. In most cases it is best to use **limitAtInit** = **false**.

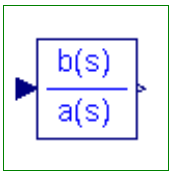
Parameters

Type	Name	Default	Description
Temp	controllerType	Modelica.Blocks.Types.Simple...	Type of controller
Real	k	1	Gain of controller
Time	Ti	0.5	Time constant of Integrator block [s]
Time	Td	0.1	Time constant of Derivative block [s]
Real	yMax	1	Upper limit of output
Real	yMin	-yMax	Lower limit of output
Real	wp	1	Set-point weight for Proportional block (0..1)
Real	wd	0	Set-point weight for Derivative block (0..1)
Real	Ni	0.9	Ni*Ti is time constant of anti-windup compensation
Real	Nd	10	The higher Nd, the more ideal the derivative block
Initialization			
Temp	initType	Modelica.Blocks.Types.InitPI...	Type of initialization
Boolean	limitsAtInit	true	= false, if limits are ignored during initializiation
Real	xi_start	0	Initial or guess value value for integrator output (= integrator state)
Real	xd_start	0	Initial or guess value for state of derivative block
Real	y_start	0	Initial value of output

Connectors

Type	Name	Description
input RealInput	u_s	Connector of setpoint input signal
input RealInput	u_m	Connector of measurement input signal
output RealOutput	y	Connector of actuator output signal

Modelica.Blocks.Continuous.TransferFunction



Linear transfer function

Information

This block defines the transfer function between the input u and the output y as (nb = dimension of b, na = dimension of a):

```
            b[1]*s^[nb-1] + b[2]*s^[nb-2] + ... + b[nb]
y(s) = ------------------------------------------------ * u(s)
            a[1]*s^[na-1] + a[2]*s^[na-2] + ... + a[na]
```

State variables **x** are defined according to **controller canonical** form. Initial values of the states can be set as start values of **x**.

Example:

```
TransferFunction g(b = {2,4}, a = {1,3});
```

results in the following transfer function:

```
     2*s + 4
y = --------- * u
```

```
      s + 3
```

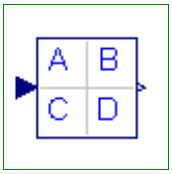
Parameters

Type	Name	Default	Description
Real	b[:]	{1}	Numerator coefficients of transfer function.
Real	a[:]	{1,1}	Denominator coefficients of transfer function.
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.N...	Type of initialization
Real	x_start[size(a, 1) - 1]	zeros(nx)	Initial or guess values of states
Real	y_start	0	Initial value of output (derivatives of y are zero upto nx-1-th derivative)

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Continuous.StateSpace



Linear state space system

Information

The State Space block defines the relation between the input u and the output y in state space form:

```
der(x) = A * x + B * u
    y  = C * x + D * u
```

The input is a vector of length nu, the output is a vector of length ny and nx is the number of states. Accordingly

```
A has the dimension: A(nx,nx),
B has the dimension: B(nx,nu),
C has the dimension: C(ny,nx),
D has the dimension: D(ny,nu)
```

Example:

```
  parameter: A = [0.12, 2;3, 1.5]
  parameter: B = [2, 7;3, 1]
  parameter: C = [0.1, 2]
  parameter: D = zeros(ny,nu)
results in the following equations:
  [der(x[1])]   [0.12  2.00] [x[1]]   [2.0  7.0] [u[1]]
  [         ] = [          ]*[    ] + [       ]*[    ]
  [der(x[2])]   [3.00  1.50] [x[2]]   [0.1  2.0] [u[2]]
                             [x[1]]             [u[1]]
       y[1]   = [0.1  2.0] * [    ] + [0  0] * [    ]
                             [x[2]]             [u[2]]
```

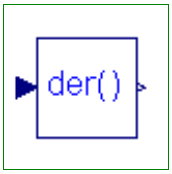
Parameters

Type	Name	Default	Description
Real	A[:, size(A, 1)]	[1, 0; 0, 1]	Matrix A of state space model
Real	B[size(A, 1), :]	[1; 1]	Matrix B of state space model
Real	C[:, size(A, 1)]	[1, 1]	Matrix C of state space model
Real	D[size(C, 1), size(B, 2)]	zeros(size(C, 1), size(B, 2))	Matrix D of state space model
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.N...	Type of initialization
Real	x_start[nx]	zeros(nx)	Initial or guess values of states
Real	y_start[ny]	zeros(ny)	Initial values of outputs (remaining states are in steady state if possible)

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y[nout]	Connector of Real output signals

Modelica.Blocks.Continuous.Der



Derivative of input (= analytic differentations)

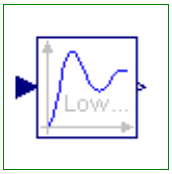
Information

Defines that the output y is the *derivative* of the input u. Note, that Modelica.Blocks.Continuous.Derivative computes the derivative in an approximate sense, where as this block computes the derivative exactly. This requires that the input u is differentiated by the Modelica translator, if this derivative is not yet present in the model.

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Continuous.LowpassButterworth



Output the input signal filtered with a low pass Butterworth filter of any order

Information

This block defines the transfer function between the input u and the output y as an n-th order low pass filter with *Butterworth* characteristics and cut-off frequency f. It is implemented as a series of second order filters and a first order filter.

If transients at the simulation start shall be avoided the states x1 and xr need to be initialized with the start value of the input signal and the states x2 need to be initialized with zeros.

```
y = PT21*PT22*...*PT2(n/2)*PT1 u
```

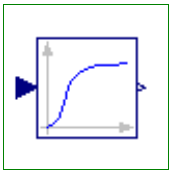
Parameters

Type	Name	Default	Description
Integer	n	2	Order of filter
Frequency	f	1	Cut-off frequency [Hz]
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.N...	Type of initialization
Real	x1_start[m]	zeros(m)	Initial or guess values of states 1 (der(x1)=x2))
Real	x2_start[m]	zeros(m)	Initial or guess values of states 2
Real	xr_start	0.0	Initial or guess value of real pole for uneven order otherwise dummy
Real	y_start	0.0	Initial value of output (states are initialized in steady state if possible)

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Continuous.CriticalDamping



Output the input signal filtered with an n-th order filter with critical damping

Information

This block defines the transfer function between the input u and the output y as an n-th order filter with *critical damping* characteristics and cut-off frequency f=1/T. It is implemented as a series of first order filters.

If transients at the simulation start shall be avoided the states x need to be initialized with the start value of the input.

```
                k
y = ------------- * u
    (T * s + 1)^n
```

Parameters

Type	Name	Default	Description
Integer	n	2	Order of filter
Frequency	f	1	Cut-off frequency [Hz]
Initialization			
Temp	initType	Modelica.Blocks.Types.Init.N...	Type of initialization
Real	x_start[n]	zeros(n)	Initial or guess values of states
Real	y_start	0.0	Initial value of output (remaining states are in steady state)

Connectors

Type	Name	Description

input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Discrete

Library of discrete input/output blocks with fixed sample period

Information

This package contains **discrete control blocks** with **fixed sample period**. Every component of this package is structured in the following way:

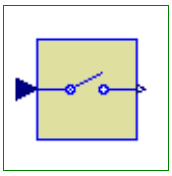
1. A component has **continuous real** input and output signals.
2. The **input** signals are **sampled** by the given sample period defined via parameter **samplePeriod**. The first sample instant is defined by parameter **startTime**.
3. The **output** signals are computed from the sampled input signals.

A **sampled data system** may consist of components of package **Discrete** and of every other purely **algebraic** input/output block, such as the components of packages **Modelica.Blocks.Math**, **Modelica.Blocks.Nonlinear** or **Modelica.Blocks.Sources**.

Package Content

Name	Description
Sampler	Ideal sampling of continuous signals
ZeroOrderHold	Zero order hold of a sampled-data system
FirstOrderHold	First order hold of a sampled-data system
UnitDelay	Unit Delay Block
TransferFunction	Discrete Transfer Function block
StateSpace	Discrete State Space block
TriggeredSampler	Triggered sampling of continuous signals
TriggeredMax	Compute maximum, absolute value of continuous signal at trigger instants

Modelica.Blocks.Discrete.Sampler



Ideal sampling of continuous signals

Information

Samples the continues input signal with a sampling rate defined via parameter **samplePeriod**.

Parameters

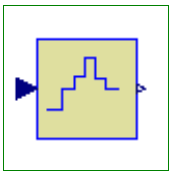
Type	Name	Default	Description
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description

input RealInput	u	Continuous input signal
output RealOutput	y	Continuous output signal

Modelica.Blocks.Discrete.ZeroOrderHold



Zero order hold of a sampled-data system

Information

The output is identical to the sampled input signal at sample time instants and holds the output at the value of the last sample instant during the sample points.

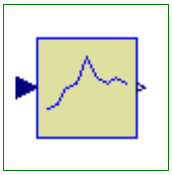
Parameters

Type	Name	Default	Description
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description
input RealInput	u	Continuous input signal
output RealOutput	y	Continuous output signal

Modelica.Blocks.Discrete.FirstOrderHold



First order hold of a sampled-data system

Information

The output signal is the extrapolation through the values of the last two sampled input signals.

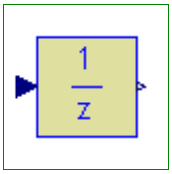
Parameters

Type	Name	Default	Description
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description
input RealInput	u	Continuous input signal
output RealOutput	y	Continuous output signal

Modelica.Blocks.Discrete.UnitDelay



Unit Delay Block

Information

This block describes a unit delay:

```
      1
y = --- * u
      z
```

that is, the output signal y is the input signal u of the previous sample instant. Before the second sample instant, the output y is identical to parameter yStart.

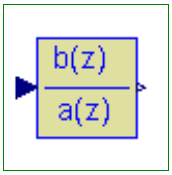
Parameters

Type	Name	Default	Description
Real	y_start	0	Initial value of output signal
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description
input RealInput	u	Continuous input signal
output RealOutput	y	Continuous output signal

Modelica.Blocks.Discrete.TransferFunction



Discrete Transfer Function block

Information

The **discrete transfer function** block defines the transfer function between the input signal u and the output signal y. The numerator has the order nb-1, the denominator has the order na-1.

```
         b(1)*z^(nb-1) + b(2)*z^(nb-2) + ... + b(nb)
y(z) = ---------------------------------------------- * u(z)
         a(1)*z^(na-1) + a(2)*z^(na-2) + ... + a(na)
```

State variables **x** are defined according to **controller canonical** form. Initial values of the states can be set as start values of **x**.

Example:

```
Blocks.Discrete.TransferFunction g(b = {2,4}, a = {1,3});
```

results in the following transfer function:

```
      2*z + 4
y = --------- * u
      z + 3
```

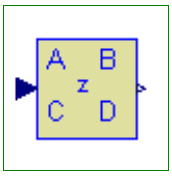
Parameters

Type	Name	Default	Description
Real	b[:]	{1}	Numerator coefficients of transfer function.
Real	a[:]	{1,1}	Denominator coefficients of transfer function.
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description
input RealInput	u	Continuous input signal
output RealOutput	y	Continuous output signal

Modelica.Blocks.Discrete.StateSpace



Discrete State Space block

Information

The **discrete state space** block defines the relation between the input u=inPort.signal and the output y=outPort.signal in state space form:

```
x = A * pre(x) + B * u
y = C * pre(x) + D * u
```

where pre(x) is the value of the discrete state x at the previous sample time instant. The input is a vector of length nu, the output is a vector of length ny and nx is the number of states. Accordingly

```
A has the dimension: A(nx,nx),
B has the dimension: B(nx,nu),
C has the dimension: C(ny,nx),
D has the dimension: D(ny,nu)
```

Example:

```
parameter: A = [0.12, 2;3, 1.5]
parameter: B = [2, 7;3, 1]
parameter: C = [0.1, 2]
parameter: D = zeros(ny,nu)

results in the following equations:
  [x[1]]   [0.12  2.00] [pre(x[1])]   [2.0  7.0] [u[1]]
  [    ] = [          ]*[         ] + [       ]*[    ]
  [x[2]]   [3.00  1.50] [pre(x[2])]   [0.1  2.0] [u[2]]
                                [pre(x[1])]            [u[1]]
      y[1]   = [0.1  2.0] * [          ] + [0  0] * [    ]
                                [pre(x[2])]            [u[2]]
```

Parameters

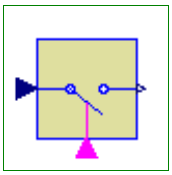
Type	Name	Default	Description
Real	A[:, size(A, 1)]	[1, 0; 0, 1]	Matrix A of state space model
Real	B[size(A, 1), :]	[1; 1]	Matrix B of state space model
Real	C[:, size(A, 1)]	[1, 1]	Matrix C of state space model
Real	D[size(C, 1), size(B, 2)]	zeros(size(C, 1), size(B, 2))	Matrix D of state space model
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description

input RealInput	u[nin]	Continuous input signals
output RealOutput	y[nout]	Continuous output signals

Modelica.Blocks.Discrete.TriggeredSampler



Triggered sampling of continuous signals

Information

Samples the continuous input signal whenever the trigger input signal is rising (i.e., trigger changes from **false** to **true**) and provides the sampled input signal as output. Before the first sampling, the output signal is equal to the initial value defined via parameter **y0**.

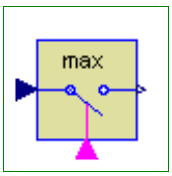
Parameters

Type	Name	Default	Description
Real	y_start	extends Real	initial value of output signal

Connectors

Type	Name	Description
input RealInput	u	Connector with an input signal of type SignalType
output RealOutput	y	Connector with an output signal of type SignalType
input BooleanInput	trigger	

Modelica.Blocks.Discrete.TriggeredMax



Compute maximum, absolute value of continuous signal at trigger instants

Information

Samples the continuous input signal whenever the trigger input signal is rising (i.e., trigger changes from **false** to **true**). The maximum, absolute value of the input signal at the sampling point is provided as output signal.

Connectors

Type	Name	Description
input RealInput	u	Connector with an input signal of type SignalType
output RealOutput	y	Connector with an output signal of type SignalType
input BooleanInput	trigger	

Modelica.Blocks.Interfaces

Library of connectors and partial models for input/output blocks

Information

This package contains interface definitions for **continuous** input/output blocks with Real, Integer and Boolean signals. Furthermore, it contains partial models for continuous and discrete blocks.

Package Content

Name	Description
RealSignal	Real port (both input/output possible)
BooleanSignal	Boolean port (both input/output possible)
IntegerSignal	Integer port (both input/output possible)
RealInput	'input Real' as connector
RealOutput	'output Real' as connector
BooleanInput	'input Boolean' as connector
BooleanOutput	'output Boolean' as connector
IntegerInput	'input Integer' as connector
IntegerOutput	'output Integer' as connector
BlockIcon	Basic graphical layout of input/output block
SO	Single Output continuous control block
MO	Multiple Output continuous control block
SISO	Single Input Single Output continuous control block
SI2SO	2 Single Input / 1 Single Output continuous control block
SIMO	Single Input Multiple Output continuous control block
MISO	Multiple Input Single Output continuous control block
MIMO	Multiple Input Multiple Output continuous control block
MIMOs	Multiple Input Multiple Output continuous control block with same number of inputs and outputs
MI2MO	2 Multiple Input / Multiple Output continuous control block
SignalSource	Base class for continuous signal source
SVcontrol	Single-Variable continuous controller
MVcontrol	Multi-Variable continuous controller
DiscreteBlockIcon	Graphical layout of discrete block component icon
DiscreteBlock	Base class of discrete control blocks
DiscreteSISO	Single Input Single Output discrete control block
DiscreteMIMO	Multiple Input Multiple Output discrete control block
DiscreteMIMOs	Multiple Input Multiple Output discrete control block
SVdiscrete	Discrete Single-Variable controller
MVdiscrete	Discrete Multi-Variable controller
BooleanBlockIcon	Basic graphical layout of Boolean block
BooleanSISO	Single Input Single Output control block with signals of type Boolean
BooleanMIMOs	Multiple Input Multiple Output continuous control block with same number of inputs and outputs of boolean type
MI2BooleanMOs	2 Multiple Input / Boolean Multiple Output block with same signal lengths

SI2BooleanSO	2 Single Input / Boolean Single Output block
BooleanSignalSource	Base class for Boolean signal sources
IntegerBlockIcon	Basic graphical layout of Integer block
IntegerSO	Single Integer Output continuous control block
IntegerMO	Multiple Integer Output continuous control block
IntegerSignalSource	Base class for continuous Integer signal source
IntegerSIBooleanSO	Integer Input Boolean Output continuous control block
IntegerMIBooleanMOs	Multiple Integer Input Multiple Boolean Output continuous control block with same number of inputs and outputs
partialBooleanBlockIcon	Basic graphical layout of logical block
partialBooleanSISO	Partial block with 1 input and 1 output Boolean signal
partialBooleanSI2SO	Partial block with 2 input and 1 output Boolean signal
partialBooleanSI3SO	Partial block with 3 input and 1 output Boolean signal
partialBooleanSI	Partial block with 1 input Boolean signal
partialBooleanSO	Partial block with 1 output Boolean signal
partialBooleanSource	Partial source block (has 1 output Boolean signal and an appropriate default icon)
partialBooleanThresholdComparison	Partial block to compare the Real input u with a threshold and provide the result as 1 Boolean output signal
partialBooleanComparison	Partial block with 2 Real input and 1 Boolean output signal (the result of a comparison of the two Real inputs
Adaptors	Obsolete package with components to send signals to a bus or receive signals from a bus (only for backward compatibility)
PartialConversionBlock	Partial block defining the interface for conversion blocks

Modelica.Blocks.Interfaces.RealSignal

Real port (both input/output possible)

Information

Connector with one signal of type Real (no icon, no input/output prefix).

Modelica.Blocks.Interfaces.BooleanSignal

Boolean port (both input/output possible)

Information

Connector with one signal of type Boolean (no icon, no input/output prefix).

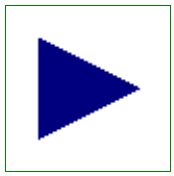
Modelica.Blocks.Interfaces.IntegerSignal

Integer port (both input/output possible)

Information

Connector with one signal of type Icon (no icon, no input/output prefix).

Modelica.Blocks.Interfaces.RealInput



'input Real' as connector

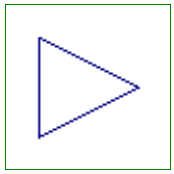
Information

Connector with one input signal of type Real.

Parameters

Type	Name	Default	Description
replaceable type SignalType		Real	

Modelica.Blocks.Interfaces.RealOutput



'output Real' as connector

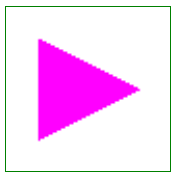
Information

Connector with one output signal of type Real.

Parameters

Type	Name	Default	Description
replaceable type SignalType		Real	

Modelica.Blocks.Interfaces.BooleanInput

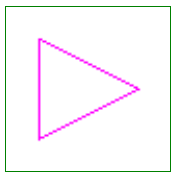


'input Boolean' as connector

Information

Connector with one input signal of type Boolean.

Modelica.Blocks.Interfaces.BooleanOutput

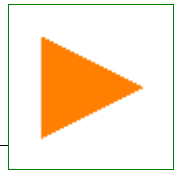


'output Boolean' as connector

Information

Connector with one output signal of type Boolean.

Modelica.Blocks.Interfaces.IntegerInput

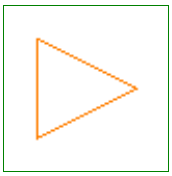


'input Integer' as connector

Information

Connector with one input signal of type Integer.

Modelica.Blocks.Interfaces.IntegerOutput

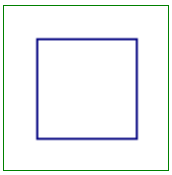


'output Integer' as connector

Information

Connector with one output signal of type Integer.

Modelica.Blocks.Interfaces.BlockIcon

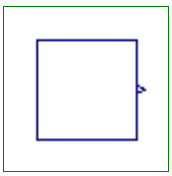


Basic graphical layout of input/output block

Information

Block that has only the basic icon for an input/output block (no declarations, no equations). Most blocks of package Modelica.Blocks inherit directly or indirectly from this block.

Modelica.Blocks.Interfaces.SO



Single Output continuous control block

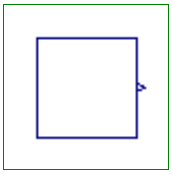
Information

Block has one continuous Real output signal.

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Interfaces.MO



Multiple Output continuous control block

Information

Block has one continuous Real output signal vector.

Parameters

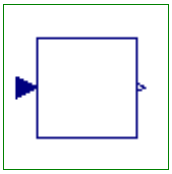
Type	Name	Default	Description
Integer	nout	1	Number of outputs

Connectors

Type	Name	Description

output RealOutput	y[nout]	Connector of Real output signals

Modelica.Blocks.Interfaces.SISO



Single Input Single Output continuous control block

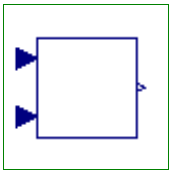
Information

Block has one continuous Real input and one continuous Real output signal.

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Interfaces.SI2SO



2 Single Input / 1 Single Output continuous control block

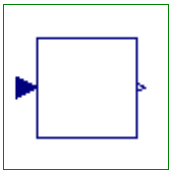
Information

Block has two continuous Real input signals u1 and u2 and one continuous Real output signal y.

Connectors

Type	Name	Description
input RealInput	u1	Connector of Real input signal 1
input RealInput	u2	Connector of Real input signal 2
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Interfaces.SIMO



Single Input Multiple Output continuous control block

Information

Block has one continuous Real input signal and a vector of continuous Real output signals.

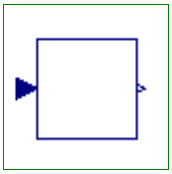
Parameters

Type	Name	Default	Description
Integer	nout	1	Number of outputs

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y[nout]	Connector of Real output signals

Modelica.Blocks.Interfaces.MISO



Multiple Input Single Output continuous control block

Information

Block has a vector of continuous Real input signals and one continuous Real output signal.

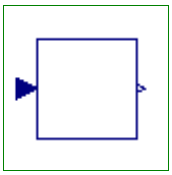
Parameters

Type	Name	Default	Description
Integer	nin	1	Number of inputs

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Interfaces.MIMO



Multiple Input Multiple Output continuous control block

Information

Block has a continuous Real input and a continuous Real output signal vector. The signal sizes of the input and output vector may be different.

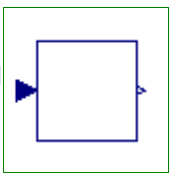
Parameters

Type	Name	Default	Description
Integer	nin	1	Number of inputs
Integer	nout	1	Number of outputs

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y[nout]	Connector of Real output signals

Modelica.Blocks.Interfaces.MIMOs



Multiple Input Multiple Output continuous control block with same number of inputs and outputs

Information

Block has a continuous Real input and a continuous Real output signal vector where the signal sizes of the input and output vector are identical.

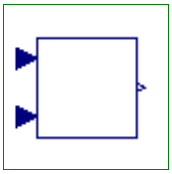
Parameters

Type	Name	Default	Description
Integer	n	1	Number of inputs (= number of outputs)

Connectors

Type	Name	Description
input RealInput	u[n]	Connector of Real input signals
output RealOutput	y[n]	Connector of Real output signals

Modelica.Blocks.Interfaces.MI2MO



2 Multiple Input / Multiple Output continuous control block

Information

Block has two continuous Real input vectors u1 and u2 and one continuous Real output vector y. All vectors have the same number of elements.

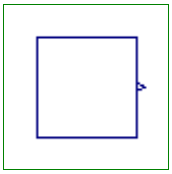
Parameters

Type	Name	Default	Description
Integer	n	1	Dimension of input and output vectors.

Connectors

Type	Name	Description
input RealInput	u1[n]	Connector 1 of Real input signals
input RealInput	u2[n]	Connector 2 of Real input signals
output RealOutput	y[n]	Connector of Real output signals

Modelica.Blocks.Interfaces.SignalSource



Base class for continuous signal source

Information

Basic block for Real sources of package Blocks.Sources. This component has one continuous Real output signal y and two parameters (offset, startTime) to shift the generated signal.

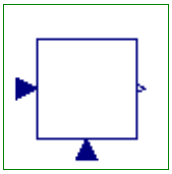
Parameters

Type	Name	Default	Description
Real	offset	0	Offset of output signal y
Time	startTime	0	Output y = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Interfaces.SVcontrol



Single-Variable continuous controller

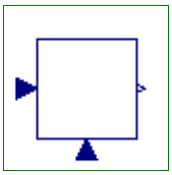
Information

Block has two continuous Real input signals and one continuous Real output signal. The block is designed to be used as base class for a corresponding controller.

Connectors

Type	Name	Description
input RealInput	u_s	Connector of setpoint input signal
input RealInput	u_m	Connector of measurement input signal
output RealOutput	y	Connector of actuator output signal

Modelica.Blocks.Interfaces.MVcontrol



Multi-Variable continuous controller

Information

Block has two continuous Real input signal vectors and one continuous Real output signal vector. The block is designed to be used as base class for a corresponding controller.

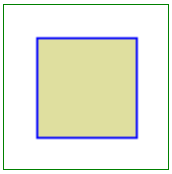
Parameters

Type	Name	Default	Description
Integer	nu_s	1	Number of setpoint inputs
Integer	nu_m	1	Number of measurement inputs
Integer	ny	1	Number of actuator outputs

Connectors

Type	Name	Description
input RealInput	u_s[nu_s]	Connector of setpoint input signals
input RealInput	u_m[nu_m]	Connector of measurement input signals
output RealOutput	y[ny]	Connector of actuator output signals

Modelica.Blocks.Interfaces.DiscreteBlockIcon

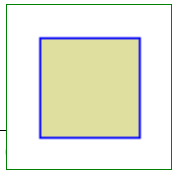


Graphical layout of discrete block component icon

Information

Block that has only the basic icon for an input/output, discrete block (no declarations, no equations), e.g., from Blocks.Discrete.

Modelica.Blocks.Interfaces.DiscreteBlock



Base class of discrete control blocks

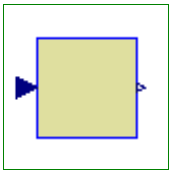
Information

Basic definitions of a discrete block of library Blocks.Discrete.

Parameters

Type	Name	Default	Description
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Modelica.Blocks.Interfaces.DiscreteSISO



Single Input Single Output discrete control block

Information

Block has one continuous input and one continuous output signal which are sampled due to the defined **samplePeriod** parameter.

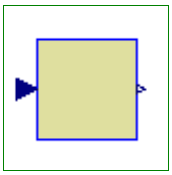
Parameters

Type	Name	Default	Description
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description
input RealInput	u	Continuous input signal
output RealOutput	y	Continuous output signal

Modelica.Blocks.Interfaces.DiscreteMIMO



Multiple Input Multiple Output discrete control block

Information

Block has a continuous input and a continuous output signal vector which are sampled due to the defined **samplePeriod** parameter.

Parameters

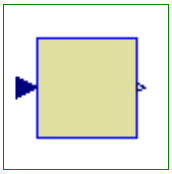
Type	Name	Default	Description
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]
Integer	nin	1	Number of inputs
Integer	nout	1	Number of outputs

Connectors

Type	Name	Description
input RealInput	u[nin]	Continuous input signals

output RealOutput	y[nout]	Continuous output signals

Modelica.Blocks.Interfaces.DiscreteMIMOs



Multiple Input Multiple Output discrete control block

Information

Block has a continuous input and a continuous output signal vector where the signal sizes of the input and output vector are identical. These signals are sampled due to the defined **samplePeriod** parameter.

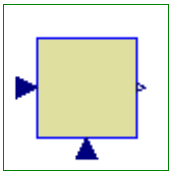
Parameters

Type	Name	Default	Description
Integer	n	1	Number of inputs (= number of outputs)
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description
input RealInput	u[n]	Continuous input signals
output RealOutput	y[n]	Continuous output signals

Modelica.Blocks.Interfaces.SVdiscrete



Discrete Single-Variable controller

Information

Block has two continuous Real input signals and one continuous Real output signal that are sampled due to the defined **samplePeriod** parameter. The block is designed to be used as base class for a corresponding controller.

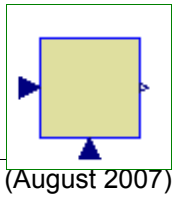
Parameters

Type	Name	Default	Description
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]

Connectors

Type	Name	Description
input RealInput	u_s	Continuous scalar setpoint input signal
input RealInput	u_m	Continuous scalar measurement input signal
output RealOutput	y	Continuous scalar actuator output signal

Modelica.Blocks.Interfaces.MVdiscrete



Discrete Multi-Variable controller

Information

Block has two continuous Real input signal vectors and one continuous Real output signal vector. The vector signals are sampled due to the defined **samplePeriod** parameter. The block is designed to be used as base class for a corresponding controller.

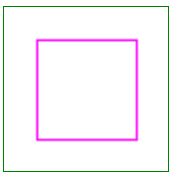
Parameters

Type	Name	Default	Description
Time	samplePeriod	0.1	Sample period of component [s]
Time	startTime	0	First sample time instant [s]
Integer	nu_s	1	Number of setpoint inputs
Integer	nu_m	1	Number of measurement inputs
Integer	ny	1	Number of actuator outputs

Connectors

Type	Name	Description
input RealInput	u_s[nu_s]	Continuous setpoint input signals
input RealInput	u_m[nu_m]	Continuous measurement input signals
output RealOutput	y[ny]	Continuous actuator output signals

Modelica.Blocks.Interfaces.BooleanBlockIcon

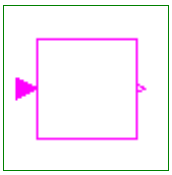


Basic graphical layout of Boolean block

Information

Block that has only the basic icon for an input/output, Boolean block (no declarations, no equations).

Modelica.Blocks.Interfaces.BooleanSISO



Single Input Single Output control block with signals of type Boolean

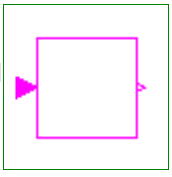
Information

Block has one continuous Boolean input and one continuous Boolean output signal.

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.BooleanMIMOs



Multiple Input Multiple Output continuous control block with same number of inputs and outputs of boolean type

Information

Block has a continuous Boolean input and a continuous Boolean output signal vector where the signal sizes of the input and output vector are identical.

Parameters

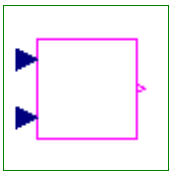
Type	Name	Default	Description
Integer	n	1	Number of inputs (= number of outputs)

Connectors

Type	Name	Description
input BooleanInput	u[n]	Connector of Boolean input signals
output BooleanOutput	y[n]	Connector of Boolean output signals

---

Modelica.Blocks.Interfaces.MI2BooleanMOs



2 Multiple Input / Boolean Multiple Output block with same signal lengths

Information

Block has two Boolean input vectors u1 and u2 and one Boolean output vector y. All vectors have the same number of elements.

Parameters

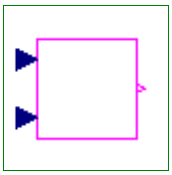
Type	Name	Default	Description
Integer	n	1	Dimension of input and output vectors.

Connectors

Type	Name	Description
input RealInput	u1[n]	Connector 1 of Boolean input signals
input RealInput	u2[n]	Connector 2 of Boolean input signals
output BooleanOutput	y[n]	Connector of Boolean output signals

---

Modelica.Blocks.Interfaces.SI2BooleanSO



2 Single Input / Boolean Single Output block

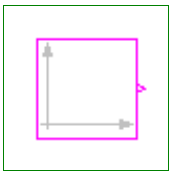
Information

Block has two Boolean input signals u1 and u2 and one Boolean output signal y.

Connectors

Type	Name	Description
input RealInput	u1	Connector 1 of Boolean input signals
input RealInput	u2	Connector 2 of Boolean input signals
output BooleanOutput	y	Connector of Boolean output signals

Modelica.Blocks.Interfaces.BooleanSignalSource



Base class for Boolean signal sources

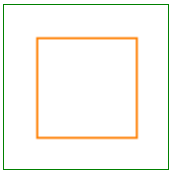
Information

Basic block for Boolean sources of package Blocks.Sources. This component has one continuous Boolean output signal y.

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.IntegerBlockIcon

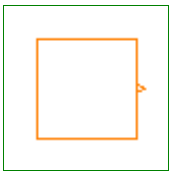


Basic graphical layout of Integer block

Information

Block that has only the basic icon for an input/output, Integer block (no declarations, no equations).

Modelica.Blocks.Interfaces.IntegerSO



Single Integer Output continuous control block

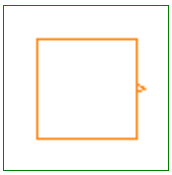
Information

Block has one continuous Integer output signal.

Connectors

Type	Name	Description
output IntegerOutput	y	Connector of Integer output signal

Modelica.Blocks.Interfaces.IntegerMO



Multiple Integer Output continuous control block

Information

Block has one continuous Integer output signal vector.

Parameters

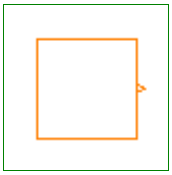
Type	Name	Default	Description
Integer	nout	1	Number of outputs

Connectors

Type	Name	Description
output IntegerOutput	y[nout]	Connector of Integer output signals

---

Modelica.Blocks.Interfaces.IntegerSignalSource



Base class for continuous Integer signal source

Information

Basic block for Integer sources of package Blocks.Sources. This component has one continuous Integer output signal y and two parameters (offset, startTime) to shift the generated signal.

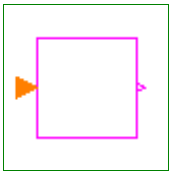
Parameters

Type	Name	Default	Description
Integer	offset	0	Offset of output signal y
Time	startTime	0	Output y = offset for time < startTime [s]

Connectors

Type	Name	Description
output IntegerOutput	y	Connector of Integer output signal

Modelica.Blocks.Interfaces.IntegerSIBooleanSO



Integer Input Boolean Output continuous control block

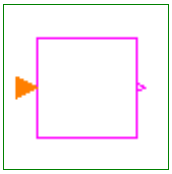
Information

Block has a continuous Integer input and a continuous Boolean output signal.

Connectors

Type	Name	Description
input IntegerInput	u	Connector of Integer input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.IntegerMIBooleanMOs



Multiple Integer Input Multiple Boolean Output continuous control block with same number of inputs and outputs

Information

Block has a continuous Integer input and a continuous Boolean output signal vector where the signal sizes of the input and output vector are identical.

Parameters

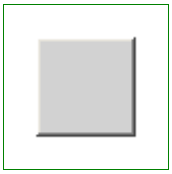
Type	Name	Default	Description
Integer	n	1	Number of inputs (= number of outputs)

Connectors

Type	Name	Description
input IntegerInput	u[n]	Connector of Integer input signals
output BooleanOutput	y[n]	Connector of Boolean output signals

---

Modelica.Blocks.Interfaces.partialBooleanBlockIcon

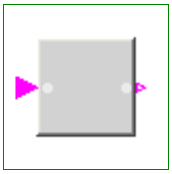


Basic graphical layout of logical block

Information

Block that has only the basic icon for an input/output, Boolean block (no declarations, no equations) used especially in the Blocks.Logical library.

Modelica.Blocks.Interfaces.partialBooleanSISO



Partial block with 1 input and 1 output Boolean signal

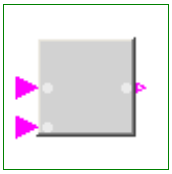
Information

Block has one continuous Boolean input and one continuous Boolean output signal with a 3D icon (e.g. used in Blocks.Logical library).

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.partialBooleanSI2SO



Partial block with 2 input and 1 output Boolean signal

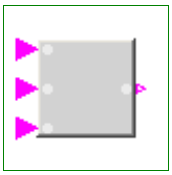
Information

Block has two continuous Boolean input and one continuous Boolean output signal with a 3D icon (e.g. used in Blocks.Logical library).

Connectors

Type	Name	Description
input BooleanInput	u1	Connector of first Boolean input signal
input BooleanInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.partialBooleanSI3SO



Partial block with 3 input and 1 output Boolean signal

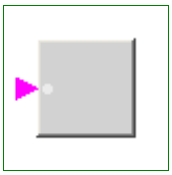
Information

Block has three continuous Boolean input and one continuous Boolean output signal with a 3D icon (e.g. used in Blocks.Logical library).

Connectors

Type	Name	Description
input BooleanInput	u1	Connector of first Boolean input signal
input BooleanInput	u2	Connector of second Boolean input signal
input BooleanInput	u3	Connector of third Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.partialBooleanSI



Partial block with 1 input Boolean signal

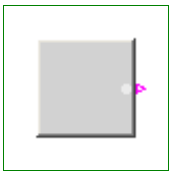
Information

Block has one continuous Boolean input signal with a 3D icon (e.g. used in Blocks.Logical library).

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal

Modelica.Blocks.Interfaces.partialBooleanSO



Partial block with 1 output Boolean signal

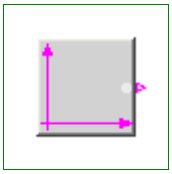
Information

Block has one continuous Boolean output signal with a 3D icon (e.g. used in Blocks.Logical library).

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.partialBooleanSource



Partial source block (has 1 output Boolean signal and an appropriate default icon)

Information

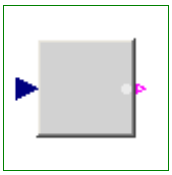
Basic block for Boolean sources of package Blocks.Sources. This component has one continuous Boolean

output signal y and a 3D icon (e.g. used in Blocks.Logical library).

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.partialBooleanThresholdComparison



Partial block to compare the Real input u with a threshold and provide the result as 1 Boolean output signal

Information

Block has one continuous Real input and one continuous Boolean output signal as well as a 3D icon (e.g. used in Blocks.Logical library).

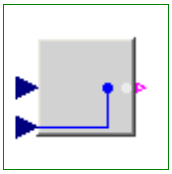
Parameters

Type	Name	Default	Description
Real	threshold	0	Comparison with respect to threshold

Connectors

Type	Name	Description
input RealInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.partialBooleanComparison



Partial block with 2 Real input and 1 Boolean output signal (the result of a comparison of the two Real inputs

Information

Block has two continuous Real input and one continuous Boolean output signal as a result of the comparision of the two input signals. The block has a 3D icon (e.g. used in Blocks.Logical library).

Connectors

Type	Name	Description
input RealInput	u1	Connector of first Boolean input signal
input RealInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Interfaces.Adaptors

Obsolete package with components to send signals to a bus or receive signals from a bus (only for backward compatibility)

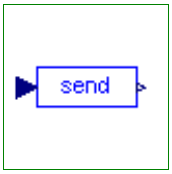
Information

The components of this package should no longer be used. They are only provided for backward compatibility. It is much more convenient and more powerful to use "expandable connectors" for signal buses, see example BusUsage.

Package Content

Name	Description
SendReal	Obsolete block to send Real signal to bus
SendBoolean	Obsolete block to send Boolean signal to bus
SendInteger	Obsolete block to send Integer signal to bus
ReceiveReal	Obsolete block to receive Real signal from bus
ReceiveBoolean	Obsolete block to receive Boolean signal from bus
ReceiveInteger	Obsolete block to receive Integer signal from bus
AdaptorReal	Completely obsolete adaptor between 'old' and 'new' Real signal connectors (only for backward compatibility)
AdaptorBoolean	Completely obsolete adaptor between 'old' and 'new' Boolean signal connectors (only for backward compatibility)
AdaptorInteger	Completely obsolete adaptor between 'old' and 'new' Integer signal connectors (only for backward compatibility)

Modelica.Blocks.Interfaces.Adaptors.SendReal



Obsolete block to send Real signal to bus

Information

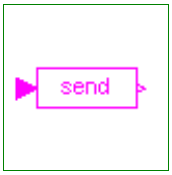
Obsolete block that was previously used to connect a Real signal to a signal in a connector. This block is only provided for backward compatibility.

It is much more convenient and more powerful to use "expandable connectors" for signal buses, see example BusUsage.

Connectors

Type	Name	Description
output RealOutput	toBus	Output signal to be connected to bus
input RealInput	u	Input signal to be send to bus

Modelica.Blocks.Interfaces.Adaptors.SendBoolean



Obsolete block to send Boolean signal to bus

Information

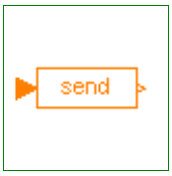
Obsolete block that was previously used to connect a Boolean signal to a signal in a connector. This block is only provided for backward compatibility.

It is much more convenient and more powerful to use "expandable connectors" for signal buses, see example BusUsage.

Connectors

Type	Name	Description
output BooleanOutput	toBus	Output signal to be connected to bus
input BooleanInput	u	Input signal to be send to bus

Modelica.Blocks.Interfaces.Adaptors.SendInteger



Obsolete block to send Integer signal to bus

Information

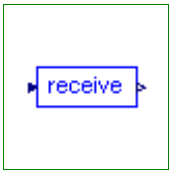
Obsolete block that was previously used to connect an Integer signal to a signal in a connector. This block is only provided for backward compatibility.

It is much more convenient and more powerful to use "expandable connectors" for signal buses, see example BusUsage.

Connectors

Type	Name	Description
output IntegerOutput	toBus	Output signal to be connected to bus
input IntegerInput	u	Input signal to be send to bus

Modelica.Blocks.Interfaces.Adaptors.ReceiveReal



Obsolete block to receive Real signal from bus

Information

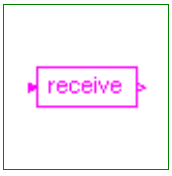
Obsolete block that was previously used to connect a Real signal in a connector to an input of a block. This block is only provided for backward compatibility.

It is much more convenient and more powerful to use "expandable connectors" for signal buses, see example BusUsage.

Connectors

Type	Name	Description
input RealInput	fromBus	To be connected with signal on bus
output RealOutput	y	Output signal to be received from bus

Modelica.Blocks.Interfaces.Adaptors.ReceiveBoolean



Obsolete block to receive Boolean signal from bus

Information

Obsolete block that was previously used to connect a Boolean signal in a connector to an input of a block. This block is only provided for backward compatibility.

It is much more convenient and more powerful to use "expandable connectors" for signal buses, see example BusUsage.

Connectors

Type	Name	Description
input BooleanInput	fromBus	To be connected with signal on bus
output BooleanOutput	y	Output signal to be received from bus

Modelica.Blocks.Interfaces.Adaptors.ReceiveInteger

Obsolete block to receive Integer signal from bus

Information

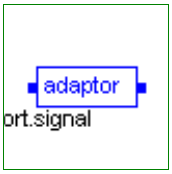
Obsolete block that was previously used to connect an Integer signal in a connector to an input of a block. This block is only provided for backward compatibility.

It is much more convenient and more powerful to use "expandable connectors" for signal buses, see example BusUsage.

Connectors

Type	Name	Description
input IntegerInput	fromBus	To be connected with signal on bus
output IntegerOutput	y	Output signal to be received from bus

Modelica.Blocks.Interfaces.Adaptors.AdaptorReal



Completely obsolete adaptor between 'old' and 'new' Real signal connectors (only for backward compatibility)

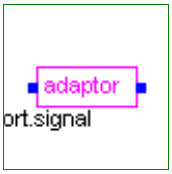
Information

Completely obsolete adaptor between the Real signal connector of version 1.6 and version ≥ 2.1 of the Modelica Standard Library. This block is only provided for backward compatibility.

Connectors

Type	Name	Description
RealSignal	newReal	Connector of Modelica version 2.1
RealPort	oldReal	Connector of Modelica version 1.6

Modelica.Blocks.Interfaces.Adaptors.AdaptorBoolean



Completely obsolete adaptor between 'old' and 'new' Boolean signal connectors (only for backward compatibility)

Information

Completely obsolete adaptor between the Real signal connector of version 1.6 and version ≥ 2.1 of the Modelica Standard Library. This block is only provided for backward compatibility.

Connectors

Type	Name	Description
BooleanSignal	newBoolean	Connector of Modelica version 2.1
BooleanPort	oldBoolean	Connector of Modelica version 1.6

Modelica.Blocks.Interfaces.Adaptors.AdaptorInteger

Completely obsolete adaptor between 'old' and 'new' Integer signal connectors (only for backward compatibility)

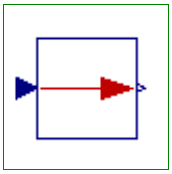
Information

Completely obsolete adaptor between the Real signal connector of version 1.6 and version ≥ 2.1 of the Modelica Standard Library. This block is only provided for backward compatibility.

Connectors

Type	Name	Description
IntegerSignal	newInteger	Connector of Modelica version 2.1
IntegerPort	oldInteger	Connector of Modelica version 1.6

Modelica.Blocks.Interfaces.PartialConversionBlock



Partial block defining the interface for conversion blocks

Information

This block defines the interface of a conversion block that converts from one unit into another one.

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal to be converted
output RealOutput	y	Connector of Real output signal containing input signal u in another unit

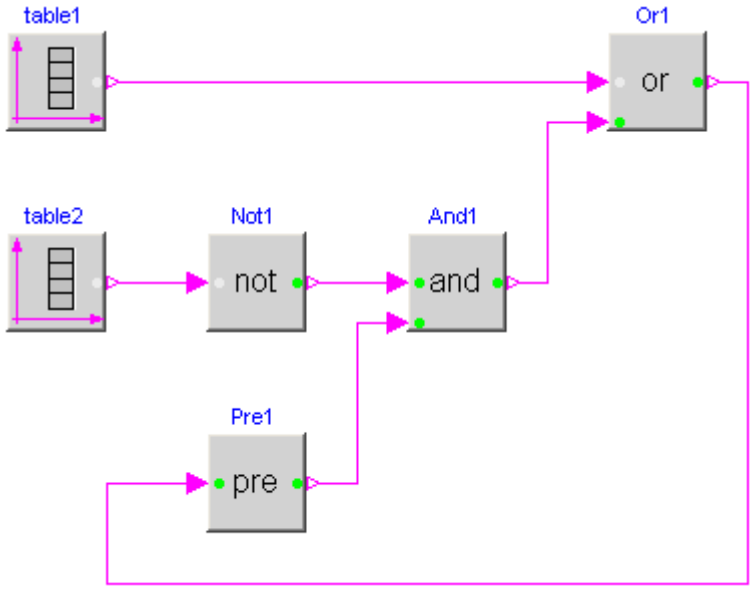
---

Modelica.Blocks.Logical

Library of components with Boolean input and output signals

Information

This package provides blocks with Boolean input and output signals to describe logical networks. A typical example for a logical network built with package Logical is shown in the next figure:



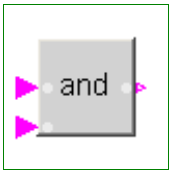
The actual value of Boolean input and/or output signals is displayed in the respective block icon as "circle", where "white" color means value **false** and "green" color means value **true**. These values are visualized in a diagram animation.

Package Content

Name	Description
And	Logical 'and': y = u1 and u2
Or	Logical 'or': y = u1 or u2
Xor	Logical 'xor': y = u1 xor u2
Nor	Logical 'nor': y = not (u1 or u2)
Nand	Logical 'nand': y = not (u1 and u2)
Not	Logical 'not': y = not u
Pre	Breaks algebraic loops by an infinitesimal small time delay (y = pre(u): event iteration continues until u = pre(u))
Edge	Output y is true, if the input u has a rising edge (y = edge(u))
FallingEdge	Output y is true, if the input u has a falling edge (y = edge(not u))
Change	Output y is true, if the input u has a rising or falling edge (y = change(u))
GreaterThreshold	Output y is true, if input u is greater than threshold
GreaterEqualThreshold	Output y is true, if input u is greater or equal than threshold
LessThreshold	Output y is true, if input u is less than threshold
LessEqualThreshold	Output y is true, if input u is less or equal than threshold
Greater	Output y is true, if input u1 is greater as input u2
GreaterEqual	Output y is true, if input u1 is greater or equal as input u2
Less	Output y is true, if input u1 is less as input u2
LessEqual	Output y is true, if input u1 is less or equal as input u2

ZeroCrossing	Trigger zero crossing of input u
LogicalSwitch	Logical Switch
Switch	Switch between two Real signals
Hysteresis	Transform Real to Boolean signal with Hysteresis
OnOffController	On-off controller
TriggeredTrapezoid	Triggered trapezoid generator
Timer	Timer measuring the time from the time instant where the Boolean input became true
TerminateSimulation	Terminate simulation if condition is fullfilled

Modelica.Blocks.Logical.And



Logical 'and': y = u1 and u2

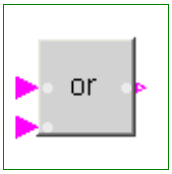
Information

The output is **true** if all inputs are **true**, otherwise the output is **false**.

Connectors

Type	Name	Description
input BooleanInput	u1	Connector of first Boolean input signal
input BooleanInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Or



Logical 'or': y = u1 or u2

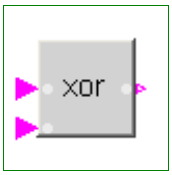
Information

The output is **true** if at least one input is **true**, otherwise the output is **false**.

Connectors

Type	Name	Description
input BooleanInput	u1	Connector of first Boolean input signal
input BooleanInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Xor



Logical 'xor': y = u1 xor u2

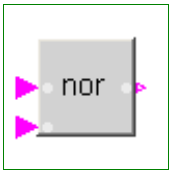
Information

The output is **true** if exactly one input is **true**, otherwise the output is **false**.

Connectors

Type	Name	Description
input BooleanInput	u1	Connector of first Boolean input signal
input BooleanInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Nor



Logical 'nor': y = not (u1 or u2)

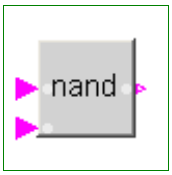
Information

The output is **true** if none of the inputs is **true**, otherwise the output is **false**.

Connectors

Type	Name	Description
input BooleanInput	u1	Connector of first Boolean input signal
input BooleanInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Nand



Logical 'nand': y = not (u1 and u2)

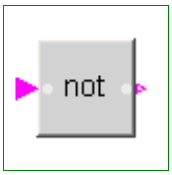
Information

The output is **true** if at least one input is **false**, otherwise the output is **false**.

Connectors

Type	Name	Description
input BooleanInput	u1	Connector of first Boolean input signal
input BooleanInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Not



Logical 'not': y = not u

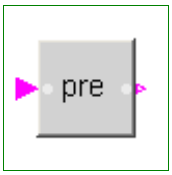
Information

The output is **true** if the input is **false**, otherwise the output is **false**.

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Pre



Breaks algebraic loops by an infinitesimal small time delay (y = pre(u): event iteration continues until u = pre(u))

Information

This block delays the Boolean input by an infinitesimal small time delay and therefore breaks algebraic loops. In a network of logical blocks, in every "closed connection loop" at least one logical block must have a delay, since algebraic systems of Boolean equations are not solveable.

The "Pre" block returns the value of the "input" signal from the last "event iteration". The "event iteration" stops, once both values are identical (u = pre(u)).

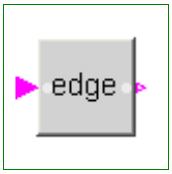
Parameters

Type	Name	Default	Description
Boolean	pre_u_start	false	Start value of pre(u) at initial time

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Edge



Output y is true, if the input u has a rising edge (y = edge(u))

Information

The output is **true** if the Boolean input has a rising edge from **false** to **true**, otherwise the output is **false**.

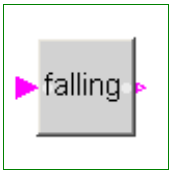
Parameters

Type	Name	Default	Description
Boolean	pre_u_start	false	Start value of pre(u) at initial time

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.FallingEdge



Output y is true, if the input u has a falling edge (y = edge(not u))

Information

The output is **true** if the Boolean input has a falling edge from **true** to **false**, otherwise the output is **false**.

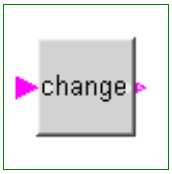
Parameters

Type	Name	Default	Description
Boolean	pre_u_start	false	Start value of pre(u) at initial time

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Change



Output y is true, if the input u has a rising or falling edge (y = change(u))

Information

The output is **true** if the Boolean input has either a rising edge from **false** to **true** or a falling edge from **true** to **false**, otherwise the output is **false**.

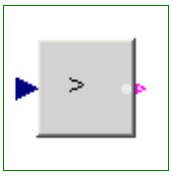
Parameters

Type	Name	Default	Description
Boolean	pre_u_start	false	Start value of pre(u) at initial time

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.GreaterThreshold



Output y is true, if input u is greater than threshold

Information

The output is **true** if the Real input is greater than parameter **threshold**, otherwise the output is **false**.

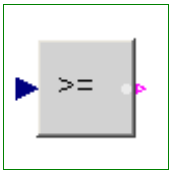
Parameters

Type	Name	Default	Description
Real	threshold	0	Comparison with respect to threshold

Connectors

Type	Name	Description
input RealInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.GreaterEqualThreshold



Output y is true, if input u is greater or equal than threshold

Information

The output is **true** if the Real input is greater than or equal to parameter **threshold**, otherwise the output is **false**.

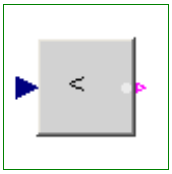
Parameters

Type	Name	Default	Description
Real	threshold	0	Comparison with respect to threshold

Connectors

Type	Name	Description
input RealInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.LessThreshold



Output y is true, if input u is less than threshold

Information

The output is **true** if the Real input is less than parameter **threshold**, otherwise the output is **false**.

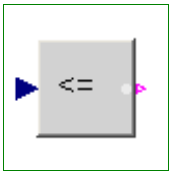
Parameters

Type	Name	Default	Description
Real	threshold	0	Comparison with respect to threshold

Connectors

Type	Name	Description
input RealInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.LessEqualThreshold



Output y is true, if input u is less or equal than threshold

Information

The output is **true** if the Real input is less than or equal to parameter **threshold**, otherwise the output is **false**.

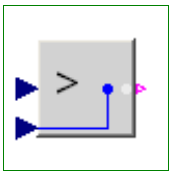
Parameters

Type	Name	Default	Description
Real	threshold	0	Comparison with respect to threshold

Connectors

Type	Name	Description
input RealInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Greater



Output y is true, if input u1 is greater as input u2

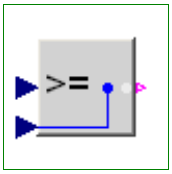
Information

The output is **true** if Real input u1 is greater than Real input u2, otherwise the output is **false**.

Connectors

Type	Name	Description
input RealInput	u1	Connector of first Boolean input signal
input RealInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.GreaterEqual



Output y is true, if input u1 is greater or equal as input u2

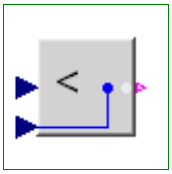
Information

The output is **true** if Real input u1 is greater than or equal to Real input u2, otherwise the output is **false**.

Connectors

Type	Name	Description
input RealInput	u1	Connector of first Boolean input signal
input RealInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Less



Output y is true, if input u1 is less as input u2

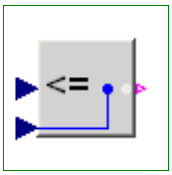
Information

The output is **true** if Real input u1 is less than Real input u2, otherwise the output is **false**.

Connectors

Type	Name	Description
input RealInput	u1	Connector of first Boolean input signal
input RealInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.LessEqual



Output y is true, if input u1 is less or equal as input u2

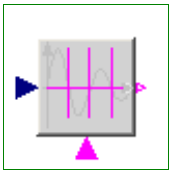
Information

The output is **true** if Real input u1 is less than or equal to Real input u2, otherwise the output is **false**.

Connectors

Type	Name	Description
input RealInput	u1	Connector of first Boolean input signal
input RealInput	u2	Connector of second Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.ZeroCrossing



Trigger zero crossing of input u

Information

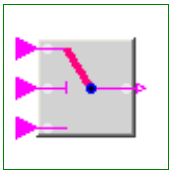
The output "y" is **true** at the time instant when the input "u" becomes zero, provided the input "enable" is **true**. At all other time instants, the output "y" is **false**. If the input "u" is zero at a time instant when the "enable" input changes its value, then the output y is **false**.

Note, that in the plot window of a Modelica simulator, the output of this block is usually identically to **false**, because the output may only be **true** at an event instant, but not during continuous integration. In order to check that this component is actually working as expected, one should connect its output to, e.g., component *ModelicaAdditions.Blocks.Discrete.TriggeredSampler*.

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal
input RealInput	u	
input BooleanInput	enable	Zero input crossing is triggered if the enable input signal is true

Modelica.Blocks.Logical.LogicalSwitch



Logical Switch

Information

The LogicalSwitch switches, depending on the Boolean u2 connector (the middle connector), between the two possible input signals u1 (upper connector) and u3 (lower connector).

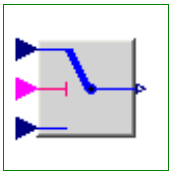
If u2 is true, connector y is set equal to u1, else it is set equal to u2.

Connectors

Type	Name	Description
input BooleanInput	u1	Connector of first Boolean input signal

input BooleanInput	u2	Connector of second Boolean input signal
input BooleanInput	u3	Connector of third Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Logical.Switch



Switch between two Real signals

Information

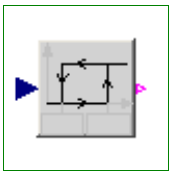
The Logical.Switch switches, depending on the logical connector u2 (the middle connector) between the two possible input signals u1 (upper connector) and u3 (lower connector).

If u2 is **true**, the output signal y is set equal to u1, else it is set equal to u3.

Connectors

Type	Name	Description
input RealInput	u1	Connector of first Real input signal
input BooleanInput	u2	Connector of Boolean input signal
input RealInput	u3	Connector of second Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Logical.Hysteresis



Transform Real to Boolean signal with Hysteresis

Information

This block transforms a **Real** input signal into a **Boolean** output signal:

- When the output was **false** and the input becomes **greater** than parameter **uHigh**, the output switches to **true**.
- When the output was **true** and the input becomes **less** than parameter **uLow**, the output switches to **false**.

The start value of the output is defined via parameter **pre_y_start** (= value of pre(y) at initial time). The default value of this parameter is **false**.

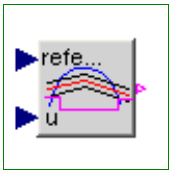
Parameters

Type	Name	Default	Description
Real	uLow	0	if y=true and u<=uLow, switch to y=false
Real	uHigh	1	if y=false and u>=uHigh, switch to y=true
Boolean	pre_y_start	false	Value of pre(y) at initial time

Connectors

Type	Name	Description
input RealInput	u	
output BooleanOutput	y	

Modelica.Blocks.Logical.OnOffController



On-off controller

Information

The block OnOffController sets the output signal **y** to **true** when the input signal **u** falls below the **reference** signal minus half of the bandwidth and sets the output signal **y** to **false** when the input signal **u** exceeds the **reference** signal plus half of the bandwidth.

Parameters

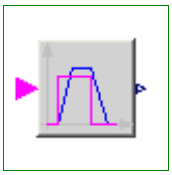
Type	Name	Default	Description
Real	bandwidth	0.1	Bandwidth around reference signal
Boolean	pre_y_start	false	Value of pre(y) at initial time

Connectors

Type	Name	Description
input RealInput	reference	Connector of Real input signal used as reference signal
input RealInput	u	Connector of Real input signal used as measurement signal
output BooleanOutput	y	Connector of Real output signal used as actuator signal

---

Modelica.Blocks.Logical.TriggeredTrapezoid



Triggered trapezoid generator

Information

The block TriggeredTrapezoid has a boolean input and a real output signal and requires the parameters *amplitude*, *rising*, *falling* and *offset*. The output signal **y** represents a trapezoidal signal dependent on the input signal **u**.

The behaviour is as follows: Assume the initial input to be false. In this case, the output will be *offset*. After a rising edge (i.e. the input changes from false to true), the output is rising during *rising* to the sum of *offset* and *amplitude*. In contrast, after a falling edge (i.e. the input changes from true to false), the output is falling during *falling* to a value of *offset*.

Note, that the case of edges before expiration of rising or falling is handled properly.

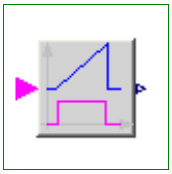
Parameters

Type	Name	Default	Description
Real	amplitude	1	Amplitude of trapezoid
Time	rising	0	Rising duration of trapezoid [s]
Time	falling	rising	Falling duration of trapezoid [s]
Real	offset	0	Offset of output signal

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Logical.Timer



Timer measuring the time from the time instant where the Boolean input became true

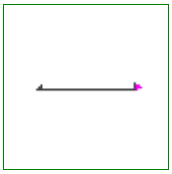
Information

When the Boolean input "u" becomes **true**, the timer is started and the output "y" is the time from the time instant where u became true. The timer is stopped and the output is reset to zero, once the input becomes false.

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Logical.TerminateSimulation



Terminate simulation if condition is fullfilled

Information

In the parameter menu, a **time varying** expression can be defined via variable **condition**, for example "condition = x < 0", where "x" is a variable that is declared in the model in which the "TerminateSimulation" block is present. If this expression becomes **true**, the simulation is (successfully) terminated. A termination message explaining the reason for the termination can be given via parameter "terminationText".

Parameters

Type	Name	Default	Description
BooleanOutput	condition	false	Terminate simulation when condition becomes true
String	terminationText	"... End condition reached"	Text that will be displayed when simulation is terminated

Connectors

Type	Name	Description
output BooleanOutput	condition	Terminate simulation when condition becomes true

---

Modelica.Blocks.Math

Library of mathematical functions as input/output blocks

Information

This package contains basic **mathematical operations**, such as summation and multiplication, and basic **mathematical functions**, such as **sqrt** and **sin**, as input/output blocks. All blocks of this library can be either connected with continuous blocks or with sampled-data blocks.

Package Content

Name	Description

UnitConversions	Conversion blocks to convert between SI and non-SI unit signals
TwoInputs	Change causality of input signals by defining that two input signals are identical (e.g. for inverse models)
TwoOutputs	Change causality of output signals by defining that two output signals are identical (e.g. for inverse models)
Gain	Output the product of a gain value with the input signal
MatrixGain	Output the product of a gain matrix with the input signal vector
Sum	Output the sum of the elements of the input vector
Feedback	Output difference between commanded and feedback input
Add	Output the sum of the two inputs
Add3	Output the sum of the three inputs
Product	Output product of the two inputs
Division	Output first input divided by second input
Abs	Output the absolute value of the input
Sign	Output the sign of the input
Sqrt	Output the square root of the input (input >= 0 required)
Sin	Output the sine of the input
Cos	Output the cosine of the input
Tan	Output the tangent of the input
Asin	Output the arc sine of the input
Acos	Output the arc cosine of the input
Atan	Output the arc tangent of the input
Atan2	Output atan(u1/u2) of the inputs u1 and u2
Sinh	Output the hyperbolic sine of the input
Cosh	Output the hyperbolic cosine of the input
Tanh	Output the hyperbolic tangent of the input
Exp	Output the exponential (base e) of the input
Log	Output the natural (base e) logarithm of the input (input > 0 required)
Log10	Output the base 10 logarithm of the input (input > 0 required)
RealToInteger	Convert Real to Integer signal
IntegerToReal	Convert integer to real signals
BooleanToReal	Convert Boolean to Real signal
BooleanToInteger	Convert Boolean to Integer signal
RealToBoolean	Convert Real to Boolean signal
IntegerToBoolean	Convert Integer to Boolean signal
Max	Pass through the largest signal
Min	Pass through the smallest signal
Edge	Indicates rising edge of boolean signal

BooleanChange	Indicates boolean signal changing
IntegerChange	Indicates integer signal changing

Modelica.Blocks.Math.UnitConversions

Conversion blocks to convert between SI and non-SI unit signals

Information

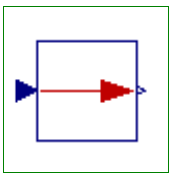
This package consists of blocks that convert an input signal with a specific unit to an output signal in another unit (e.g. conversion of an angle signal from "deg" to "rad"). Block "ConvertAllUnits" converts between a set of units that can be selected in a pull-down menu of the parameter menu. All other blocks convert exactly between two different units.

Package Content

Name	Description
ConvertAllUnits	Convert signal to a signal with different unit
To_degC	Convert from Kelvin to °Celsius
From_degC	Convert from °Celsius to Kelvin
To_degF	Convert from Kelvin to °Fahrenheit
From_degF	Convert from °Fahrenheit to Kelvin
To_degRk	Convert from Kelvin to °Rankine
From_degRk	Convert from °Rankine to Kelvin
To_deg	Convert from radian to degree
From_deg	Convert from degree to radian
To_rpm	Convert from radian per second to revolutions per minute
From_rpm	Convert from revolutions per minute to radian per second
To_kmh	Convert from metre per second to kilometre per hour
From_kmh	Convert from kilometre per hour to metre per second
To_day	Convert from second to day
From_day	Convert from day to second
To_hour	Convert from second to hour
From_hour	Convert from hour to second
To_minute	Convert from second to minute
From_minute	Convert from minute to second
To_litre	Convert from cubic metre to litre
From_litre	Convert from litre to cubic metre
To_kWh	Convert from Joule to kilo Watt hour
From_kWh	Convert from kilo Watt hour to Joule
To_bar	Convert from Pascal to bar

From_bar	Convert from bar to Pascal
To_gps	Convert from kilogram per second to gram per second
From_gps	Convert from gram per second to kilogram per second

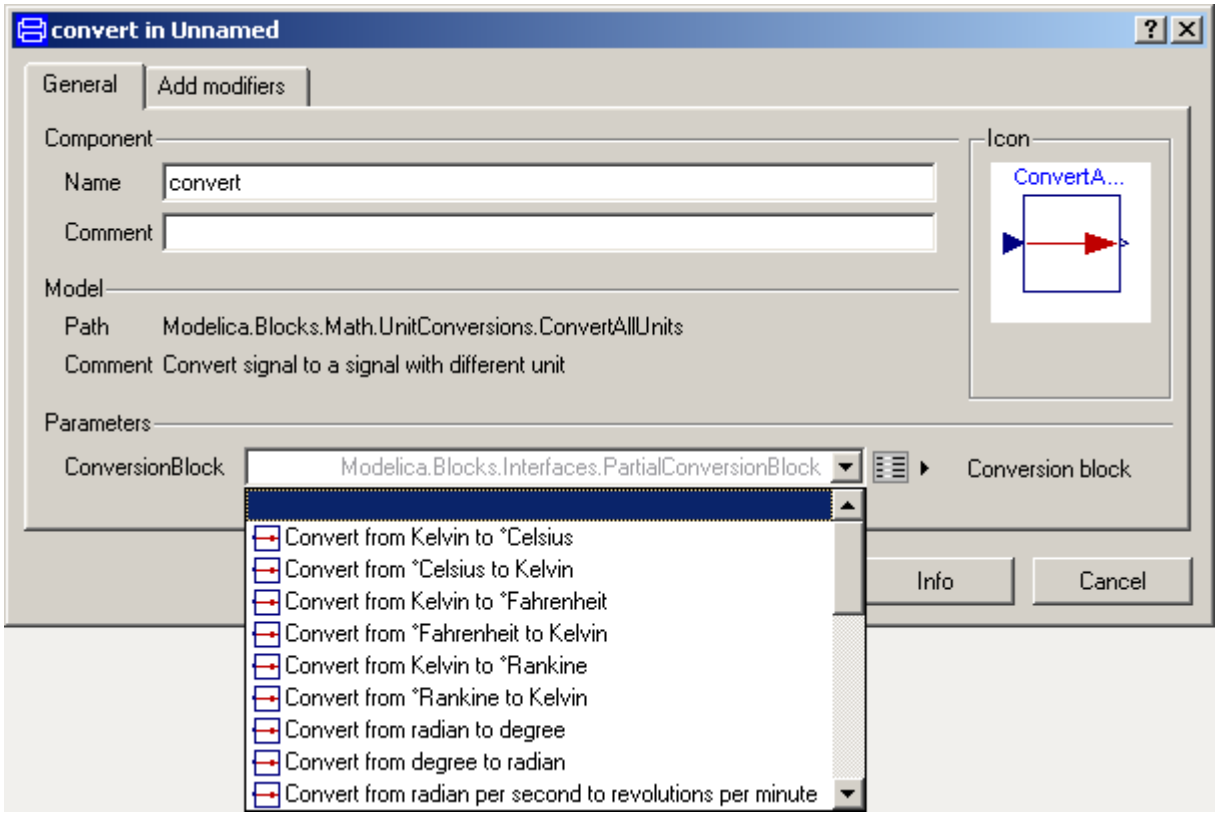
Modelica.Blocks.Math.UnitConversions.ConvertAllUnits



Convert signal to a signal with different unit

Information

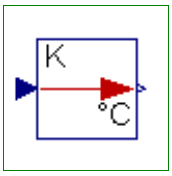
This block implements the Modelica.SIunits.Conversions functions as a fixed causality block to simplify their use. The block contains a replaceable block class **ConversionBlock** that can be changed to be any of the blocks defined in Modelica.Blocks.Math.UnitConversions, and more generally, any blocks that extend from Modelica.Blocks.Interfaces.PartialConversionBlock.



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal to be converted
output RealOutput	y	Connector of Real output signal containing input signal u in another unit

Modelica.Blocks.Math.UnitConversions.To_degC

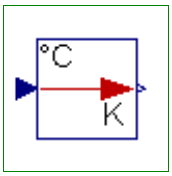


Convert from Kelvin to °Celsius

Information

This block converts the input signal from Kelvin to °Celsius and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_degC

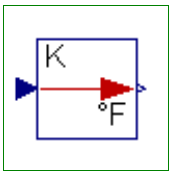


Convert from °Celsius to Kelvin

Information

This block converts the input signal from °Celsius to Kelvin and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_degF

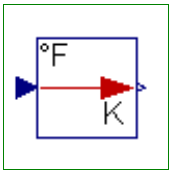


Convert from Kelvin to °Fahrenheit

Information

This block converts the input signal from Kelvin to °Fahrenheit and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_degF

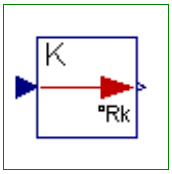


Convert from °Fahrenheit to Kelvin

Information

This block converts the input signal from °Fahrenheit to Kelvin and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_degRk

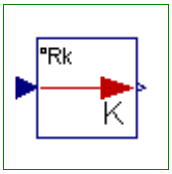


Convert from Kelvin to °Rankine

Information

This block converts the input signal from Kelvin to °Rankine and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_degRk

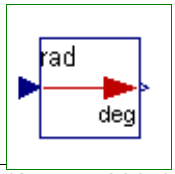


Convert from °Rankine to Kelvin

Information

This block converts the input signal from °Rankine to Kelvin and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_deg

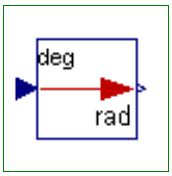


Convert from radian to degree

Information

This block converts the input signal from radian to degree and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_deg

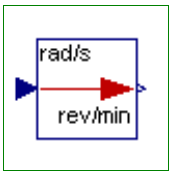


Convert from degree to radian

Information

This block converts the input signal from degree to radian and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_rpm

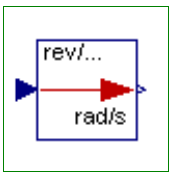


Convert from radian per second to revolutions per minute

Information

This block converts the input signal from radian per second to revolutions per minute and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_rpm

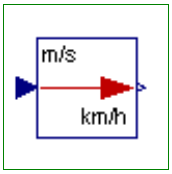


Convert from revolutions per minute to radian per second

Information

This block converts the input signal from revolutions per minute to radian per second and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_kmh

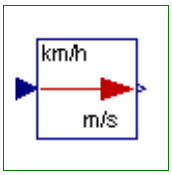


Convert from metre per second to kilometre per hour

Information

This block converts the input signal from metre per second to kilometre per hour and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_kmh

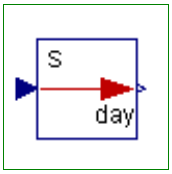


Convert from kilometre per hour to metre per second

Information

This block converts the input signal from kilometre per hour to metre per second and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_day

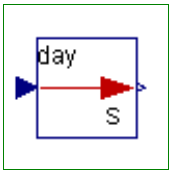


Convert from second to day

Information

This block converts the input signal from second to day and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_day

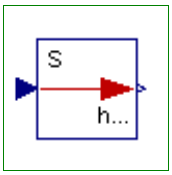


Convert from day to second

Information

This block converts the input signal from day to second and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_hour

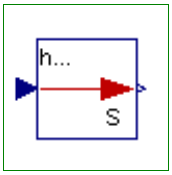


Convert from second to hour

Information

This block converts the input signal from second to hour and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_hour

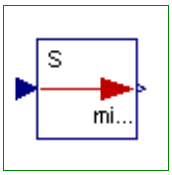


Convert from hour to second

Information

This block converts the input signal from hour to second and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_minute

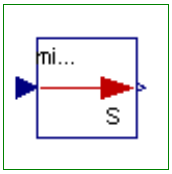


Convert from second to minute

Information

This block converts the input signal from second to minute and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_minute

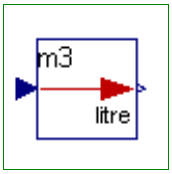


Convert from minute to second

Information

This block converts the input signal from minute to second and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_litre

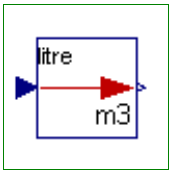


Convert from cubic metre to litre

Information

This block converts the input signal from metre to litre and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_litre

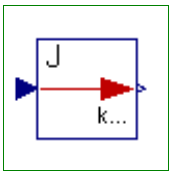


Convert from litre to cubic metre

Information

This block converts the input signal from litre to cubic metre and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_kWh

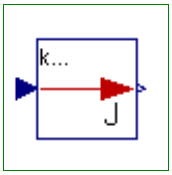


Convert from Joule to kilo Watt hour

Information

This block converts the input signal from Joule to kilo Watt hour and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_kWh

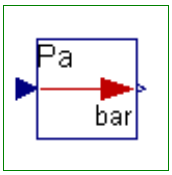


Convert from kilo Watt hour to Joule

Information

This block converts the input signal from kilo Watt hour to Joule and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_bar

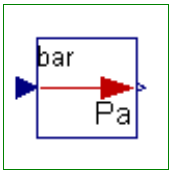


Convert from Pascal to bar

Information

This block converts the input signal from Pascal to bar and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_bar

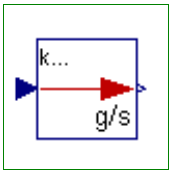


Convert from bar to Pascal

Information

This block converts the input signal from bar to Pascal and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.To_gps

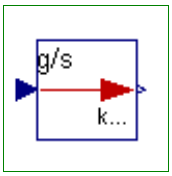


Convert from kilogram per second to gram per second

Information

This block converts the input signal from kilogram per second to gram per seconds and returns the result as output signal.

Modelica.Blocks.Math.UnitConversions.From_gps

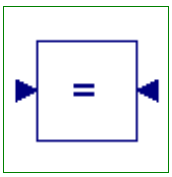


Convert from gram per second to kilogram per second

Information

This block converts the input signal from gram per second to kilogram per second and returns the result as output signal.

Modelica.Blocks.Math.TwoInputs



Change causality of input signals by defining that two input signals are identical (e.g. for inverse models)

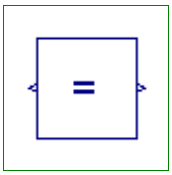
Information

This block is used to enable asignment of values to variables preliminary defined as outputs (e.g. useful for inverse model generation).

Connectors

Type	Name	Description
input RealInput	u1	Connector of first Real input signal
input RealInput	u2	Connector of second Real input signal (u1=u2)

Modelica.Blocks.Math.TwoOutputs



Change causality of output signals by defining that two output signals are identical (e.g. for inverse models)

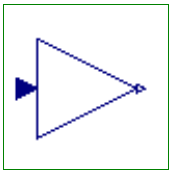
Information

This block is used to enable calculation of values preliminary defined as inputs. (e.g. useful for inverse model generation).

Connectors

Type	Name	Description
output RealOutput	y1	Connector of first Real output signal
output RealOutput	y2	Connector of second Real output signal (y1=y2)

Modelica.Blocks.Math.Gain



Output the product of a gain value with the input signal

Information

This block computes output *y* as *product* of gain *k* with the input *u*:

```
y = k * u;
```

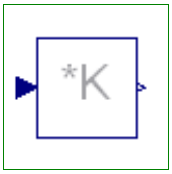
Parameters

Type	Name	Default	Description
Real	k	1	Gain value multiplied with input signal

Connectors

Type	Name	Description
input RealInput	u	Input signal connector
output RealOutput	y	Output signal connector

Modelica.Blocks.Math.MatrixGain



Output the product of a gain matrix with the input signal vector

Information

This blocks computes output vector **y** as *product* of the gain matrix **K** with the input signal vector **u**:

```
y = K * u;
```

Example:

```
parameter: K = [0.12 2; 3 1.5]

results in the following equations:

  | y[1] |     | 0.12  2.00 |   | u[1] |
  |      |  =  |            | * |      |
  | y[2] |     | 3.00  1.50 |   | u[2] |
```

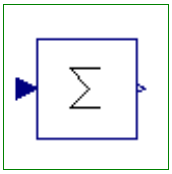
Parameters

Type	Name	Default	Description
Real	K[:, :]	[1, 0; 0, 1]	Gain matrix which is multiplied with the input

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y[nout]	Connector of Real output signals

Modelica.Blocks.Math.Sum



Output the sum of the elements of the input vector

Information

This blocks computes output **y** as *sum* of the elements of the input signal vector **u**:

```
y = u[1] + u[2] + ...;
```

Example:

```
parameter:   nin = 3;

results in the following equations:

y = u[1] + u[2] + u[3];
```

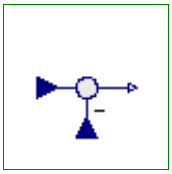
Parameters

Type	Name	Default	Description
Integer	nin	1	Number of inputs
Real	k[nin]	ones(nin)	Optional: sum coefficients

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Feedback



Output difference between commanded and feedback input

Information

This blocks computes output **y** as *difference* of the commanded input **u1** and the feedback input **u2**:

```
y = u1 - u2;
```

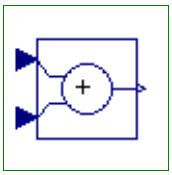
Example:

```
parameter:   n = 2

results in the following equations:

y = u1 - u2
```

Connectors

Type	Name	Description
input RealInput	u1	
input RealInput	u2	
output RealOutput	y	

Modelica.Blocks.Math.Add



Output the sum of the two inputs

Information

This blocks computes output **y** as *sum* of the two input signals **u1** and **u2**:

```
y = k1*u1 + k2*u2;
```

Example:

```
parameter:   k1= +2, k2= -3

results in the following equations:

y = 2 * u1 - 3 * u2
```

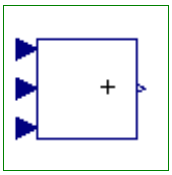
Parameters

Type	Name	Default	Description
Real	k1	+1	Gain of upper input
Real	k2	+1	Gain of lower input

Connectors

Type	Name	Description
input RealInput	u1	Connector of Real input signal 1
input RealInput	u2	Connector of Real input signal 2
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Add3



Output the sum of the three inputs

Information

This blocks computes output **y** as *sum* of the three input signals **u1**, **u2** and **u3**:

```
y = k1*u1 + k2*u2 + k3*u3;
```

Example:

```
parameter:   k1= +2, k2= -3, k3=1;

results in the following equations:

y = 2 * u1 - 3 * u2 + u3;
```

Parameters

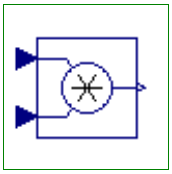
Type	Name	Default	Description
Real	k1	+1	Gain of upper input

Real	k2	+1	Gain of middle input
Real	k3	+1	Gain of lower input

Connectors

Type	Name	Description
input RealInput	u1	Connector 1 of Real input signals
input RealInput	u2	Connector 2 of Real input signals
input RealInput	u3	Connector 3 of Real input signals
output RealOutput	y	Connector of Real output signals

Modelica.Blocks.Math.Product



Output product of the two inputs

Information

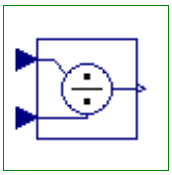
This blocks computes the output **y** (element-wise) as *product* of the corresponding elements of the two inputs **u1** and **u2**:

```
y = u1 * u2;
```

Connectors

Type	Name	Description
input RealInput	u1	Connector of Real input signal 1
input RealInput	u2	Connector of Real input signal 2
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Division



Output first input divided by second input

Information

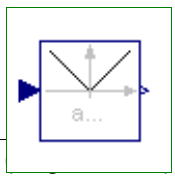
This block computes the output **y** (element-wise) by *dividing* the corresponding elements of the two inputs **u1** and **u2**:

```
y = u1 / u2;
```

Connectors

Type	Name	Description
input RealInput	u1	Connector of Real input signal 1
input RealInput	u2	Connector of Real input signal 2
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Abs



Output the absolute value of the input

Information

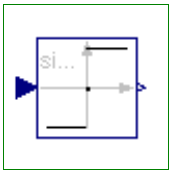
This blocks computes the output **y** as *absolute value* of the input **u**:

```
y = abs( u );
```

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Sign



Output the sign of the input

Information

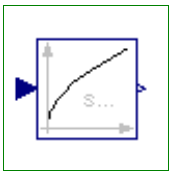
This blocks computes the output **y** as **sign** of the input **u**:

```
      1   if u > 0
y =   0   if u == 0
     -1   if u < 0
```

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Sqrt



Output the square root of the input (input >= 0 required)

Information

This blocks computes the output **y** as *square root* of the input **u**:

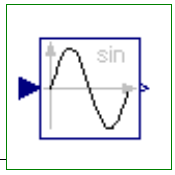
```
y = sqrt( u );
```

All elements of the input vector shall be zero or positive. Otherwise an error occurs.

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Sin

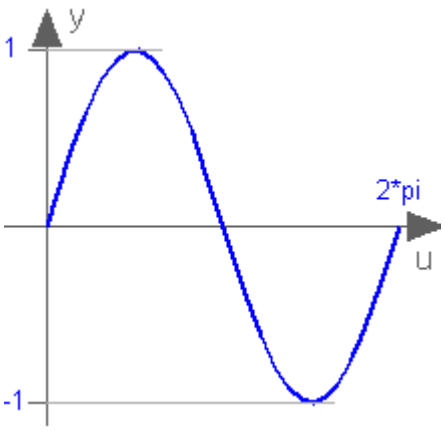


Output the sine of the input

Information

This blocks computes the output **y** as **sine** of the input **u**:

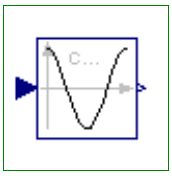
```
y = sin( u );
```



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Cos

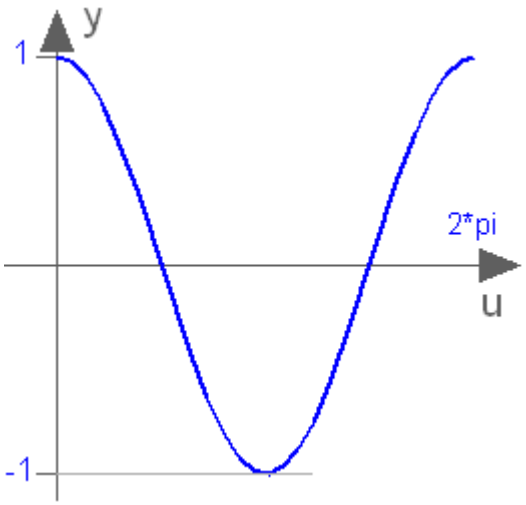


Output the cosine of the input

Information

This blocks computes the output **y** as **cos** of the input **u**:

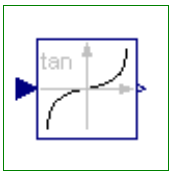
```
y = cos( u );
```



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Tan

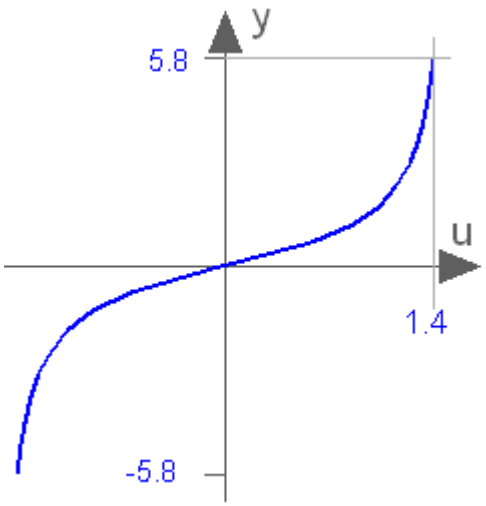


Output the tangent of the input

Information

This blocks computes the output **y** as **tan** of the input **u**:

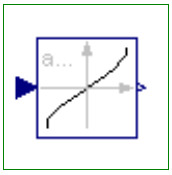
```
y = tan( u );
```



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Asin



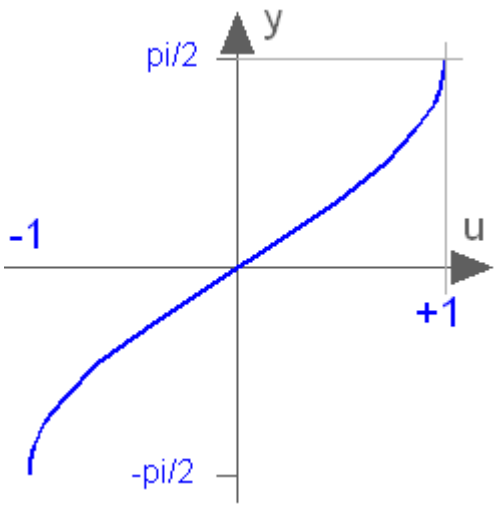
Output the arc sine of the input

Information

This blocks computes the output **y** as the *sine-inverse* of the input **u**:

```
y = asin( u );
```

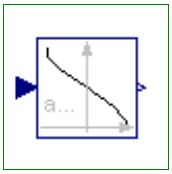
The absolute values of the elements of the input **u** need to be less or equal to one (**abs**(u) <= 1). Otherwise an error occurs.



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Acos



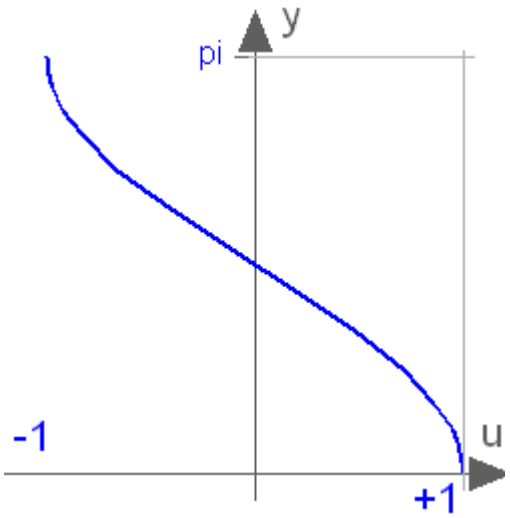
Output the arc cosine of the input

Information

This blocks computes the output **y** as the *cosine-inverse* of the input **u**:

```
y = acos( u );
```

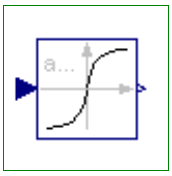
The absolute values of the elements of the input **u** need to be less or equal to one (**abs**(u) <= 1). Otherwise an error occurs.



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Atan

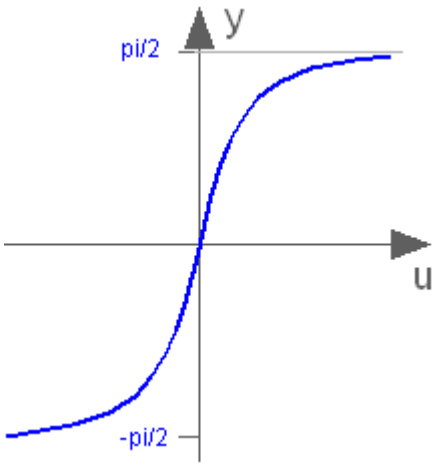


Output the arc tangent of the input

Information

This blocks computes the output **y** as the *tangent-inverse* of the input **u**:

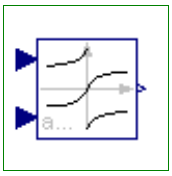
```
y= atan( u );
```



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Atan2



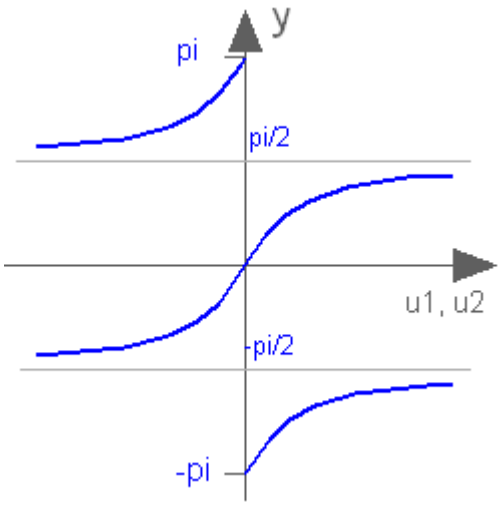
Output atan(u1/u2) of the inputs u1 and u2

Information

This blocks computes the output **y** as the *tangent-inverse* of the input **u1** divided by input **u2**:

```
y = atan2( u1, u2 );
```

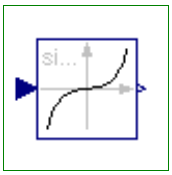
u1 and u2 shall not be zero at the same time instant. **Atan2** uses the sign of u1 and u2 in order to construct the solution in the range -180 deg ≤ y ≤ 180 deg, whereas block **Atan** gives a solution in the range -90 deg ≤ y ≤ 90 deg.



Connectors

Type	Name	Description
input RealInput	u1	Connector of Real input signal 1
input RealInput	u2	Connector of Real input signal 2
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Sinh

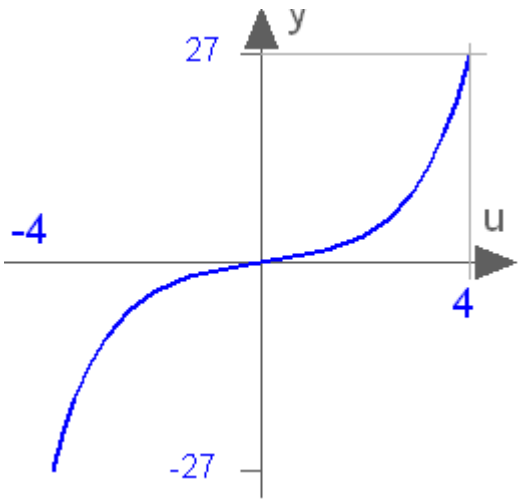


Output the hyperbolic sine of the input

Information

This blocks computes the output **y** as the *hyperbolic sine* of the input **u**:

```
y = sinh( u );
```

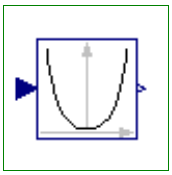


Connectors

Type	Name	Description

input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Cosh

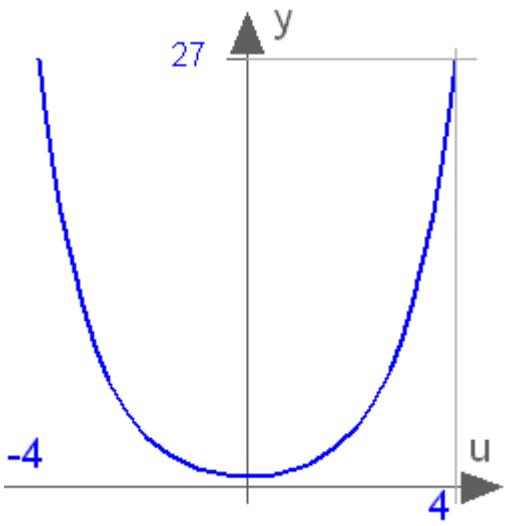


Output the hyperbolic cosine of the input

Information

This blocks computes the output **y** as the *hyperbolic cosine* of the input **u**:

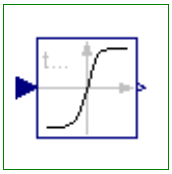
```
y = cosh( u );
```



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Tanh

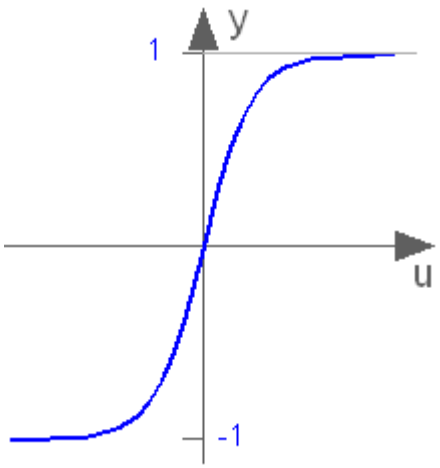


Output the hyperbolic tangent of the input

Information

This blocks computes the output **y** as the *hyperbolic tangent* of the input **u**:

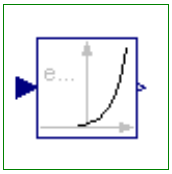
```
y = tanh( u );
```



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Exp

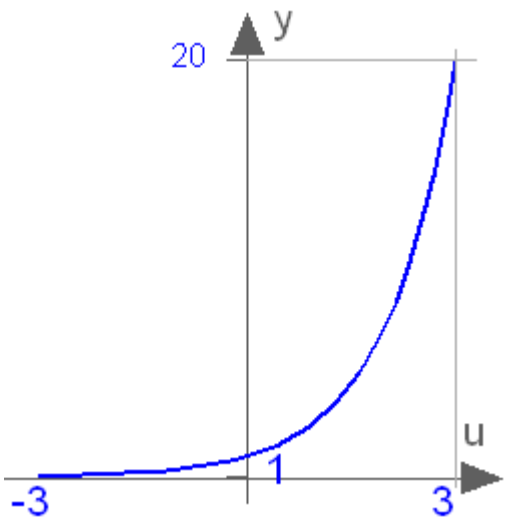


Output the exponential (base e) of the input

Information

This blocks computes the output **y** as the *exponential* (of base e) of the input **u**:

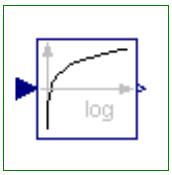
```
y = exp( u );
```



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Log



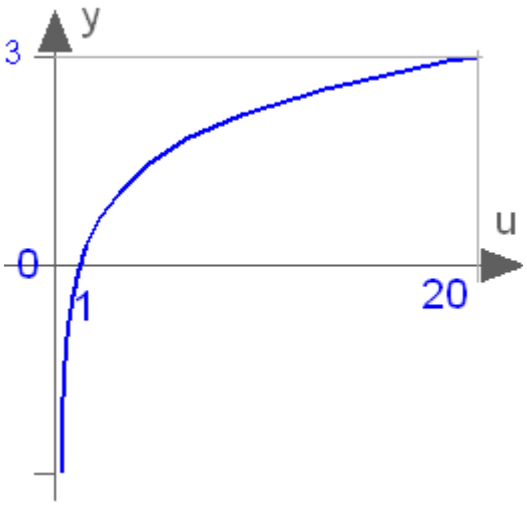
Output the natural (base e) logarithm of the input (input > 0 required)

Information

This blocks computes the output **y** as the *natural (base e) logarithm* of the input **u**:

```
y = log( u );
```

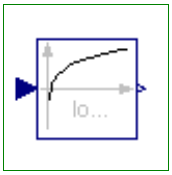
An error occurs if the elements of the input **u** are zero or negative.



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Log10



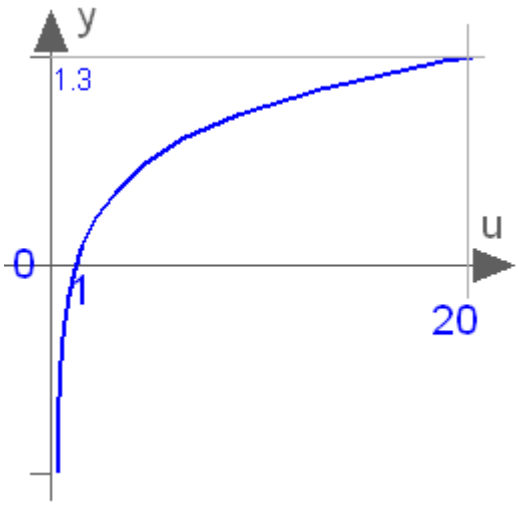
Output the base 10 logarithm of the input (input > 0 required)

Information

This blocks computes the output **y** as the *base 10 logarithm* of the input **u**:

```
y = log10( u );
```

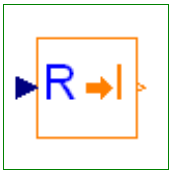
An error occurs if the elements of the input **u** are zero or negative.



Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.RealToInteger



Convert Real to Integer signal

Information

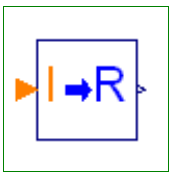
This block computes the output **y** as *nearest integer value* of the input **u**:

```
y = integer( floor( u + 0.5 ) )  for  u > 0;
y = integer( ceil ( u - 0.5 ) )  for  u < 0;
```

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output IntegerOutput	y	Connector of Integer output signal

Modelica.Blocks.Math.IntegerToReal



Convert integer to real signals

Information

This block computes the output **y** as *Real equivalent* of the Integer input **u**:

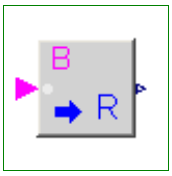
```
y = u;
```

where **u** is of Integer and **y** of Real type.

Connectors

Type	Name	Description
input IntegerInput	u	Connector of Integer input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.BooleanToReal



Convert Boolean to Real signal

Information

This block computes the output **y** as *Real equivalent* of the Boolean input **u**:

```
y = if u then realTrue else realFalse;
```

where **u** is of Boolean and **y** of Real type, and **realTrue** and **realFalse** are parameters.

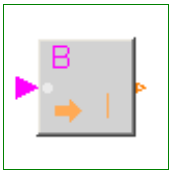
Parameters

Type	Name	Default	Description
Real	realTrue	1.0	Output signal for true Boolean input
Real	realFalse	0.0	Output signal for false Boolean input

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.BooleanToInteger



Convert Boolean to Integer signal

Information

This block computes the output **y** as *Integer equivalent* of the Boolean input **u**:

```
y = if u then integerTrue else integerFalse;
```

where **u** is of Boolean and **y** of Integer type, and **integerTrue** and **integerFalse** are parameters.

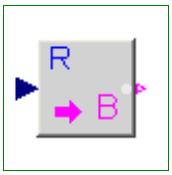
Parameters

Type	Name	Default	Description
Integer	integerTrue	1	Output signal for true Boolean input
Integer	integerFalse	0	Output signal for false Boolean input

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output IntegerOutput	y	Connector of Integer output signal

Modelica.Blocks.Math.RealToBoolean



Convert Real to Boolean signal

Information

This block computes the Boolean output **y** from the Real input **u** by the equation:

```
y = u ≥ threshold;
```

where **threshold** is a parameter.

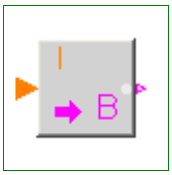
Parameters

Type	Name	Default	Description
Real	threshold	0.5	Output signal y is true, if input u >= threshold

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Math.IntegerToBoolean



Convert Integer to Boolean signal

Information

This block computes the Boolean output **y** from the Integer input **u** by the equation:

```
y = u ≥ threshold;
```

where **threshold** is a parameter.

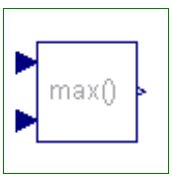
Parameters

Type	Name	Default	Description
Integer	threshold	1	Output signal y is true, if input u >= threshold

Connectors

Type	Name	Description
input IntegerInput	u	Connector of Integer input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Math.Max



Pass through the largest signal

Information

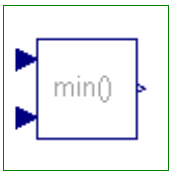
This block computes the output **y** as *maximum* of the two Real inputs **u1** and **u2**:

```
y = max ( u1 , u2 );
```

Connectors

Type	Name	Description
input RealInput	u1	Connector of Real input signal 1
input RealInput	u2	Connector of Real input signal 2
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Min



Pass through the smallest signal

Information

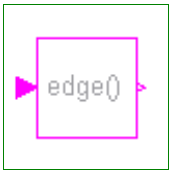
This block computes the output **y** as *minimum* of the two Real inputs **u1** and **u2**:

```
y = min ( u1 , u2 );
```

Connectors

Type	Name	Description
input RealInput	u1	Connector of Real input signal 1
input RealInput	u2	Connector of Real input signal 2
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Math.Edge



Indicates rising edge of boolean signal

Information

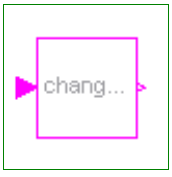
This block sets the Boolean output **y** to true, when the Boolean input **u** shows a *rising edge*:

```
y = edge( u );
```

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Math.BooleanChange



Indicates boolean signal changing

Information

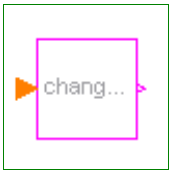
This block sets the Boolean output **y** to true, when the Boolean input **u** shows a *rising or falling edge*, i.e., when the signal changes:

```
y = change( u );
```

Connectors

Type	Name	Description
input BooleanInput	u	Connector of Boolean input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Math.IntegerChange



Indicates integer signal changing

Information

This block sets the Boolean output **y** to true, when the Integer input **u** changes:

```
y = change( u );
```

Connectors

Type	Name	Description
input IntegerInput	u	Connector of Integer input signal
output BooleanOutput	y	Connector of Boolean output signal

Modelica.Blocks.Nonlinear

Library of discontinuous or non-differentiable algebraic control blocks

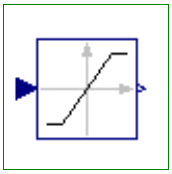
Information

This package contains **discontinuous** and **non-differentiable, algebraic** input/output blocks.

Package Content

Name	Description
Limiter	Limit the range of a signal
VariableLimiter	Limit the range of a signal with variable limits
DeadZone	Provide a region of zero output
FixedDelay	Delay block with fixed DelayTime
PadeDelay	Pade approximation of delay block with fixed DelayTime
VariableDelay	Delay block with variable DelayTime

Modelica.Blocks.Nonlinear.Limiter



Limit the range of a signal

Information

The Limiter block passes its input signal as output signal as long as the input is within the specified upper and lower limits. If this is not the case, the corresponding limits are passed as output.

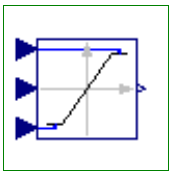
Parameters

Type	Name	Default	Description
Real	uMax	1	Upper limits of input signals
Real	uMin	-uMax	Lower limits of input signals
Boolean	limitsAtInit	true	= false, if limits are ignored during initializiation (i.e., y=u)

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Nonlinear.VariableLimiter



Limit the range of a signal with variable limits

Information

The Limiter block passes its input signal as output signal as long as the input is within the upper and lower limits specified by the two additional inputs limit1 and limit2. If this is not the case, the corresponding limit is passed as output.

Parameters

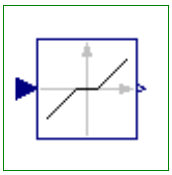
Type	Name	Default	Description
Boolean	limitsAtInit	true	= false, if limits are ignored during initializiation (i.e., y=u)

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal
input RealInput	limit1	Connector of Real input signal used as maximum of input u
input RealInput	limit2	Connector of Real input signal used as minimum of input u

---

Modelica.Blocks.Nonlinear.DeadZone



Provide a region of zero output

Information

The DeadZone block defines a region of zero output.

If the input is within uMin ... uMax, the output is zero. Outside of this zone, the output is a linear function of the input with a slope of 1.

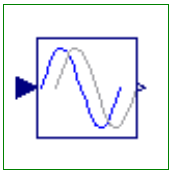
Parameters

Type	Name	Default	Description
Real	uMax	1	Upper limits of dead zones
Real	uMin	-uMax	Lower limits of dead zones
Boolean	deadZoneAtInit	true	= false, if dead zone is ignored during initializiation (i.e., y=u)

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Nonlinear.FixedDelay



Delay block with fixed DelayTime

Information

The Input signal is delayed by a given time instant, or more precisely:

```
y = u(time - delayTime)  for time > time.start + delayTime
  = u(time.start)        for time ≤ time.start + delayTime
```

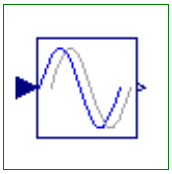
Parameters

Type	Name	Default	Description
Time	delayTime	1	Delay time of output with respect to input signal [s]

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Nonlinear.PadeDelay



Pade approximation of delay block with fixed DelayTime

Information

The Input signal is delayed by a given time instant, or more precisely:

```
y = u(time - delayTime)  for time > time.start + delayTime
  = u(time.start)        for time ≤ time.start + delayTime
```

The delay is approximated by a Pade approximation, i.e., by a transfer function

```
              b[1]*s^m + b[2]*s^[m-1] + ... + b[m+1]
y(s) = ---------------------------------------------- * u(s)
```

```
a[1]*s^n + a[2]*s^[n-1] + ... + a[n+1]
```

where the coefficients b[:] and a[:] are calculated such that the coefficients of the Taylor expansion of the delay exp(-T*s) around s=0 are identical upto order n+m.

The main advantage of this approach is that the delay is approximated by a linear differential equation system, which is continuous and continuously differentiable. For example, it is uncritical to linearize a system containing a Pade-approximated delay.

The standard text book version uses order "m=n", which is also the default setting of this block. The setting "m=n-1" may yield a better approximation in certain cases.

Literature:
Otto Foellinger: Regelungstechnik, 8. Auflage, chapter 11.9, page 412-414, Huethig Verlag Heidelberg, 1994

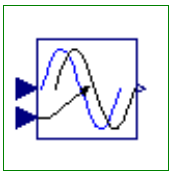
Parameters

Type	Name	Default	Description
Time	delayTime	1	Delay time of output with respect to input signal [s]
Integer	n	1	Order of pade approximation
Integer	m	n	Order of numerator

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Nonlinear.VariableDelay



Delay block with variable DelayTime

Information

The Input signal is delayed by a given time instant, or more precisely:

```
y = u(time - delayTime)  for time > time.start + delayTime
  = u(time.start)        for time ≤ time.start + delayTime
```

where delayTime is an additional input signal which must follow the following relationship:

```
0 ≤ delayTime ≤ delayMax
```

Parameters

Type	Name	Default	Description
Real	delayMax	1	maximum delay time

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal
input RealInput	delayTime	

Modelica.Blocks.Routing

Library of blocks to combine and extract signals

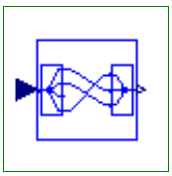
Information

This package contains blocks to combine and extract signals.

Package Content

Name	Description
ExtractSignal	Extract signals from an input signal vector
Extractor	Extract scalar signal out of signal vector dependent on IntegerRealInput index
Multiplex2	Multiplexer block for two input connectors
Multiplex3	Multiplexer block for three input connectors
Multiplex4	Multiplexer block for four input connectors
Multiplex5	Multiplexer block for five input connectors
Multiplex6	Multiplexer block for six input connectors
DeMultiplex2	DeMultiplexer block for two output connectors
DeMultiplex3	DeMultiplexer block for three output connectors
DeMultiplex4	DeMultiplexer block for four output connectors
DeMultiplex5	DeMultiplexer block for five output connectors
DeMultiplex6	DeMultiplexer block for six output connectors
RealPassThrough	Pass a Real signal through without modification
IntegerPassThrough	Pass a Integer signal through without modification
BooleanPassThrough	Pass a Boolean signal through without modification

Modelica.Blocks.Routing.ExtractSignal



Extract signals from an input signal vector

Information

Extract signals from the input connector and transfer them to the output connector.

The extracting scheme is given by the integer vector 'extract'. This vector specifies, which input signals are taken and in which order they are transfered to the output vector. Note, that the dimension of 'extract' has to match the number of outputs. Additionally, the dimensions of the input connector signals and the output connector signals have to be explicitly defined via the parameters 'nin' and 'nout'.

Example:

```
nin = 7 "Number of inputs";
nout = 4 "Number of outputs";
extract[nout] = {6,3,3,2} "Extracting vector";
```

extracts four output signals (nout=4) from the seven elements of the input vector (nin=7):

```
output no. 1 is set equal to input no. 6
output no. 2 is set equal to input no. 3
output no. 3 is set equal to input no. 3
output no. 4 is set equal to input no. 2
```

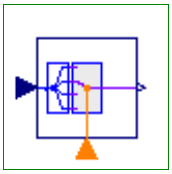
Parameters

Type	Name	Default	Description
Integer	nin	1	Number of inputs
Integer	nout	1	Number of outputs
Integer	extract[nout]	1:nout	Extracting vector

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y[nout]	Connector of Real output signals

Modelica.Blocks.Routing.Extractor



Extract scalar signal out of signal vector dependent on IntegerRealInput index

Information

This block extracts a scalar output signal out the vector of input signals dependent on the Integer value of the additional u index:

```
y = u [ index ] ;
```

where index is an additional Integer input signal.

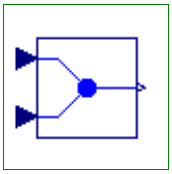
Parameters

Type	Name	Default	Description
Integer	nin	1	Number of inputs
Boolean	allowOutOfRange	false	Index may be out of range
Real	outOfRangeValue	1e10	Output signal if index is out of range

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y	Connector of Real output signal
input IntegerInput	index	

Modelica.Blocks.Routing.Multiplex2



Multiplexer block for two input connectors

Information

The output connector is the **concatenation** of the two input connectors. Note, that the dimensions of the input connector signals have to be explicitly defined via parameters n1 and n2.

Parameters

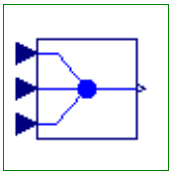
Type	Name	Default	Description
Integer	n1	1	dimension of input signal connector 1
Integer	n2	1	dimension of input signal connector 2

Connectors

Type	Name	Description
input RealInput	u1[n1]	Connector of Real input signals 1
input RealInput	u2[n2]	Connector of Real input signals 2
output RealOutput	y[n1 + n2]	Connector of Real output signals

---

Modelica.Blocks.Routing.Multiplex3



Multiplexer block for three input connectors

Information

The output connector is the **concatenation** of the three input connectors. Note, that the dimensions of the input connector signals have to be explicitly defined via parameters n1, n2 and n3.

Parameters

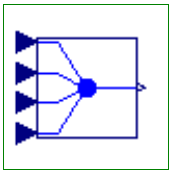
Type	Name	Default	Description
Integer	n1	1	dimension of input signal connector 1
Integer	n2	1	dimension of input signal connector 2
Integer	n3	1	dimension of input signal connector 3

Connectors

Type	Name	Description
input RealInput	u1[n1]	Connector of Real input signals 1
input RealInput	u2[n2]	Connector of Real input signals 2
input RealInput	u3[n3]	Connector of Real input signals 3
output RealOutput	y[n1 + n2 + n3]	Connector of Real output signals

---

Modelica.Blocks.Routing.Multiplex4



Multiplexer block for four input connectors

Information

The output connector is the **concatenation** of the four input connectors. Note, that the dimensions of the input connector signals have to be explicitly defined via parameters n1, n2, n3 and n4.

Parameters

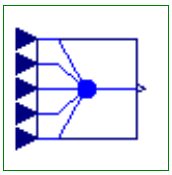
Type	Name	Default	Description
Integer	n1	1	dimension of input signal connector 1
Integer	n2	1	dimension of input signal connector 2
Integer	n3	1	dimension of input signal connector 3
Integer	n4	1	dimension of input signal connector 4

Connectors

Type	Name	Description
input RealInput	u1[n1]	Connector of Real input signals 1
input RealInput	u2[n2]	Connector of Real input signals 2
input RealInput	u3[n3]	Connector of Real input signals 3
input RealInput	u4[n4]	Connector of Real input signals 4
output RealOutput	y[n1 + n2 + n3 + n4]	Connector of Real output signals

---

Modelica.Blocks.Routing.Multiplex5



Multiplexer block for five input connectors

Information

The output connector is the **concatenation** of the five input connectors. Note, that the dimensions of the input connector signals have to be explicitly defined via parameters n1, n2, n3, n4 and n5.

Parameters

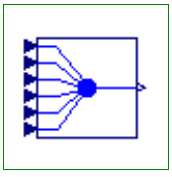
Type	Name	Default	Description
Integer	n1	1	dimension of input signal connector 1
Integer	n2	1	dimension of input signal connector 2
Integer	n3	1	dimension of input signal connector 3
Integer	n4	1	dimension of input signal connector 4
Integer	n5	1	dimension of input signal connector 5

Connectors

Type	Name	Description
input RealInput	u1[n1]	Connector of Real input signals 1
input RealInput	u2[n2]	Connector of Real input signals 2
input RealInput	u3[n3]	Connector of Real input signals 3
input RealInput	u4[n4]	Connector of Real input signals 4
input RealInput	u5[n5]	Connector of Real input signals 5
output RealOutput	y[n1 + n2 + n3 + n4 + n5]	Connector of Real output signals

---

Modelica.Blocks.Routing.Multiplex6



Multiplexer block for six input connectors

Information

The output connector is the **concatenation** of the six input connectors. Note, that the dimensions of the input connector signals have to be explicitly defined via parameters n1, n2, n3, n4, n5 and n6.

Parameters

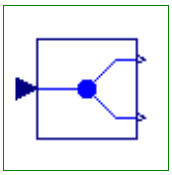
Type	Name	Default	Description
Integer	n1	1	dimension of input signal connector 1
Integer	n2	1	dimension of input signal connector 2
Integer	n3	1	dimension of input signal connector 3
Integer	n4	1	dimension of input signal connector 4
Integer	n5	1	dimension of input signal connector 5
Integer	n6	1	dimension of input signal connector 6

Connectors

Type	Name	Description
input RealInput	u1[n1]	Connector of Real input signals 1
input RealInput	u2[n2]	Connector of Real input signals 2
input RealInput	u3[n3]	Connector of Real input signals 3
input RealInput	u4[n4]	Connector of Real input signals 4
input RealInput	u5[n5]	Connector of Real input signals 5
input RealInput	u6[n6]	Connector of Real input signals 6
output RealOutput	y[n1 + n2 + n3 + n4 + n5 + n6]	Connector of Real output signals

---

Modelica.Blocks.Routing.DeMultiplex2



DeMultiplexer block for two output connectors

Information

The input connector is **splitted** up into two output connectors. Note, that the dimensions of the output connector signals have to be explicitly defined via parameters n1 and n2.

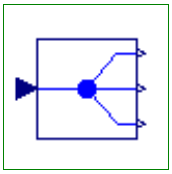
Parameters

Type	Name	Default	Description
Integer	n1	1	dimension of output signal connector 1
Integer	n2	1	dimension of output signal connector 2

Connectors

Type	Name	Description
input RealInput	u[n1 + n2]	Connector of Real input signals
output RealOutput	y1[n1]	Connector of Real output signals 1
output RealOutput	y2[n2]	Connector of Real output signals 2

Modelica.Blocks.Routing.DeMultiplex3



DeMultiplexer block for three output connectors

Information

The input connector is **splitted** up into three output connectors. Note, that the dimensions of the output connector signals have to be explicitly defined via parameters n1, n2 and n3.

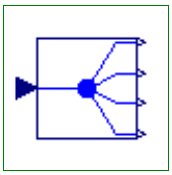
Parameters

Type	Name	Default	Description
Integer	n1	1	dimension of output signal connector 1
Integer	n2	1	dimension of output signal connector 2
Integer	n3	1	dimension of output signal connector 3

Connectors

Type	Name	Description
input RealInput	u[n1 + n2 + n3]	Connector of Real input signals
output RealOutput	y1[n1]	Connector of Real output signals 1
output RealOutput	y2[n2]	Connector of Real output signals 2
output RealOutput	y3[n3]	Connector of Real output signals 3

Modelica.Blocks.Routing.DeMultiplex4



DeMultiplexer block for four output connectors

Information

The input connector is **splitted** up into four output connectors. Note, that the dimensions of the output connector signals have to be explicitly defined via parameters n1, n2, n3 and n4.

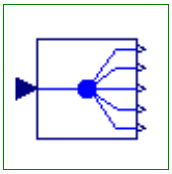
Parameters

Type	Name	Default	Description
Integer	n1	1	dimension of output signal connector 1
Integer	n2	1	dimension of output signal connector 2
Integer	n3	1	dimension of output signal connector 3
Integer	n4	1	dimension of output signal connector 4

Connectors

Type	Name	Description
input RealInput	u[n1 + n2 + n3 + n4]	Connector of Real input signals
output RealOutput	y1[n1]	Connector of Real output signals 1
output RealOutput	y2[n2]	Connector of Real output signals 2
output RealOutput	y3[n3]	Connector of Real output signals 3
output RealOutput	y4[n4]	Connector of Real output signals 4

Modelica.Blocks.Routing.DeMultiplex5



DeMultiplexer block for five output connectors

Information

The input connector is **splitted** up into five output connectors. Note, that the dimensions of the output connector signals have to be explicitly defined via parameters n1, n2, n3, n4 and n5.

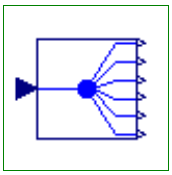
Parameters

Type	Name	Default	Description
Integer	n1	1	dimension of output signal connector 1
Integer	n2	1	dimension of output signal connector 2
Integer	n3	1	dimension of output signal connector 3
Integer	n4	1	dimension of output signal connector 4
Integer	n5	1	dimension of output signal connector 5

Connectors

Type	Name	Description
input RealInput	u[n1 + n2 + n3 + n4 + n5]	Connector of Real input signals
output RealOutput	y1[n1]	Connector of Real output signals 1
output RealOutput	y2[n2]	Connector of Real output signals 2
output RealOutput	y3[n3]	Connector of Real output signals 3
output RealOutput	y4[n4]	Connector of Real output signals 4
output RealOutput	y5[n5]	Connector of Real output signals 5

Modelica.Blocks.Routing.DeMultiplex6



DeMultiplexer block for six output connectors

Information

The input connector is **splitted** up into six output connectors. Note, that the dimensions of the output connector signals have to be explicitly defined via parameters n1, n2, n3, n4, n5 and n6.

Parameters

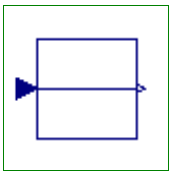
Type	Name	Default	Description
Integer	n1	1	dimension of output signal connector 1
Integer	n2	1	dimension of output signal connector 2
Integer	n3	1	dimension of output signal connector 3
Integer	n4	1	dimension of output signal connector 4
Integer	n5	1	dimension of output signal connector 5
Integer	n6	1	dimension of output signal connector 6

Connectors

Type	Name	Description
input RealInput	u[n1 + n2 + n3 + n4 + n5 + n6]	Connector of Real input signals

output RealOutput	y1[n1]	Connector of Real output signals 1
output RealOutput	y2[n2]	Connector of Real output signals 2
output RealOutput	y3[n3]	Connector of Real output signals 3
output RealOutput	y4[n4]	Connector of Real output signals 4
output RealOutput	y5[n5]	Connector of Real output signals 5
output RealOutput	y6[n6]	Connector of Real output signals 6

Modelica.Blocks.Routing.RealPassThrough



Pass a Real signal through without modification

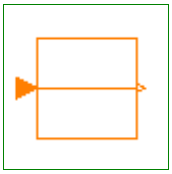
Information

Passes a Real signal through without modification. Enables signals to be read out of one bus, have their name changed and be sent back to a bus.

Connectors

Type	Name	Description
input RealInput	u	Input signal
output RealOutput	y	Output signal

Modelica.Blocks.Routing.IntegerPassThrough



Pass a Integer signal through without modification

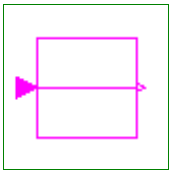
Information

Passes a Integer signal through without modification. Enables signals to be read out of one bus, have their name changed and be sent back to a bus.

Connectors

Type	Name	Description
input IntegerInput	u	Input signal
output IntegerOutput	y	Output signal

Modelica.Blocks.Routing.BooleanPassThrough



Pass a Boolean signal through without modification

Information

Passes a Boolean signal through without modification. Enables signals to be read out of one bus, have their name changed and be sent back to a bus.

Connectors

Type	Name	Description
input BooleanInput	u	Input signal

output BooleanOutput	y	Output signal

Modelica.Blocks.Sources

Library of signal source blocks generating Real and Boolean signals

Information

This package contains **source** components, i.e., blocks which have only output signals. These blocks are used as signal generators for Real, Integer and Boolean signals.

All Real source signals (with the exception of the Constant source) have at least the following two parameters:

offset	Value which is added to the signal
startTime	Start time of signal. For time < startTime, the output y is set to offset.

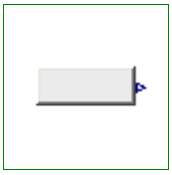
The **offset** parameter is especially useful in order to shift the corresponding source, such that at initial time the system is stationary. To determine the corresponding value of offset, usually requires a trimming calculation.

Package Content

Name	Description
RealExpression	Set output signal to a time varying Real expression
IntegerExpression	Set output signal to a time varying Integer expression
BooleanExpression	Set output signal to a time varying Boolean expression
Clock	Generate actual time signal
Constant	Generate constant signal of type Real
Step	Generate step signal of type Real
Ramp	Generate ramp signal
Sine	Generate sine signal
ExpSine	Generate exponentially damped sine signal
Exponentials	Generate a rising and falling exponential signal
Pulse	Generate pulse signal of type Real
SawTooth	Generate saw tooth signal
Trapezoid	Generate trapezoidal signal of type Real
KinematicPTP	Move as fast as possible along a distance within given kinematic constraints
KinematicPTP2	Move as fast as possible from start to end position within given kinematic constraints with output signals q, qd=der(q), qdd=der(qd)
TimeTable	Generate a (possibly discontinuous) signal by linear interpolation in a table
CombiTimeTable	Table look-up with respect to time and linear/perodic extrapolation methods (data from matrix/file)
BooleanConstant	Generate constant signal of type Boolean
BooleanStep	Generate step signal of type Boolean
BooleanPulse	Generate pulse signal of type Boolean

SampleTrigger	Generate sample trigger signal
BooleanTable	Generate a Boolean output signal based on a vector of time instants
IntegerConstant	Generate constant signal of type Integer
IntegerStep	Generate step signal of type Integer

Modelica.Blocks.Sources.RealExpression



Set output signal to a time varying Real expression

Information

The (time varying) Real output signal of this block can be defined in its parameter menu via variable **y**. The purpose is to support the easy definition of Real expressions in a block diagram. For example, in the y-menu the definition "if time < 1 then 0 else 1" can be given in order to define that the output signal is one, if time ≥ 1 and otherwise it is zero. Note, that "time" is a built-in variable that is always accessible and represents the "model time" and that Variable **y** is both a variable and a connector.

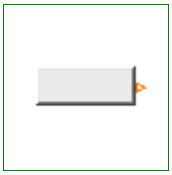
Parameters

Type	Name	Default	Description
Time varying output signal			
RealOutput	y	0.0	Value of Real output

Connectors

Type	Name	Description
Time varying output signal		
output RealOutput	y	Value of Real output

Modelica.Blocks.Sources.IntegerExpression



Set output signal to a time varying Integer expression

Information

The (time varying) Integer output signal of this block can be defined in its parameter menu via variable **y**. The purpose is to support the easy definition of Integer expressions in a block diagram. For example, in the y-menu the definition "if time < 1 then 0 else 1" can be given in order to define that the output signal is one, if time ≥ 1 and otherwise it is zero. Note, that "time" is a built-in variable that is always accessible and represents the "model time" and that Variable **y** is both a variable and a connector.

Parameters

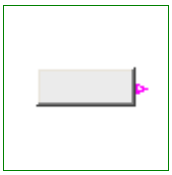
Type	Name	Default	Description
Time varying output signal			
IntegerOutput	y	0	Value of Integer output

Connectors

Type	Name	Description

Time varying output signal		
output IntegerOutput	y	Value of Integer output

Modelica.Blocks.Sources.BooleanExpression



Set output signal to a time varying Boolean expression

Information

The (time varying) Boolean output signal of this block can be defined in its parameter menu via variable **y**. The purpose is to support the easy definition of Boolean expressions in a block diagram. For example, in the y-menu the definition "time >= 1 and time <= 2" can be given in order to define that the output signal is **true** in the time interval 1 ≤ time ≤ 2 and otherwise it is **false**. Note, that "time" is a built-in variable that is always accessible and represents the "model time" and that Variable **y** is both a variable and a connector.

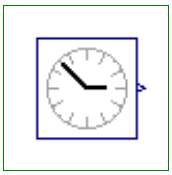
Parameters

Type	Name	Default	Description
Time varying output signal			
BooleanOutput	y	false	Value of Boolean output

Connectors

Type	Name	Description
Time varying output signal		
output BooleanOutput	y	Value of Boolean output

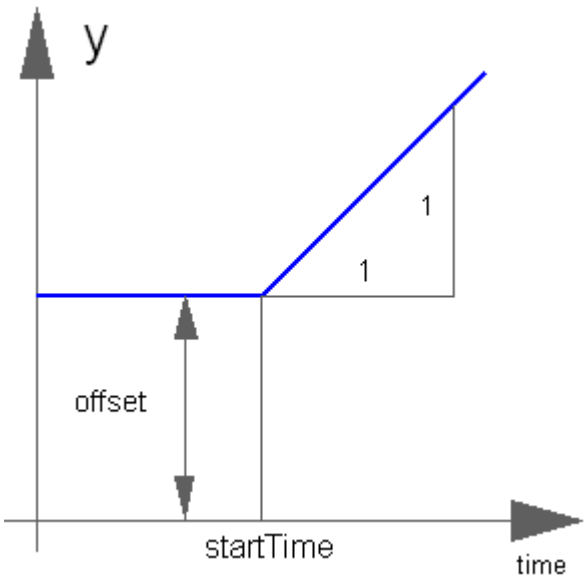
Modelica.Blocks.Sources.Clock



Generate actual time signal

Information

The Real output y is a clock signal:



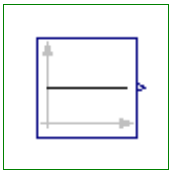
Parameters

Type	Name	Default	Description
Real	offset	0	Offset of output signal
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

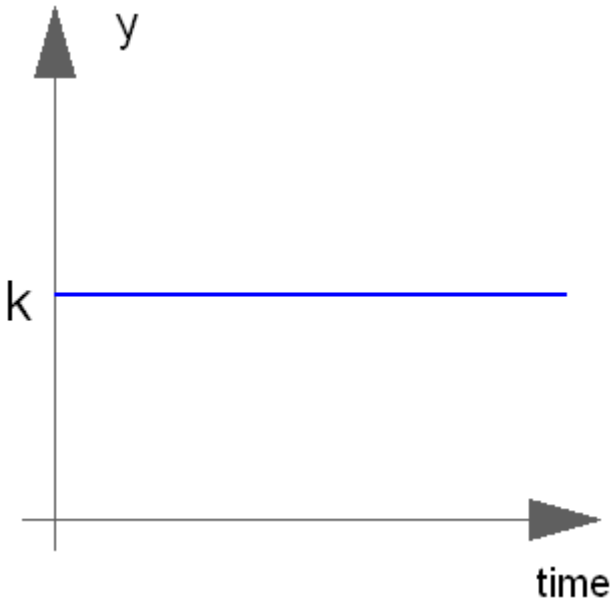
Modelica.Blocks.Sources.Constant



Generate constant signal of type Real

Information

The Real output y is a constant signal:



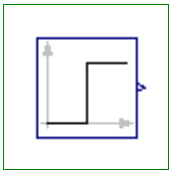
Parameters

Type	Name	Default	Description
Real	k	1	Constant output value

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

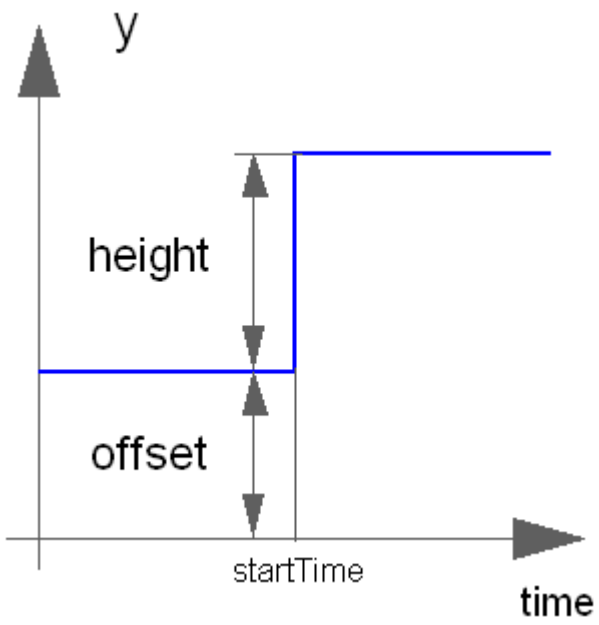
Modelica.Blocks.Sources.Step



Generate step signal of type Real

Information

The Real output y is a step signal:



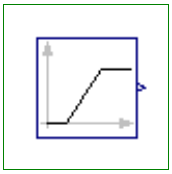
Parameters

Type	Name	Default	Description
Real	height	1	Height of step
Real	offset	0	Offset of output signal y
Time	startTime	0	Output y = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

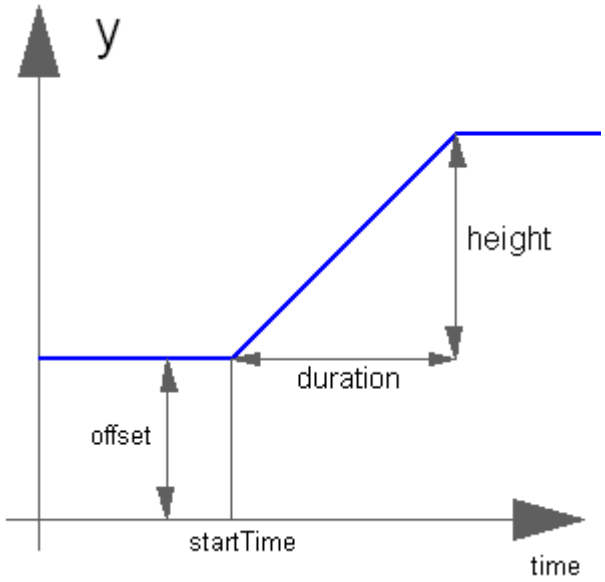
Modelica.Blocks.Sources.Ramp



Generate ramp signal

Information

The Real output y is a ramp signal:



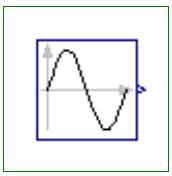
Parameters

Type	Name	Default	Description
Real	height	1	Height of ramps
Real	duration	2	Durations of ramp
Real	offset	0	Offset of output signal
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

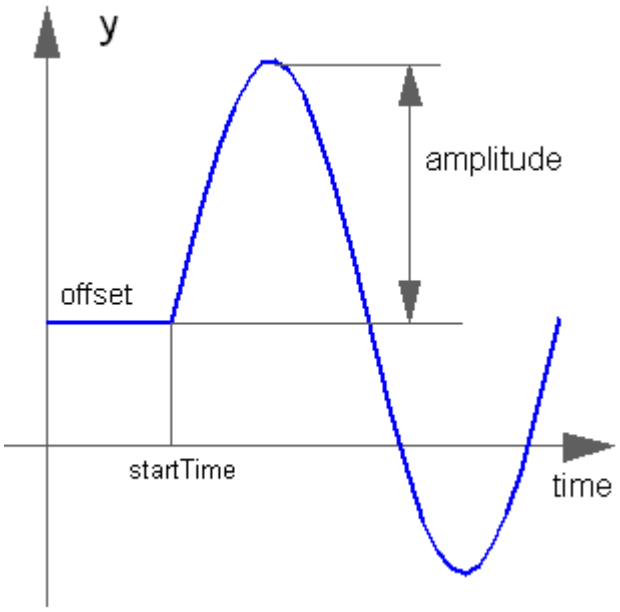
Modelica.Blocks.Sources.Sine



Generate sine signal

Information

The Real output y is a sine signal:



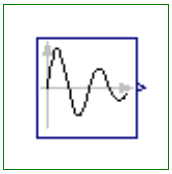
Parameters

Type	Name	Default	Description
Real	amplitude	1	Amplitude of sine wave
Frequency	freqHz	1	Frequency of sine wave [Hz]
Angle	phase	0	Phase of sine wave [rad]
Real	offset	0	Offset of output signal
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

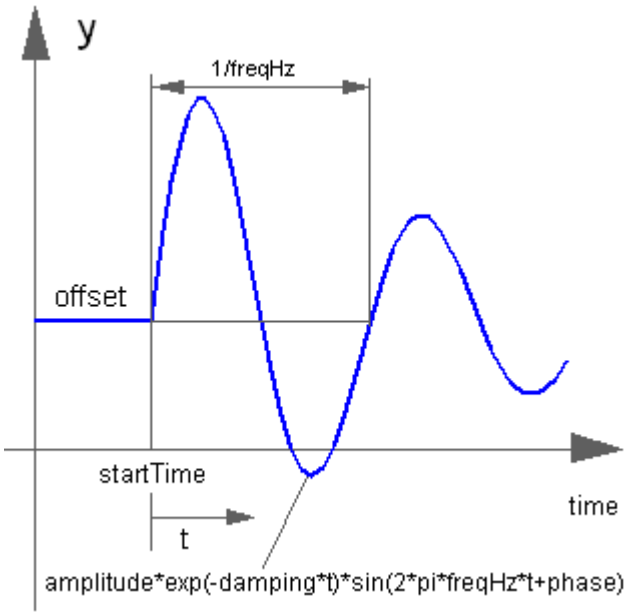
Modelica.Blocks.Sources.ExpSine



Generate exponentially damped sine signal

Information

The Real output y is a sine signal with exponentially changing amplitude:



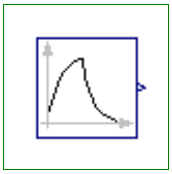
Parameters

Type	Name	Default	Description
Real	amplitude	1	Amplitude of sine wave
Frequency	freqHz	2	Frequency of sine wave [Hz]
Angle	phase	0	Phase of sine wave [rad]
Damping	damping	1	Damping coefficient of sine wave [s-1]
Real	offset	0	Offset of output signal
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

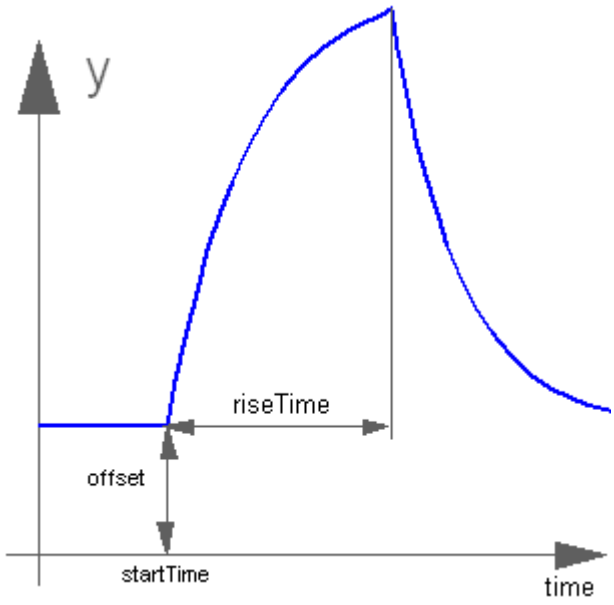
Modelica.Blocks.Sources.Exponentials



Generate a rising and falling exponential signal

Information

The Real output y is a rising exponential followed by a falling exponential signal:



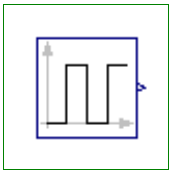
Parameters

Type	Name	Default	Description
Real	outMax	1	Height of output for infinite riseTime
Time	riseTime	0.5	Rise time [s]
Time	riseTimeConst	0.1	Rise time constant [s]
Time	fallTimeConst	riseTimeConst	Fall time constant [s]
Real	offset	0	Offset of output signal
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

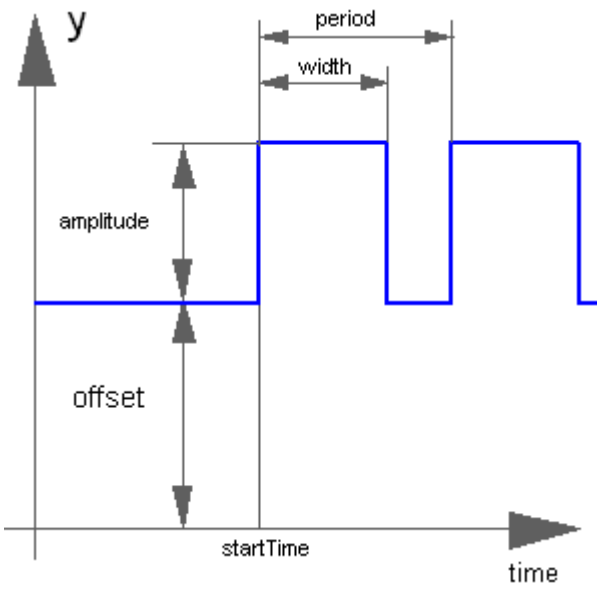
Modelica.Blocks.Sources.Pulse



Generate pulse signal of type Real

Information

The Real output y is a pulse signal:



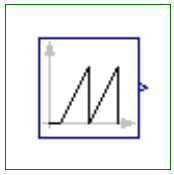
Parameters

Type	Name	Default	Description
Real	amplitude	1	Amplitude of pulse
Real	width	50	Width of pulse in % of periods
Time	period	1	Time for one period [s]
Real	offset	0	Offset of output signals
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

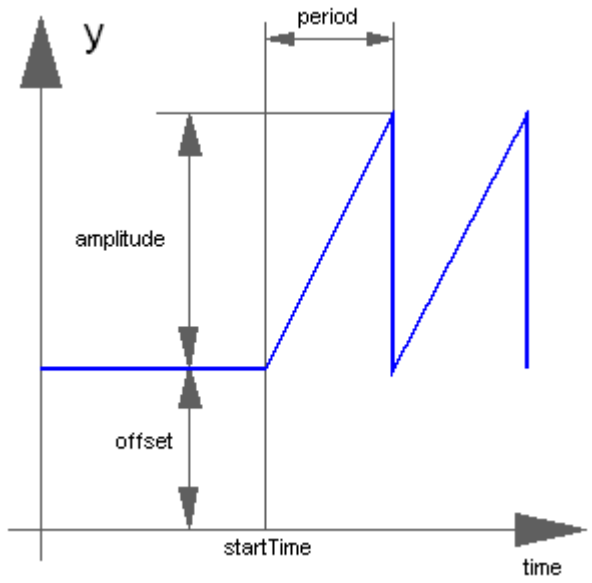
Modelica.Blocks.Sources.SawTooth



Generate saw tooth signal

Information

The Real output y is a saw tooth signal:



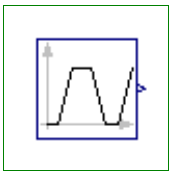
Parameters

Type	Name	Default	Description
Real	amplitude	1	Amplitude of saw tooth
Time	period	1	Time for one period [s]
Real	offset	0	Offset of output signals
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

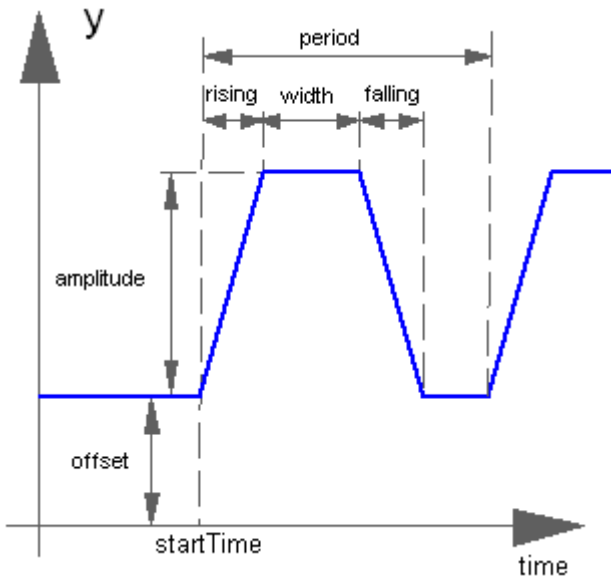
Modelica.Blocks.Sources.Trapezoid



Generate trapezoidal signal of type Real

Information

The Real output y is a trapezoid signal:



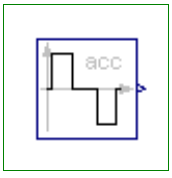
Parameters

Type	Name	Default	Description
Real	amplitude	1	Amplitude of trapezoid
Time	rising	0	Rising duration of trapezoid [s]
Time	width	0.5	Width duration of trapezoid [s]
Time	falling	0	Falling duration of trapezoid [s]
Time	period	1	Time for one period [s]
Integer	nperiod	-1	Number of periods (< 0 means infinite number of periods)
Real	offset	0	Offset of output signal
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

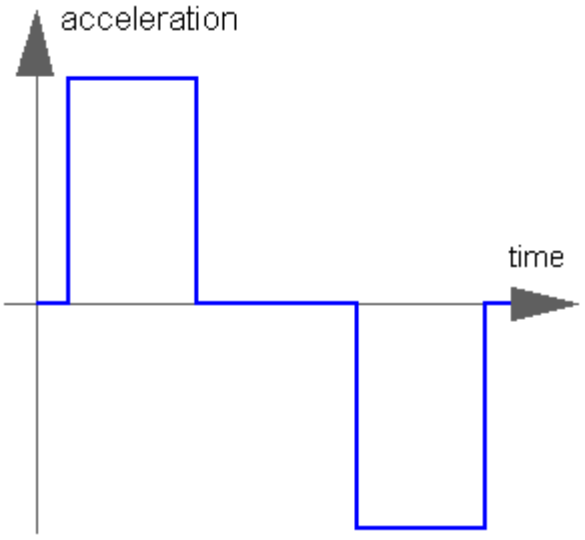
Modelica.Blocks.Sources.KinematicPTP



Move as fast as possible along a distance within given kinematic constraints

Information

The goal is to move as **fast** as possible along a distance **deltaq** under given **kinematical constraints**. The distance can be a positional or angular range. In robotics such a movement is called **PTP** (Point-To-Point). This source block generates the **acceleration** qdd of this signal as output:



After integrating the output two times, the position q is obtained. The signal is constructed in such a way that it is not possible to move faster, given the **maximally** allowed **velocity** qd_max and the **maximally** allowed **acceleration** qdd_max.

If several distances are given (vector deltaq has more than 1 element), an acceleration output vector is constructed such that all signals are in the same periods in the acceleration, constant velocity and deceleration phase. This means that only one of the signals is at its limits whereas the others are sychnronized in such a way that the end point is reached at the same time instant.

This element is useful to generate a reference signal for a controller which controls a drive train or in combination with model Modelica.Mechanics.Rotational.**Accelerate** to drive a flange according to a given acceleration.

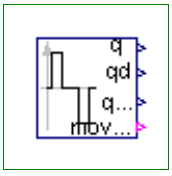
Parameters

Type	Name	Default	Description
Real	deltaq[:]	{1}	Distance to move
Real	qd_max[:]	{1}	Maximum velocities der(q)
Real	qdd_max[:]	{1}	Maximum accelerations der(qd)
Time	startTime	0	Time instant at which movement starts [s]

Connectors

Type	Name	Description
output RealOutput	y[nout]	Connector of Real output signals

Modelica.Blocks.Sources.KinematicPTP2

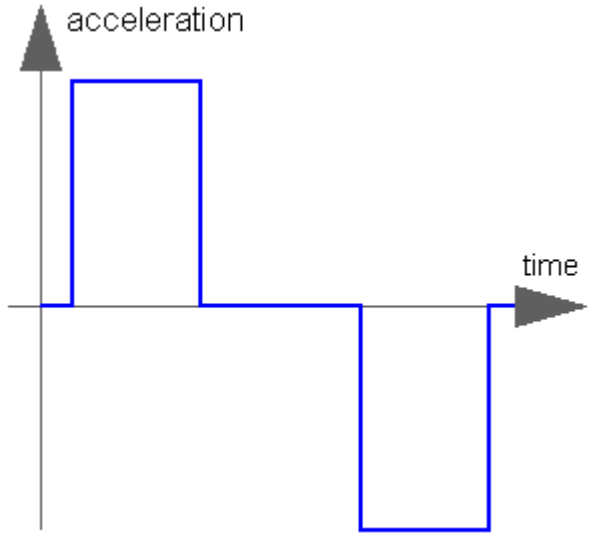


Move as fast as possible from start to end position within given kinematic constraints with output signals q, qd=der(q), qdd=der(qd)

Information

The goal is to move as **fast** as possible from start position **q_begin** to end position **q_end** under given **kinematical constraints**. The positions can be translational or rotational definitions (i.e., q_begin/q_end is given). In robotics such a movement is called **PTP** (Point-To-Point). This source block generates the **position** q(t), the **speed** qd(t) = der(q), and the **acceleration** qdd = der(qd) as output. The signals are

constructed in such a way that it is not possible to move faster, given the **maximally** allowed **velocity** qd_max and the **maximally** allowed **acceleration** qdd_max:



If vectors q_begin/q_end have more than 1 element, the output vectors are constructed such that all signals are in the same periods in the acceleration, constant velocity and deceleration phase. This means that only one of the signals is at its limits whereas the others are sychnronized in such a way that the end point is reached at the same time instant.

This element is useful to generate a reference signal for a controller which controls, e.g., a drive train, or to drive a flange according to a given acceleration.

Parameters

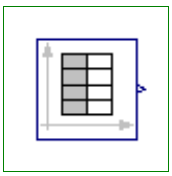
Type	Name	Default	Description
Real	q_begin[:]	{0}	Start position
Real	q_end[:]	{1}	End position
Real	qd_max[:]	{1}	Maximum velocities der(q)
Real	qdd_max[:]	{1}	Maximum accelerations der(qd)
Time	startTime	0	Time instant at which movement starts [s]

Connectors

Type	Name	Description
output RealOutput	q[nout]	Reference position of path planning
output RealOutput	qd[nout]	Reference speed of path planning
output RealOutput	qdd[nout]	Reference acceleration of path planning
output BooleanOutput	moving[nout]	= true, if end position not yet reached; = false, if end position reached or axis is completely at rest

---

Modelica.Blocks.Sources.TimeTable



Generate a (possibly discontinuous) signal by linear interpolation in a table

Information

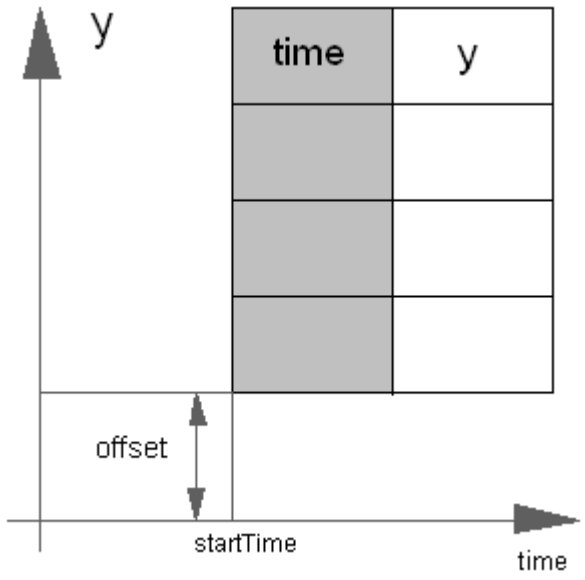
This block generates an output signal by **linear interpolation** in a table. The time points and function values

are stored in a matrix **table[i,j]**, where the first column table[:,1] contains the time points and the second column contains the data to be interpolated. The table interpolation has the following proporties:

- The time points need to be **monotonically increasing**.
- **Discontinuities** are allowed, by providing the same time point twice in the table.
- Values **outside** of the table range, are computed by **extrapolation** through the last or first two points of the table.
- If the table has only **one row**, no interpolation is performed and the function value is just returned independantly of the actual time instant.
- Via parameters **startTime** and **offset** the curve defined by the table can be shifted both in time and in the ordinate value.
- The table is implemented in a numerically sound way by generating **time events** at interval boundaries, in order to not integrate over a discontinuous or not differentiable points.

Example:

```
   table = [0  0
            1  0
            1  1
            2  4
            3  9
            4 16]
If, e.g., time = 1.0, the output y =  0.0 (before event), 1.0 (after event)
    e.g., time = 1.5, the output y =  2.5,
    e.g., time = 2.0, the output y =  4.0,
    e.g., time = 5.0, the output y = 23.0 (i.e. extrapolation).
```



Parameters

Type	Name	Default	Description
Real	table[:, 2]	[0, 0; 1, 1; 2, 4]	Table matrix (time = first column)
Real	offset	0	Offset of output signal
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description

output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Sources.CombiTimeTable

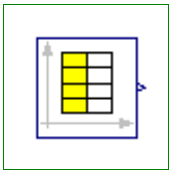
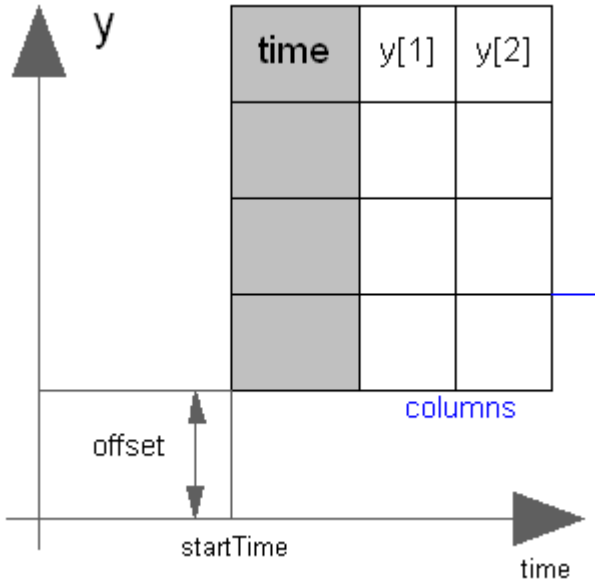


Table look-up with respect to time and linear/perodic extrapolation methods (data from matrix/file)

Information

This block generates an output signal y[:] by **linear interpolation** in a table. The time points and function values are stored in a matrix **table[i,j]**, where the first column table[:,1] contains the time points and the other columns contain the data to be interpolated.



Via parameter **columns** it can be defined which columns of the table are interpolated. If, e.g., columns={2,4}, it is assumed that 2 output signals are present and that the first output is computed by interpolation of column 2 and the second output is computed by interpolation of column 4 of the table matrix. The table interpolation has the following properties:

- The time points need to be **monotonically increasing**.
- **Discontinuities** are allowed, by providing the same time point twice in the table.
- Values **outside** of the table range, are computed by extrapolation according to the setting of parameter **extrapolation**:

```
extrapolation = 0: hold the first or last value of the table,
                   if outside of the range.
              = 1: extrapolate through the last or first two
                   points of the table.
              = 2: periodically repeat the table data
                   (periodical function).
```

- Via parameter **smoothness** it is defined how the data is interpolated:

```
smoothness = 0: linear interpolation
           = 1: smooth interpolation with Akima Splines such
                that der(y) is continuous.
```

- If the table has only **one row**, no interpolation is performed and the table values of this row are just returned.

- Via parameters **startTime** and **offset** the curve defined by the table can be shifted both in time and in the ordinate value. The time instants stored in the table are therefore **relative** to **startTime**. If time < startTime, no interpolation is performed and the offset is used as ordinate value for all outputs.
- The table is implemented in a numerically sound way by generating **time events** at interval boundaries, in order to not integrate over a discontinuous or not differentiable points.
- For special applications it is sometimes needed to know the minimum and maximum time instant defined in the table as a parameter. For this reason parameters **t_min** and **t_max** are provided and can be access from the outside of the table object.

Example:

```
   table = [0  0
            1  0
            1  1
            2  4
            3  9
            4 16]; extrapolation = 1 (default)
If, e.g., time = 1.0, the output y =  0.0 (before event), 1.0 (after event)
    e.g., time = 1.5, the output y =  2.5,
    e.g., time = 2.0, the output y =  4.0,
    e.g., time = 5.0, the output y = 23.0 (i.e. extrapolation via last 2
points).
```

The table matrix can be defined in the following ways:

1. Explicitly supplied as **parameter matrix** "table", and the other parameters have the following values:

   ```
   tableName is "NoName" or has only blanks,
   fileName  is "NoName" or has only blanks.
   ```

2. **Read** from a **file** "fileName" where the matrix is stored as "tableName". Both ASCII and binary file format is possible. (the ASCII format is described below). It is most convenient to generate the binary file from Matlab (Matlab 4 storage format), e.g., by command

   ```
   save tables.mat tab1 tab2 tab3 -V4
   ```

   when the three tables tab1, tab2, tab3 should be used from the model.
3. Statically stored in function "usertab" in file "usertab.c". The matrix is identified by "tableName". Parameter fileName = "NoName" or has only blanks.

Table definition methods (1) and (3) do **not** allocate dynamic memory, and do not access files, whereas method (2) does. Therefore (1) and (3) are suited for hardware-in-the-loop simulation (e.g. with dSpace hardware). When the constant "NO_FILE" is defined in "usertab.c", all parts of the source code of method (2) are removed by the C-preprocessor, such that no dynamic memory allocation and no access to files takes place.

If tables are read from an ASCII-file, the file need to have the following structure ("-----" is not part of the file content):

```
-------------------------------------------------
#1
double tab1(6,2)    # comment line
  0   0
  1   0
  1   1
  2   4
  3   9
  4  16
double tab2(6,2)    # another comment line
  0   0
  2   0
```

```
  2    2
  4    8
  6   18
  8   32
-------------------------------------------------------
```

Note, that the first two characters in the file need to be "#1". Afterwards, the corresponding matrix has to be declared with type, name and actual dimensions. Finally, in successive rows of the file, the elements of the matrix have to be given. Several matrices may be defined one after another.

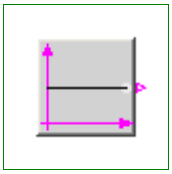
Parameters

Type	Name	Default	Description
table data definition			
Boolean	tableOnFile	false	= true, if table is defined on file or in function usertab
Real	table[:, :]	fill(0.0, 0, 2)	Table matrix (time = first column)
String	tableName	"NoName"	Table name on file or in function usertab (see docu)
String	fileName	"NoName"	File where matrix is stored
table data interpretation			
Integer	columns[:]	2:size(table, 2)	Columns of table to be interpolated
Temp	smoothness	Blocks.Types.Smoothness.Line...	Smoothness of table interpolation
Temp	extrapolation	Blocks.Types.Extrapolation.L...	Extrapolation of data outside the definition range
Real	offset[:]	{0}	Offsets of output signals
Time	startTime	0	Output = offset for time < startTime [s]

Connectors

Type	Name	Description
output RealOutput	y[nout]	Connector of Real output signals

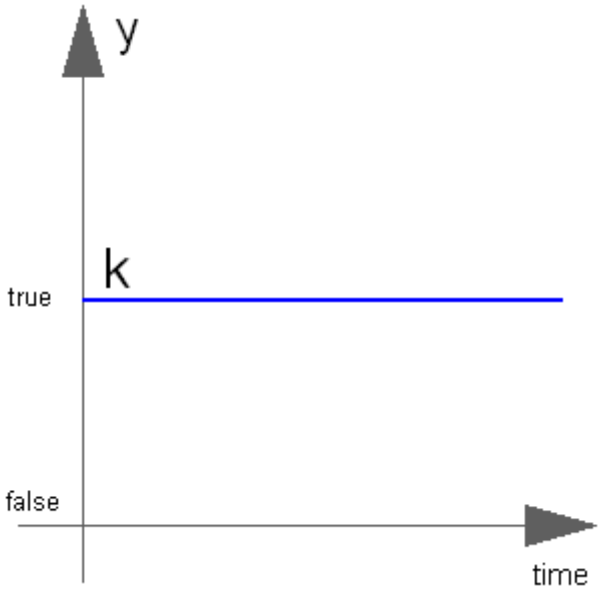
Modelica.Blocks.Sources.BooleanConstant



Generate constant signal of type Boolean

Information

The Boolean output y is a constant signal:



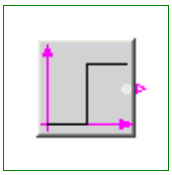
Parameters

Type	Name	Default	Description
Boolean	k	true	Constant output value

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal

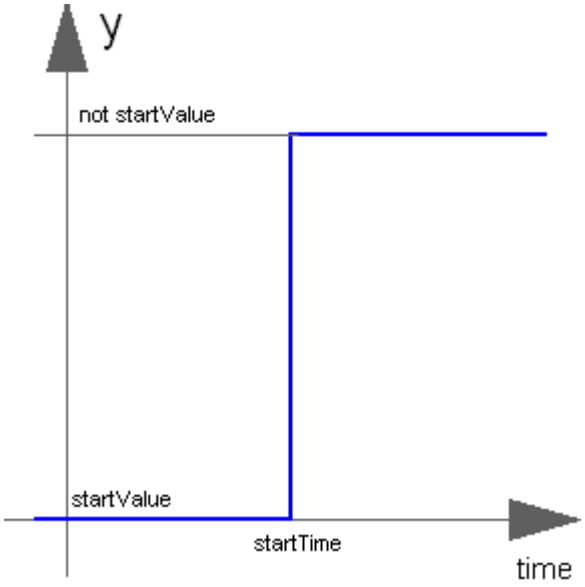
Modelica.Blocks.Sources.BooleanStep



Generate step signal of type Boolean

Information

The Boolean output y is a step signal:



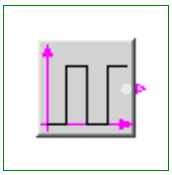
Parameters

Type	Name	Default	Description
Time	startTime	0	Time instant of step start [s]
Boolean	startValue	false	Output before startTime

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal

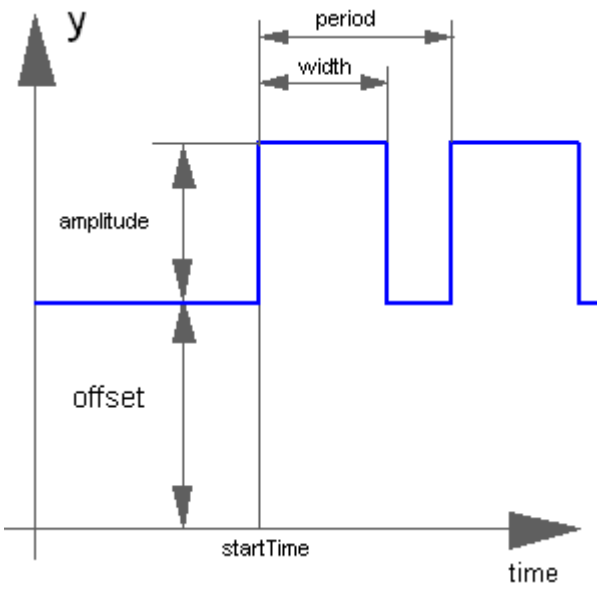
Modelica.Blocks.Sources.BooleanPulse



Generate pulse signal of type Boolean

Information

The Boolean output y is a pulse signal:



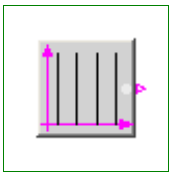
Parameters

Type	Name	Default	Description
Real	width	50	Width of pulse in % of period
Time	period	1	Time for one period [s]
Time	startTime	0	Time instant of first pulse [s]

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal

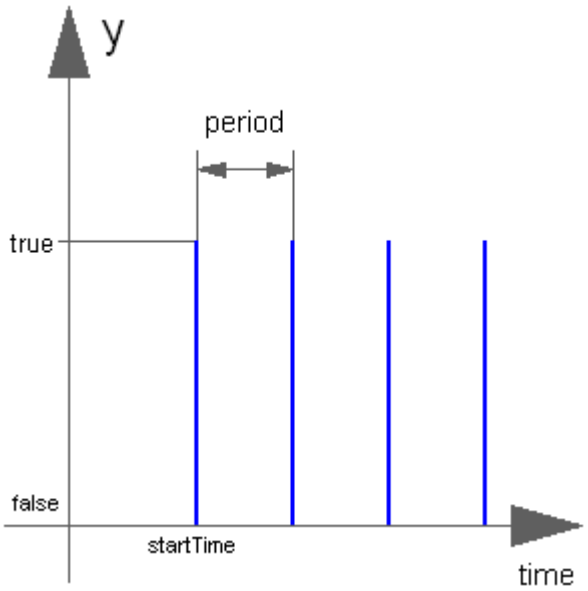
Modelica.Blocks.Sources.SampleTrigger



Generate sample trigger signal

Information

The Boolean output y is a trigger signal where the output y is only **true** at sample times (defined by parameter **period**) and is otherwise **false**.



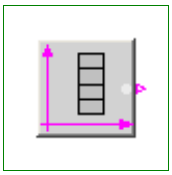
Parameters

Type	Name	Default	Description
Time	period	0.01	Sample period [s]
Time	startTime	0	Time instant of first sample trigger [s]

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal

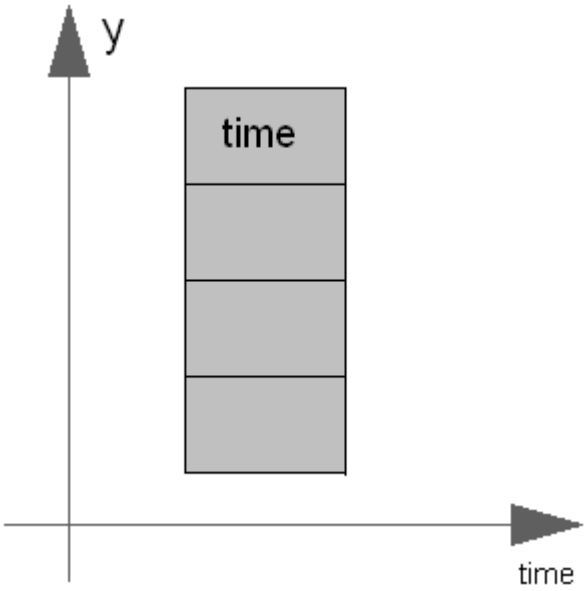
Modelica.Blocks.Sources.BooleanTable



Generate a Boolean output signal based on a vector of time instants

Information

The Boolean output y is a signal defined by parameter vector **table**. In the vector time points are stored. At every time point, the output y changes its value to the negated value of the previous one.



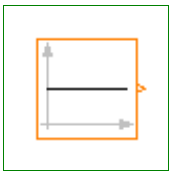
Parameters

Type	Name	Default	Description
Boolean	startValue	false	Start value of y. At time = table[1], y changes to 'not startValue'
Time	table[:]		Vector of time points. At every time point, the output y gets its opposite value [s]

Connectors

Type	Name	Description
output BooleanOutput	y	Connector of Boolean output signal

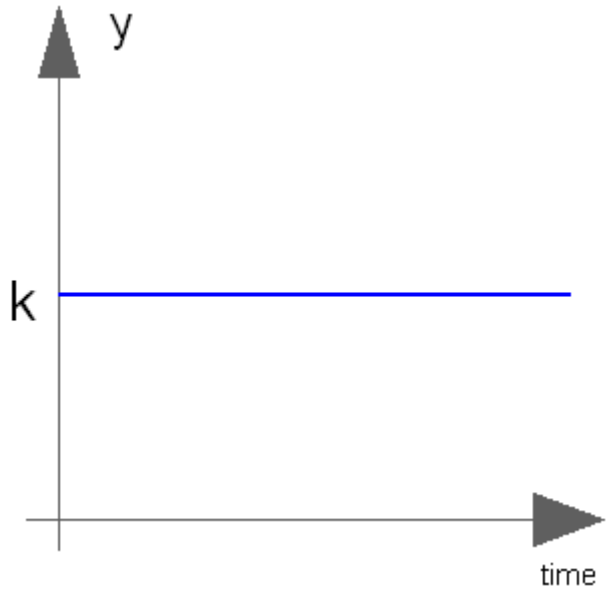
Modelica.Blocks.Sources.IntegerConstant



Generate constant signal of type Integer

Information

The Integer output y is a constant signal:



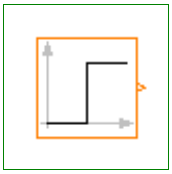
Parameters

Type	Name	Default	Description
Integer	k	1	Constant output value

Connectors

Type	Name	Description
output IntegerOutput	y	Connector of Integer output signal

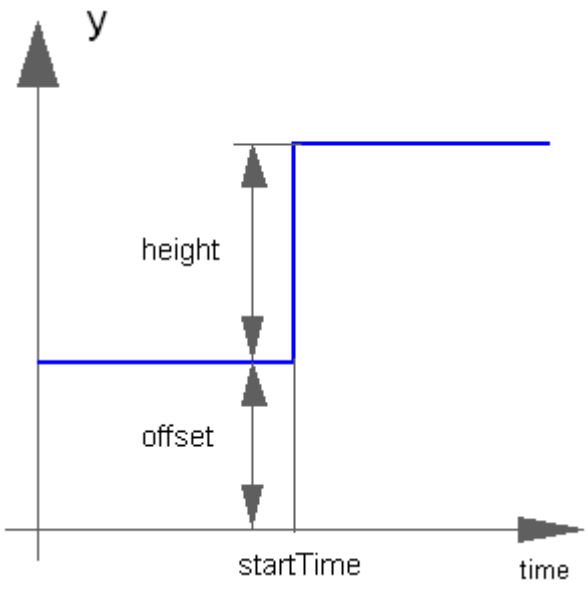
Modelica.Blocks.Sources.IntegerStep



Generate step signal of type Integer

Information

The Integer output y is a step signal:



Parameters

Type	Name	Default	Description
Integer	height	1	Height of step
Integer	offset	0	Offset of output signal y
Time	startTime	0	Output y = offset for time < startTime [s]

Connectors

Type	Name	Description
output IntegerOutput	y	Connector of Integer output signal

Modelica.Blocks.Tables

Library of blocks to interpolate in one and two-dimensional tables

Information

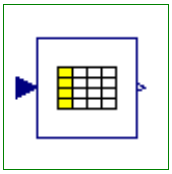
This package contains blocks for one- and two-dimensional interpolation in tables.

Package Content

Name	Description
CombiTable1D	Table look-up in one dimension (matrix/file) with n inputs and n outputs
CombiTable1Ds	Table look-up in one dimension (matrix/file) with one input and n outputs
CombiTable2D	Table look-up in two dimensions (matrix/file)

Modelica.Blocks.Tables.CombiTable1D

Table look-up in one dimension (matrix/file) with n inputs and n outputs



Information

Linear interpolation in **one** dimension of a **table**. Via parameter **columns** it can be defined how many columns of the table are interpolated. If, e.g., columns={2,4}, it is assumed that 2 input and 2 output signals are present and that the first output interpolates the first input via column 2 and the second output interpolates the second input via column 4 of the table matrix.

The grid points and function values are stored in a matrix "table[i,j]", where the first column "table[:,1]" contains the grid points and the other columns contain the data to be interpolated. Example:

```
table = [0,  0;
         1,  1;
         2,  4;
         4, 16]
If, e.g., the input u = 1.0, the output y =  1.0,
    e.g., the input u = 1.5, the output y =  2.5,
    e.g., the input u = 2.0, the output y =  4.0,
    e.g., the input u =-1.0, the output y = -1.0 (i.e. extrapolation).
```

- The interpolation is **efficient**, because a search for a new interpolation starts at the interval used in the last call.
- If the table has only **one row**, the table value is returned, independent of the value of the input signal.
- If the input signal **u[i]** is **outside** of the defined **interval**, i.e., u[i] > table[size(table,1),i+1] or u[i] < table[1,1], the corresponding value is also determined by linear interpolation through the last or first two points of the table.
- The grid values (first column) have to be **strict** monotonically increasing.

The table matrix can be defined in the following ways:

1. Explicitly supplied as **parameter matrix** "table", and the other parameters have the following values:

   ```
   tableName is "NoName" or has only blanks,
   fileName  is "NoName" or has only blanks.
   ```

2. **Read** from a **file** "fileName" where the matrix is stored as "tableName". Both ASCII and binary file format is possible. (the ASCII format is described below). It is most convenient to generate the binary file from Matlab (Matlab 4 storage format), e.g., by command

   ```
   save tables.mat tab1 tab2 tab3 -V4
   ```

   when the three tables tab1, tab2, tab3 should be used from the model.
3. Statically stored in function "usertab" in file "usertab.c". The matrix is identified by "tableName". Parameter fileName = "NoName" or has only blanks.

Table definition methods (1) and (3) do **not** allocate dynamic memory, and do not access files, whereas method (2) does. Therefore (1) and (3) are suited for hardware-in-the-loop simulation (e.g. with dSpace hardware). When the constant "NO_FILE" is defined in "usertab.c", all parts of the source code of method (2) are removed by the C-preprocessor, such that no dynamic memory allocation and no access to files takes place.

If tables are read from an ASCII-file, the file need to have the following structure ("-----" is not part of the file content):

```
-------------------------------------------------
#1
double tab1(5,2)    # comment line
  0   0
  1   1
  2   4
  3   9
  4  16
```

```
double tab2(5,2)    # another comment line
  0   0
  2   2
  4   8
  6  18
  8  32
-----------------------------------------------------
```

Note, that the first two characters in the file need to be "#1". Afterwards, the corresponding matrix has to be declared with type, name and actual dimensions. Finally, in successive rows of the file, the elements of the matrix have to be given. Several matrices may be defined one after another.

Parameters

Type	Name	Default	Description
table data definition			
Boolean	tableOnFile	false	true, if table is defined on file or in function usertab
Real	table[:, :]	fill(0.0, 0, 2)	table matrix (grid = first column)
String	tableName	"NoName"	table name on file or in function usertab (see docu)
String	fileName	"NoName"	file where matrix is stored
table data interpretation			
Integer	columns[:]	2:size(table, 2)	columns of table to be interpolated
Temp	smoothness	Types.Smoothness.LinearSegme...	smoothness of table interpolation

Connectors

Type	Name	Description
input RealInput	u[n]	Connector of Real input signals
output RealOutput	y[n]	Connector of Real output signals

Modelica.Blocks.Tables.CombiTable1Ds

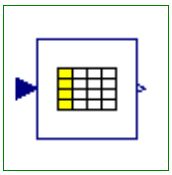


Table look-up in one dimension (matrix/file) with one input and n outputs

Information

Linear interpolation in **one** dimension of a **table**. Via parameter **columns** it can be defined how many columns of the table are interpolated. If, e.g., icol={2,4}, it is assumed that one input and 2 output signals are present and that the first output interpolates via column 2 and the second output interpolates via column 4 of the table matrix.

The grid points and function values are stored in a matrix "table[i,j]", where the first column "table[:,1]" contains the grid points and the other columns contain the data to be interpolated. Example:

```
table = [0,  0;
         1,  1;
         2,  4;
         4, 16]
If, e.g., the input u = 1.0, the output y =  1.0,
    e.g., the input u = 1.5, the output y =  2.5,
    e.g., the input u = 2.0, the output y =  4.0,
    e.g., the input u =-1.0, the output y = -1.0 (i.e. extrapolation).
```

- The interpolation is **efficient**, because a search for a new interpolation starts at the interval used in the last call.
- If the table has only **one row**, the table value is returned, independent of the value of the input signal.
- If the input signal **u** is **outside** of the defined **interval**, i.e., u > table[size(table,1),1] or u < table[1,1], the corresponding value is also determined by linear interpolation through the last or first two points of the table.
- The grid values (first column) have to be **strict** monotonically increasing.

The table matrix can be defined in the following ways:

1. Explicitly supplied as **parameter matrix** "table", and the other parameters have the following values:

   ```
   tableName is "NoName" or has only blanks,
   fileName  is "NoName" or has only blanks.
   ```

2. **Read** from a **file** "fileName" where the matrix is stored as "tableName". Both ASCII and binary file format is possible. (the ASCII format is described below). It is most convenient to generate the binary file from Matlab (Matlab 4 storage format), e.g., by command

   ```
   save tables.mat tab1 tab2 tab3 -V4
   ```

   when the three tables tab1, tab2, tab3 should be used from the model.
3. Statically stored in function "usertab" in file "usertab.c". The matrix is identified by "tableName". Parameter fileName = "NoName" or has only blanks.

Table definition methods (1) and (3) do **not** allocate dynamic memory, and do not access files, whereas method (2) does. Therefore (1) and (3) are suited for hardware-in-the-loop simulation (e.g. with dSpace hardware). When the constant "NO_FILE" is defined, all parts of the source code of method (2) are removed by the C-preprocessor, such that no dynamic memory allocation and no access to files takes place.

If tables are read from an ASCII-file, the file need to have the following structure ("-----" is not part of the file content):

```
-------------------------------------------------
#1
double tab1(5,2)    # comment line
  0   0
  1   1
  2   4
  3   9
  4  16
double tab2(5,2)    # another comment line
  0   0
  2   2
  4   8
  6  18
  8  32
-------------------------------------------------
```

Note, that the first two characters in the file need to be "#1". Afterwards, the corresponding matrix has to be declared with type, name and actual dimensions. Finally, in successive rows of the file, the elements of the matrix have to be given. Several matrices may be defined one after another.

Parameters

Type	Name	Default	Description
table data definition			
Boolean	tableOnFile	false	true, if table is defined on file or in function usertab

Real	table[:, :]	fill(0.0, 0, 2)	table matrix (grid = first column)
String	tableName	"NoName"	table name on file or in function usertab (see docu)
String	fileName	"NoName"	file where matrix is stored
table data interpretation			
Integer	columns[:]	2:size(table, 2)	columns of table to be interpolated
Temp	smoothness	Types.Smoothness.LinearSegme...	smoothness of table interpolation

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y[nout]	Connector of Real output signals

Modelica.Blocks.Tables.CombiTable2D

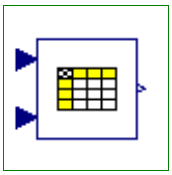


Table look-up in two dimensions (matrix/file)

Information

Linear interpolation in **two** dimensions of a **table**. The grid points and function values are stored in a matrix "table[i,j]", where:

- the first column "table[2:,1]" contains the u[1] grid points,
- the first row "table[1,2:]" contains the u[2] grid points,
- the other rows and columns contain the data to be interpolated.

Example:

```
          |       |       |       |
          |  1.0  |  2.0  |  3.0  |  // u2
     -----*-------*-------*-------*
     1.0  |  1.0  |  3.0  |  5.0  |
     -----*-------*-------*-------*
     2.0  |  2.0  |  4.0  |  6.0  |
     -----*-------*-------*-------*
   // u1
is defined as
   table = [0.0,   1.0,   2.0,   3.0;
            1.0,   1.0,   3.0,   5.0;
            2.0,   2.0,   4.0,   6.0]
If, e.g. the input u is [1.0;1.0], the output y is 1.0,
    e.g. the input u is [2.0;1.5], the output y is 3.0.
```

- The interpolation is **efficient**, because a search for a new interpolation starts at the interval used in the last call.
- If the table has only **one element**, the table value is returned, independent of the value of the input signal.
- If the input signal **u1** or **u2** is **outside** of the defined **interval**, the corresponding value is also determined by linear interpolation through the last or first two points of the table.
- The grid values (first column and first row) have to be **strict** monotonically increasing.

The table matrix can be defined in the following ways:

1. Explicitly supplied as **parameter matrix** "table", and the other parameters have the following values:

   ```
   tableName is "NoName" or has only blanks,
   ```

```
fileName  is "NoName" or has only blanks.
```

2. **Read** from a **file** "fileName" where the matrix is stored as "tableName". Both ASCII and binary file format is possible. (the ASCII format is described below). It is most convenient to generate the binary file from Matlab (Matlab 4 storage format), e.g., by command

   ```
   save tables.mat tab1 tab2 tab3 -V4
   ```

   when the three tables tab1, tab2, tab3 should be used from the model.
3. Statically stored in function "usertab" in file "usertab.c". The matrix is identified by "tableName". Parameter fileName = "NoName" or has only blanks.

Table definition methods (1) and (3) do **not** allocate dynamic memory, and do not access files, whereas method (2) does. Therefore (1) and (3) are suited for hardware-in-the-loop simulation (e.g. with dSpace hardware). When the constant "NO_FILE" is defined, all parts of the source code of method (2) are removed by the C-preprocessor, such that no dynamic memory allocation and no access to files takes place.

If tables are read from an ASCII-file, the file need to have the following structure ("-----" is not part of the file content):

```
-------------------------------------------------------
#1
double tab1(5,2)    # comment line
  0   0
  1   1
  2   4
  3   9
  4  16
double tab2(5,2)    # another comment line
  0   0
  2   2
  4   8
  6  18
  8  32
-------------------------------------------------------
```

Note, that the first two characters in the file need to be "#1". Afterwards, the corresponding matrix has to be declared with type, name and actual dimensions. Finally, in successive rows of the file, the elements of the matrix have to be given. Several matrices may be defined one after another.

Parameters

Type	Name	Default	Description
table data definition			
Boolean	tableOnFile	false	true, if table is defined on file or in function usertab
Real	table[:, :]	fill(0.0, 0, 2)	table matrix (grid u1 = first column, grid u2 = first row)
String	tableName	"NoName"	table name on file or in function usertab (see docu)
String	fileName	"NoName"	file where matrix is stored
table data interpretation			
Temp	smoothness	Types.Smoothness.LinearSegme...	smoothness of table interpolation

Connectors

Type	Name	Description

input RealInput	u1	Connector of Real input signal 1
input RealInput	u2	Connector of Real input signal 2
output RealOutput	y	Connector of Real output signal

Modelica.Blocks.Types

Library of constants and types with choices, especially to build menus

Information

In this package **types** and **constants** are defined that are used in library Modelica.Blocks. The types have additional annotation choices definitions that define the menus to be built up in the graphical user interface when the type is used as parameter in a declaration.

Package Content

Name	Description
Extrapolation	Type, constants and menu choices to define the extrapolation of time table interpolation
Init	Type, constants and menu choices to define initialization of blocks
InitPID	Type, constants and menu choices to define initialization of PID and LimPID blocks
SimpleController	Type, constants and menu choices to define a simple controller type
Smoothness	Type, constants and menu choices to define the smoothness of table interpolation
StateSelection	Type, constants and menu choices to define state selection of variables

Modelica.Blocks.Types.Extrapolation

Type, constants and menu choices to define the extrapolation of time table interpolation

Information

Package Content

Name	Description
HoldLastPoint=0	Hold the last table point outside of the table scope
LastTwoPoints=1	Extrapolate linearly through the last two table points outside of the table scope
Periodic=2	Repeat the table scope periodically
Temp	Temporary type of Extrapolation with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer HoldLastPoint=0
  "Hold the last table point outside of the table scope";

  constant Integer LastTwoPoints=1
  "Extrapolate linearly through the last two table points outside of the table
scope";

  constant Integer Periodic=2 "Repeat the table scope periodically";
```

```
  type Temp
  "Temporary type of Extrapolation with choices for menus (until enumerations
are available)"

    extends Modelica.Icons.TypeInteger;
  end Temp;
```

Modelica.Blocks.Types.Init

Type, constants and menu choices to define initialization of blocks

Information

Package Content

Name	Description
NoInit=1	no initialization (start values are used as guess values with fixed=false)
SteadyState=2	steady state initialization (derivatives of states are zero)
InitialState=3	initialization with initial states
InitialOutput=4	initialization with initial outputs (and steady state of the states if possibles)
Temp	Temporary type of initialization with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer NoInit=1
  "no initialization (start values are used as guess values with fixed=false)";

  constant Integer SteadyState=2
  "steady state initialization (derivatives of states are zero)";

  constant Integer InitialState=3 "initialization with initial states";

  constant Integer InitialOutput=4
  "initialization with initial outputs (and steady state of the states if
possibles)";

  type Temp
  "Temporary type of initialization with choices for menus (until enumerations
are available)"
    extends Modelica.Icons.TypeInteger(min=1,max=4);

  end Temp;
```

Modelica.Blocks.Types.InitPID

Type, constants and menu choices to define initialization of PID and LimPID blocks

Information

This initialization type is identical to Types.Init and has just one additional option

DoNotUse_InitialIntegratorState. This options is only introduced in order that the default initialization for the Continuous.PID and Continuous.LimPID blocks are backward compatible. In Modelica 2.2, the integrators have been initialized with their given states wheresas the D-part has not been initialized. The option "DoNotUse_InitialIntegratorState" leads to this initialization definition.

Package Content

Name	Description
NoInit=1	no initialization (start values are used as guess values with fixed=false)
SteadyState=2	steady state initialization (derivatives of states are zero)
InitialState=3	initialization with initial states
InitialOutput=4	initialization with initial outputs (and steady state of the states if possibles)
DoNotUse_InitialIntegratorState=5	don't use, only for backward compatibility (initialize only integrator state)
Temp	Temporary type of initialization with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer NoInit=1
  "no initialization (start values are used as guess values with fixed=false)";

  constant Integer SteadyState=2
  "steady state initialization (derivatives of states are zero)";

  constant Integer InitialState=3 "initialization with initial states";

  constant Integer InitialOutput=4
  "initialization with initial outputs (and steady state of the states if
possibles)";

  constant Integer DoNotUse_InitialIntegratorState=5
  "don't use, only for backward compatibility (initialize only integrator
state)";

  type Temp
  "Temporary type of initialization with choices for menus (until enumerations
are available)"
    extends Modelica.Icons.TypeInteger(min=1,max=5);

  end Temp;
```

Modelica.Blocks.Types.SimpleController

Type, constants and menu choices to define a simple controller type

Information

Package Content

Name	Description

P=1	P controller
PI=2	PI controller
PD=3	PD controller
PID=4	PID controller
Temp	Temporary type of simple controller type with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer P=1 "P controller";

  constant Integer PI=2 "PI controller";

  constant Integer PD=3 "PD controller";

  constant Integer PID=4 "PID controller";

  type Temp
  "Temporary type of simple controller type with choices for menus (until
enumerations are available)"
    extends Modelica.Icons.TypeInteger(min=1,max=4);

  end Temp;
```

Modelica.Blocks.Types.Smoothness

Type, constants and menu choices to define the smoothness of table interpolation

Information

Interpolation type of a table

Package Content

Name	Description
LinearSegments=0	Table points are linearly interpolated
ContinuousDerivative=1	Table points are interpolated such that the first derivative is continuous
Temp	Temporary type of Smoothness with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer LinearSegments=0 "Table points are linearly interpolated";

  constant Integer ContinuousDerivative=1
  "Table points are interpolated such that the first derivative is continuous";

  type Temp
  "Temporary type of Smoothness with choices for menus (until enumerations are
available)"

    extends Modelica.Icons.TypeInteger;
```

```
  end Temp;
```

Modelica.Blocks.Types.StateSelection

Type, constants and menu choices to define state selection of variables

Information

Type to define the stateSelection attribute.

Package Content

Name	Description
Never=1	Never (never use as state)
Avoid=2	Avoid (avoid to use as state)
Default=3	Default (default behaviour)
Prefer=4	Prefer (use as state if possible)
Always=5	Always (always use as state)
Temp	Temporary type of state selection with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer Never=1 "Never (never use as state)";

  constant Integer Avoid=2 "Avoid (avoid to use as state)";

  constant Integer Default=3 "Default (default behaviour)";

  constant Integer Prefer=4 "Prefer (use as state if possible)";

  constant Integer Always=5 "Always (always use as state)";

  type Temp
  "Temporary type of state selection with choices for menus (until enumerations
are available)"
    extends Modelica.Icons.TypeInteger(min=1,max=5);

  end Temp;
```

Modelica.Constants

Library of mathematical constants and constants of nature (e.g., pi, eps, R, sigma)

Information

This package provides often needed constants from mathematics, machine dependent constants and constants from nature. The latter constants (name, value, description) are from the following source:

Peter J. Mohr and Barry N. Taylor (1999):
CODATA Recommended Values of the Fundamental Physical Constants: 1998. Journal of

Physical and Chemical Reference Data, Vol. 28, No. 6, 1999 and Reviews of Modern Physics, Vol. 72, No. 2, 2000. See also http://physics.nist.gov/cuu/Constants/

CODATA is the Committee on Data for Science and Technology.

Main Author:
Martin Otter
Deutsches Zentrum für Luft und Raumfahrt e. V. (DLR)
Oberpfaffenhofen
Postfach 11 16
D-82230 Weßling
email: Martin.Otter@dlr.de

Copyright © 1998-2007, Modelica Association and DLR.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
e=Modelica.Math.exp(1.0)	
pi=2*Modelica.Math.asin(1.0)	
D2R=pi/180	Degree to Radian
R2D=180/pi	Radian to Degree
eps=1.e-15	Biggest number such that 1.0 + eps = 1.0
small=1.e-60	Smallest number such that small and -small are representable on the machine
inf=1.e+60	Biggest Real number such that inf and -inf are representable on the machine
Integer_inf=2147483647	Biggest Integer number such that Integer_inf and -Integer_inf are representable on the machine
c=299792458	Speed of light in vacuum
g_n=9.80665	Standard acceleration of gravity on earth
G=6.6742e-11	Newtonian constant of gravitation
h=6.6260693e-34	Planck constant
k=1.3806505e-23	Boltzmann constant
R=8.314472	Molar gas constant
sigma=5.670400e-8	Stefan-Boltzmann constant
N_A=6.0221415e23	Avogadro constant
mue_0=4*pi*1.e-7	Magnetic constant
epsilon_0=1/(mue_0*c*c)	Electric constant
T_zero=-273.15	Absolute zero temperature

Types and constants

```
constant Real e=Modelica.Math.exp(1.0);

constant Real pi=2*Modelica.Math.asin(1.0);

constant Real D2R=pi/180 "Degree to Radian";
```

```
  constant Real R2D=180/pi "Radian to Degree";

  constant Real eps=1.e-15 "Biggest number such that 1.0 + eps = 1.0";

  constant Real small=1.e-60
  "Smallest number such that small and -small are representable on the machine";

  constant Real inf=1.e+60
  "Biggest Real number such that inf and -inf are representable on the machine";

  constant Integer Integer_inf=2147483647
  "Biggest Integer number such that Integer_inf and -Integer_inf are
representable on the machine";

  constant SI.Velocity c=299792458 "Speed of light in vacuum";

  constant SI.Acceleration g_n=9.80665
  "Standard acceleration of gravity on earth";

  constant Real G(final unit="m3/(kg.s2)") = 6.6742e-11
  "Newtonian constant of gravitation";

  constant Real h(final unit="J.s") = 6.6260693e-34 "Planck constant";

  constant Real k(final unit="J/K") = 1.3806505e-23 "Boltzmann constant";

  constant Real R(final unit="J/(mol.K)") = 8.314472 "Molar gas constant";

  constant Real sigma(final unit="W/(m2.K4)") = 5.670400e-8
  "Stefan-Boltzmann constant";

  constant Real N_A(final unit="1/mol") = 6.0221415e23 "Avogadro constant";

  constant Real mue_0(final unit="N/A2") = 4*pi*1.e-7 "Magnetic constant";

  constant Real epsilon_0(final unit="F/m") = 1/(mue_0*c*c) "Electric constant";

  constant NonSI.Temperature_degC T_zero=-273.15 "Absolute zero temperature";
```

Modelica.Electrical

Library of electrical models (analog, digital, machines, multi-phase)

Information

This library contains electrical components to build up analog and digital circuits, as well as machines to model electrical motors and generators, especially three phase induction machines such as an asynchronous motor.

Package Content

Name	Description

Analog	Library for analog electrical models
Digital	Library for digital electrical components based on the VHDL standard with 9-valued logic and conversion to 2-,3-,4-valued logic
Machines	Library for electric machines
MultiPhase	Library for electrical components with 2, 3 or more phases

Modelica.Electrical.Analog

Library for analog electrical models

Information

This package contains packages for analog electrical components:

- Basic: basic components (resistor, capacitor, conductor, inductor, transformer, gyrator)
- Semiconductors: semiconductor devices (diode, bipolar and MOS transistors)
- Lines: transmission lines (lossy and lossless)
- Ideal: ideal elements (switches, diode, transformer, idle, short, ...)
- Sources: time-dependend and controlled voltage and current sources
- Sensors: sensors to measure potential, voltage, and current

Main Authors:

Christoph Clauß <clauss@eas.iis.fhg.de>
André Schneider <schneider@eas.iis.fhg.de>
Fraunhofer Institute for Integrated Circuits
Design Automation Department
Zeunerstraße 38
D-01069 Dresden

Copyright © 1998-2007, Modelica Association and Fraunhofer-Gesellschaft.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Examples	Examples that demonstrate the usage of the Analog electrical components
Basic	Basic electrical components such as resistor, capacitor, transformer
Ideal	Ideal electrical elements such as switches, diode, transformer, operational amplifier
Interfaces	Connectors and partial models for Analog electrical components
Lines	Lossy and lossless segmented transmission lines, and LC distributed line models
Semiconductors	Semiconductor devices such as diode, MOS and bipolar transistor
Sensors	Potential, voltage, current, and power sensors
Sources	Time-dependend and controlled voltage and current sources

Modelica.Electrical.Analog.Examples

Examples that demonstrate the usage of the Analog electrical components

Information

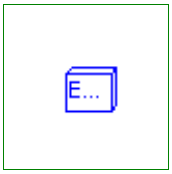
This package contains examples that demonstrate the usage of the components of the Electrical.Analog library.

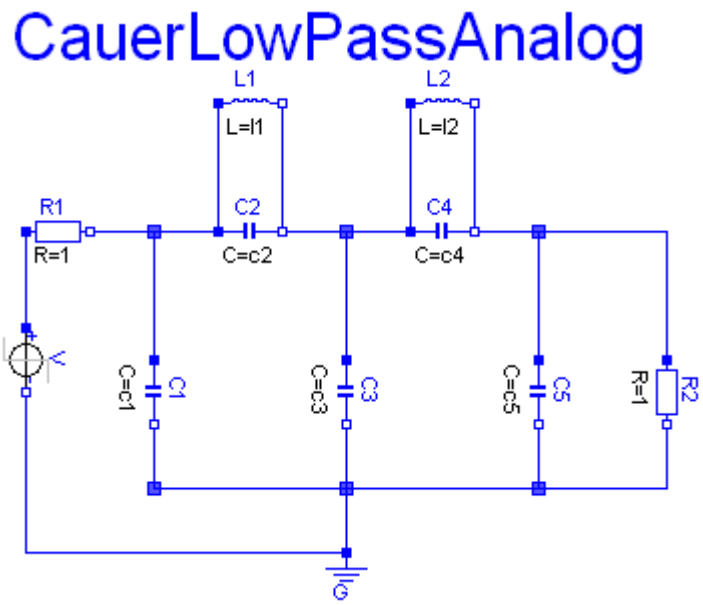
Package Content

Name	Description
CauerLowPassAnalog	Cauer low pass filter with analog components
CauerLowPassOPV	Cauer low pass filter with operational amplifiers
CauerLowPassSC	Cauer low-pass filter with operational amplifiers and switched capacitors
CharacteristicIdealDiodes	Characteristic of ideal diodes
CharacteristicThyristors	Characteristic of ideal thyristors
ChuaCircuit	Chua's circuit, ns, V, A
DifferenceAmplifier	Simple NPN transistor amplifier circuit
HeatingMOSInverter	Heating MOS Inverter
HeatingNPN_OrGate	Heating NPN Or Gate
HeatingRectifier	Heating rectifier
NandGate	CMOS NAND Gate (see Tietze/Schenk, page 157)
Rectifier	B6 diode bridge
ShowSaturatingInductor	Simple demo to show behaviour of SaturatingInductor component
ShowVariableResistor	Simple demo of a VariableResistor model
Utilities	Utility components used by package Examples

Modelica.Electrical.Analog.Examples.CauerLowPassAnalog

Cauer low pass filter with analog components





Information

The example Cauer Filter is a low-pass-filter of the fifth order. It is realized using an analog network. The voltage source V is the input voltage (step), and the R2.p.v is the filter output voltage. The pulse response is calculated.

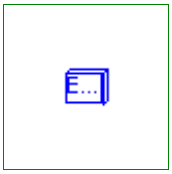
The simulation end time should be 60. Please plot both V.p.v (input voltage) and R2.p.v (output voltage).

Parameters

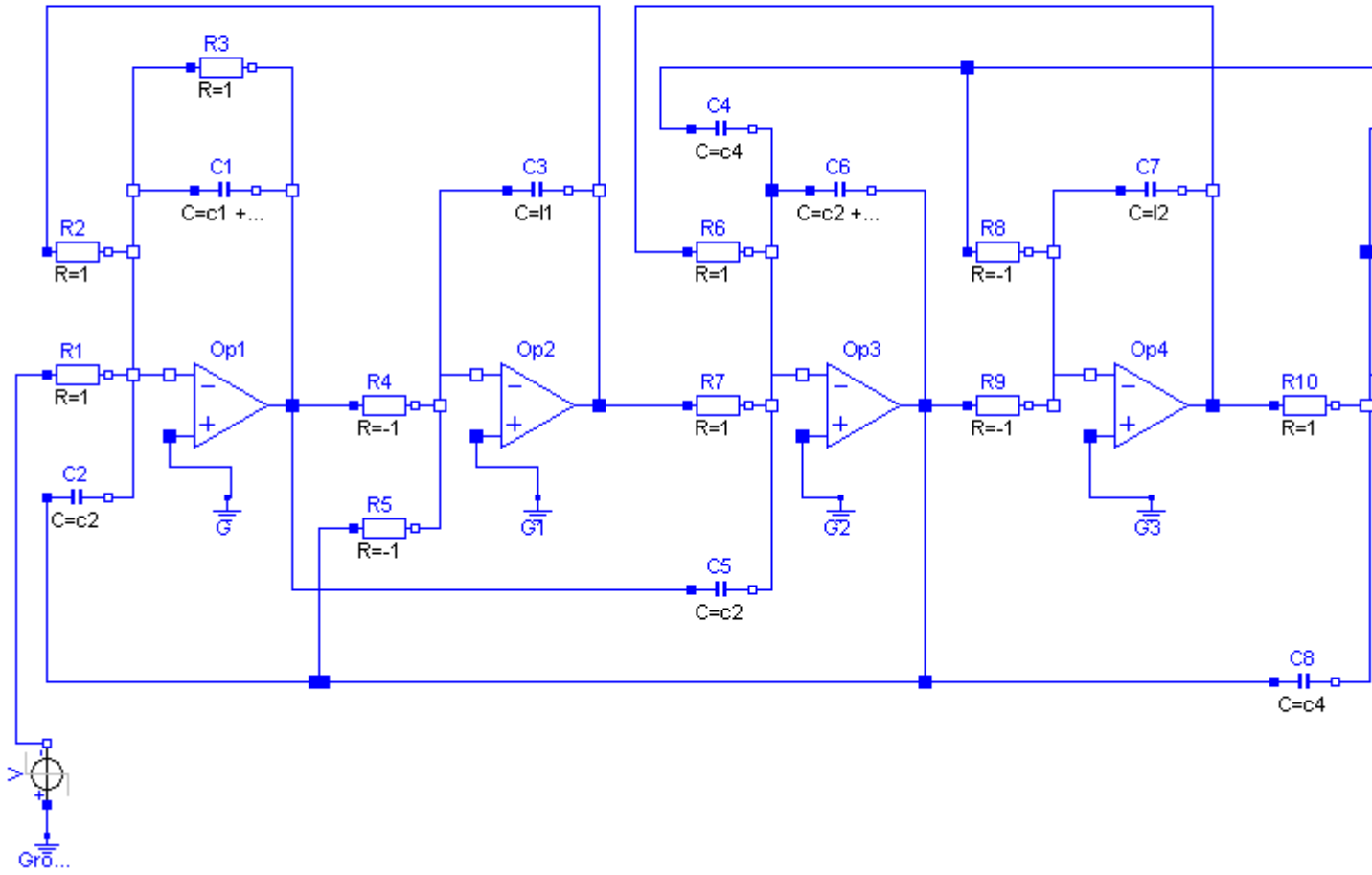
Type	Name	Default	Description
Inductance	l1	1.304	[H]
Inductance	l2	0.8586	[H]
Capacitance	c1	1.072	[F]
Capacitance	c2	1/(1.704992^2*l1)	[F]
Capacitance	c3	1.682	[F]
Capacitance	c4	1/(1.179945^2*l2)	[F]
Capacitance	c5	0.7262	[F]

Modelica.Electrical.Analog.Examples.CauerLowPassOPV

Cauer low pass filter with operational amplifiers



CauerLowPassOPV



Information

The example Cauer Filter is a low-pass-filter of the fifth order. It is realized using an analog network with operational amplifiers. The voltage source V is the input voltage (step), and the OP5.out.v is the filter output voltage. The pulse response is calculated.

This model is identical to the CauerLowPassAnalog example, but inverting. To get the same response as that of the CauerLowPassAnalog example, a negative voltage step is used as input.

The simulation end time should be 60. Please plot both V.v (which is the inverted input voltage) and OP5.p.v (output voltage). Compare this result with the CauerLowPassAnalog result.

During translation some warnings are issued concerning resistor values (Value=-1 not in range[0,1.e+100]). Do not worry about it. The negative values are o.k.

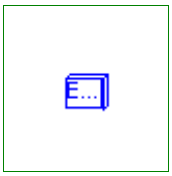
Parameters

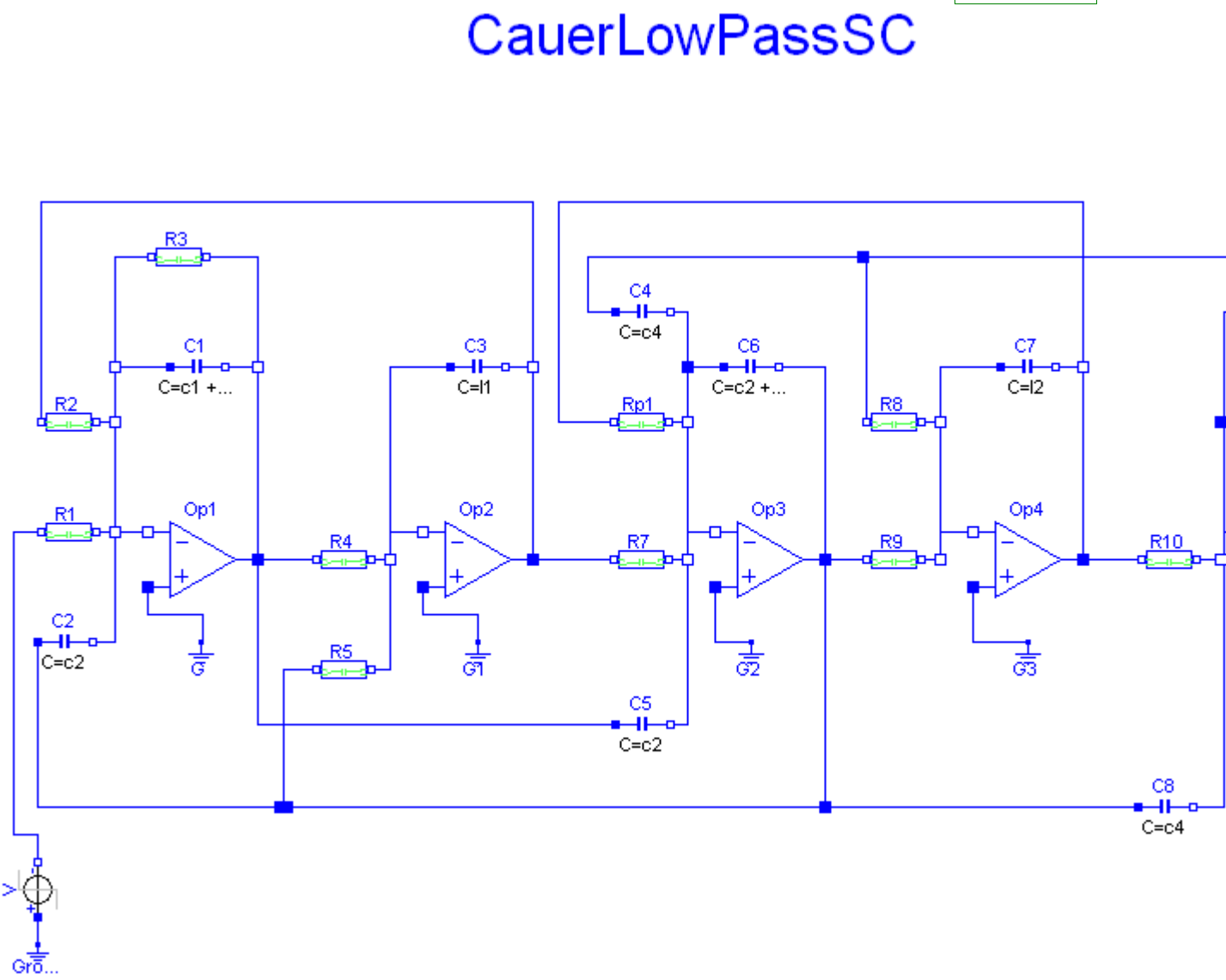
Type	Name	Default	Description
Capacitance	l1	1.304	[F]

Capacitance	l2	0.8586	[F]
Capacitance	c1	1.072	[F]
Capacitance	c2	1/(1.704992^2*l1)	[F]
Capacitance	c3	1.682	[F]
Capacitance	c4	1/(1.179945^2*l2)	[F]
Capacitance	c5	0.7262	[F]

Modelica.Electrical.Analog.Examples.CauerLowPassSC

Cauer low-pass filter with operational amplifiers and switched capacitors





Information

The example CauerLowPassSC is a low-pass-filter of the fifth order. It is realized using an switched-capacitor network with operational amplifiers. The voltage source V is the input voltage (step), and the OP5.out.v is the filter output voltage. The pulse response is calculated.

This model is identical to the CauerLowPassAnalog example, but inverting. To get the same response as that of the CauerLowPassAnalog example, a negative voltage step is used as input.

This model is identical to the CauerLowPassOPV example. But the resistors are realized by switched capacitors. There are two such resistors Rp (of value +1), and Rn (of value -1). In this models the switching clock source is included. In a typical switched capacitor circuit there would be a central clock source.

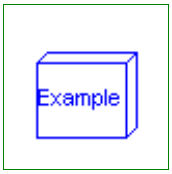
The simulation end time should be 60. Please plot both V.v (which is the inverted input voltage) and OP5.p.v (output voltage). Compare this result with the CauerLowPassAnalog result.

Due to the recharging of the capacitances after switching the performance of simulation is not as good as in the CauerLowPassOPV example.

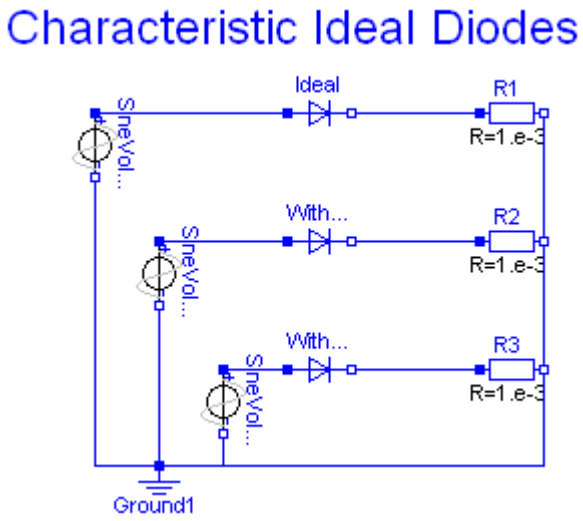
Parameters

Type	Name	Default	Description
Real	l1	1.304	
Real	l2	0.8586	
Real	c1	1.072	
Real	c2	1/(1.704992^2*l1)	
Real	c3	1.682	
Real	c4	1/(1.179945^2*l2)	
Real	c5	0.7262	

Modelica.Electrical.Analog.Examples.CharacteristicIdealDiodes



Characteristic of ideal diodes



Information

Three examples of ideal diodes are shown:

the **totally ideal diode** (Ideal) with all parameters to be zero
the **nearly ideal diode** with *Ron=0.1* and *Goff=0.1*
the nearly ideal but **displaced diode** with *Vknee=5* and *Ron=0.1* and *Goff=0.1*

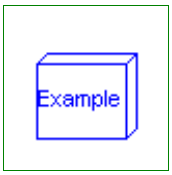
The resistance and conductance are chosen untypically high since the slopes should be seen in the graphics.

Simulate until T=1 s.

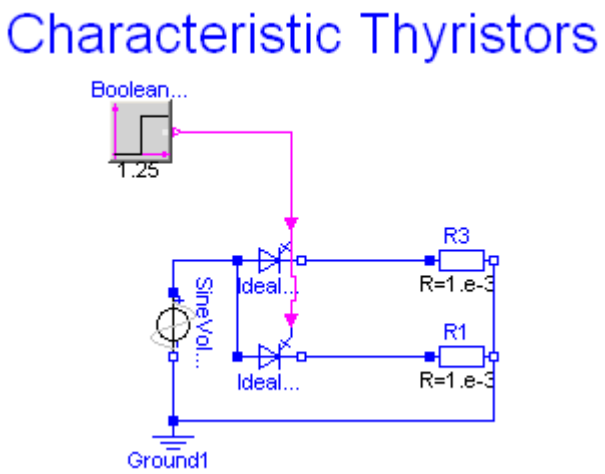
Plot in separate windows:

Ideal.i versus Ideal.v
With_Ron_Goff.i versus With_Ron_Goff.v
With_Ron_Goff_Vknee.i versus With_Ron_Goff_Vknee.v

Modelica.Electrical.Analog.Examples.CharacteristicThyristors



Characteristic of ideal thyristors



Information

Two examples of thyristors are shown:

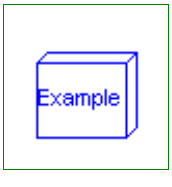
the **ideal thyristor**
and the **ideal GTO thyristor** with *Vknee=5*

Simulate until T=2 s.

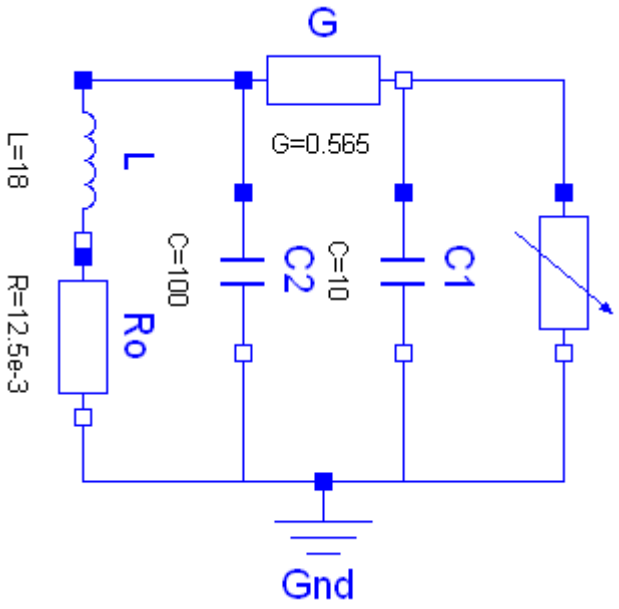
Plot in separate windows:

IdealThyristor1.i and IdealGTOThyristor1.i
IdealThyristor1.v and IdealGTOThyristor1.v

Modelica.Electrical.Analog.Examples.ChuaCircuit



Chua's circuit, ns, V, A



Information

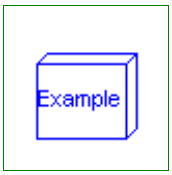
Chua's circuit is the most simple nonlinear circuit which shows chaotic behaviour. The circuit consists of linear basic elements (capacitors, resistor, conductor, inductor), and one nonlinear element, which is called Chua's diode. The chaotic behaviour is simulated.

The simulation end time should be set to 5e4. To get the chaotic behaviour please plot C1.v. Choose C2.v as the independent variable.

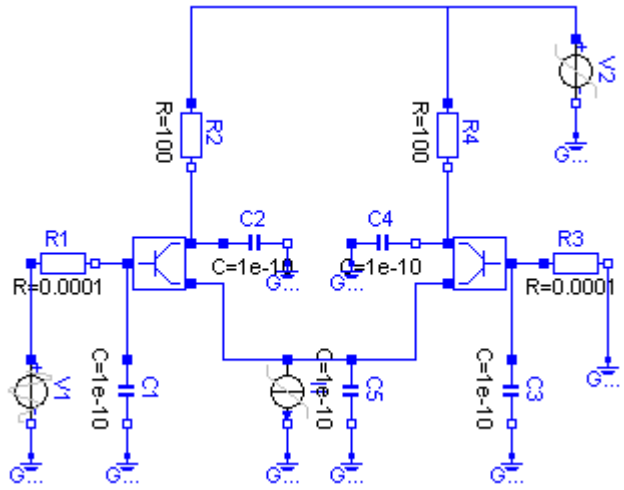
Reference:

Kennedy, M.P.: Three Steps to Chaos - Part I: Evolution. IEEE Transactions on CAS I 40 (1993)10, 640-656

Modelica.Electrical.Analog.Examples.DifferenceAmplifier



Simple NPN transistor amplifier circuit



Information

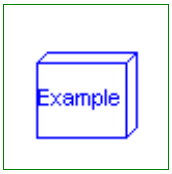
It is a simple NPN transistor amplifier circuit. The voltage difference between R1.p and R3.n is amplified. The output signal is the voltage between R2.n and R4.n. In this example the voltage at V1 is amplified because R3.n is grounded.

The simulation end time should be set to 1e- 8. Please plot the input voltage V1.v, and the output voltages R2.n.v, and R4.n.v.

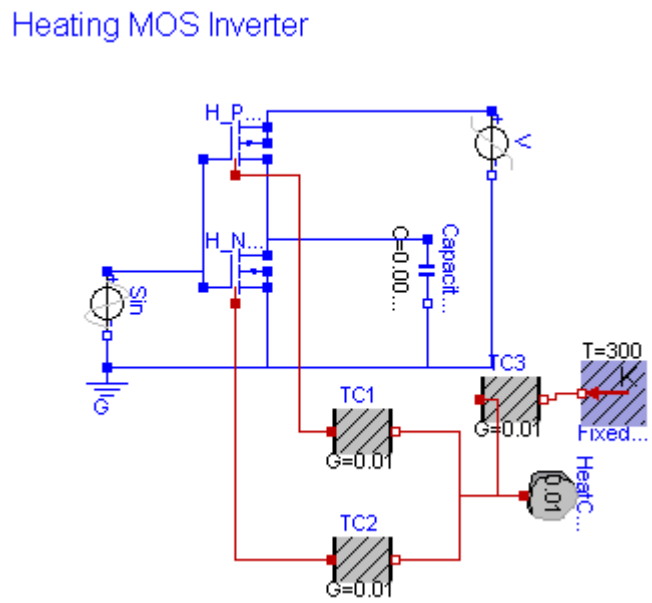
Reference:

Tietze, U.; Schenk, Ch.: Halbleiter-Schaltungstechnik. Springer-Verlag Berlin Heidelberg NewYork 1980, p. 59

Modelica.Electrical.Analog.Examples.HeatingMOSInverter



Heating MOS Inverter



Information

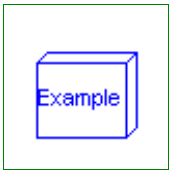
The heating MOS inverter shows a heat flow always if a transistor is leading.

Simulate until T=5 s.

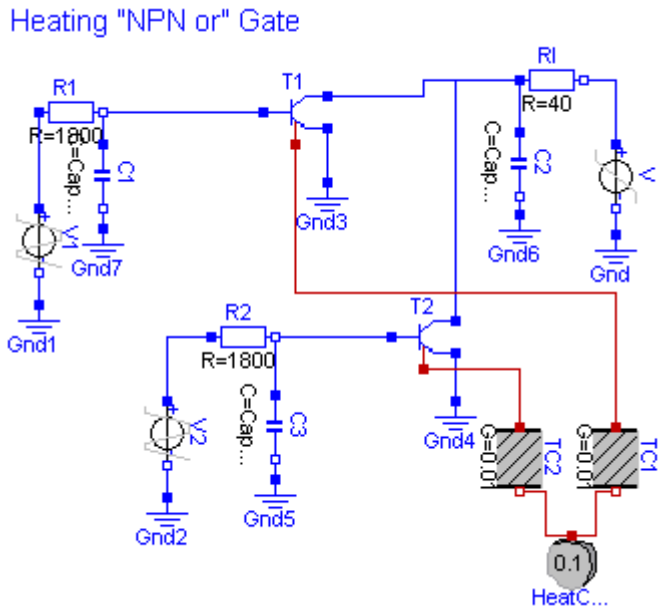
Plot in separate windows:

Sin.p.v and Capacitor1.p.v
HeatCapacitor1.port.T and H_PMOS.heatPort.T and H_NMOS.heatPort.T
H_PMOS.heatPort.Q_flow and H_NMOS.heatPort.Q_flow

Modelica.Electrical.Analog.Examples.HeatingNPN_OrGate



Heating NPN Or Gate



Information

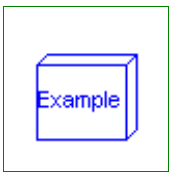
The heating "NPN or" gate shows a heat flow always if a transistor is leading.

Simulate until T=200 s.

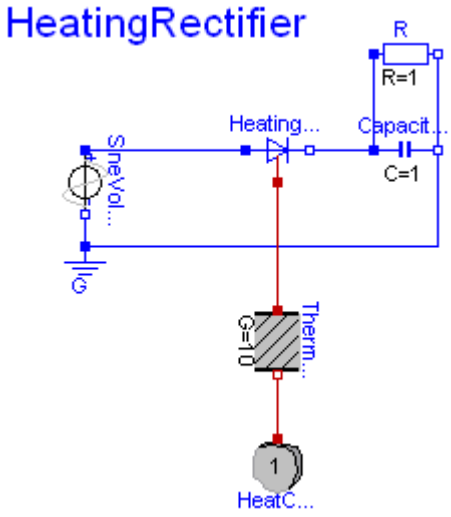
Plot in separate windows:

V1.v and V2.v and C2.v
HeatCapacitor1.port.T and T1.heatPort.T and T2.heatPort.T
T1.heatPort.Q_flow and T2.heatPort.Q_flow

Modelica.Electrical.Analog.Examples.HeatingRectifier



Heating rectifier



Information

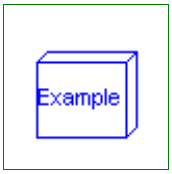
The heating rectifier shows a heat flow always if the electrical capacitor is loaded.

Simulate until T=5 s.

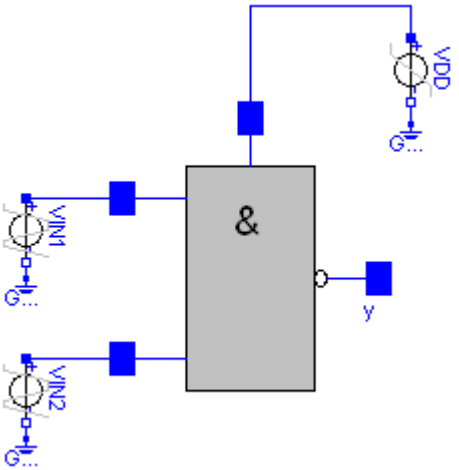
Plot in separate windows:

SineVoltage1.v and Capacitor1.p.v
HeatCapacitor1.port.T and HeatingDiode1.heatPort.T
HeatingDiode1.heatPort.Q_flow

Modelica.Electrical.Analog.Examples.NandGate



CMOS NAND Gate (see Tietze/Schenk, page 157)



Information

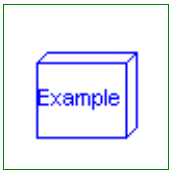
The nand gate is a basic CMOS building block. It consists of four CMOS transistors. The output voltage Nand.y.v is low if and only if the two input voltages at Nand.x1.v and Nand.x2.v are both high. In this way the nand functionality is realized.

The simulation end time should be set to 1e-7. Please plot the input voltages Nand.x1.v, d Nand.x2.v, and the output voltage Nand.y.v.

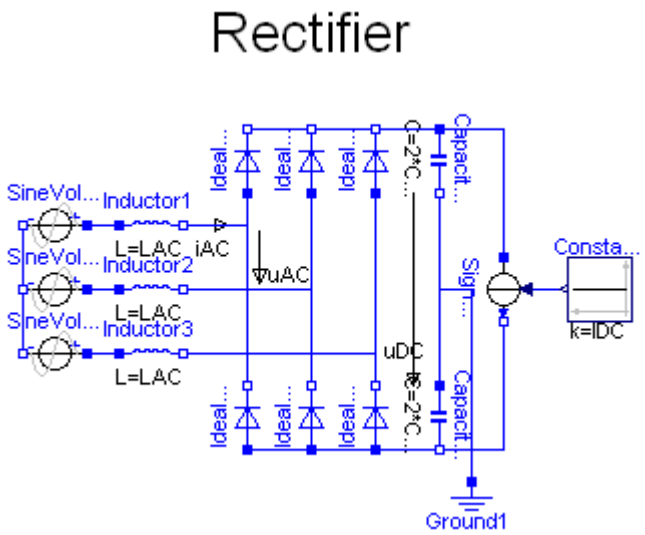
Reference:

Tietze, U.; Schenk, Ch.: Halbleiter-Schaltungstechnik. Springer-Verlag Berlin Heidelberg NewYork 1980, p. 157

Modelica.Electrical.Analog.Examples.Rectifier



B6 diode bridge



Information

The rectifier example shows a B6 diode bridge fed by a three phase sinusoidal voltage, loaded by a DC current.
DC capacitors start at ideal no-load voltage, thus making easier initial transient.

Simulate until T=0.1 s.

Plot in separate windows:

uDC ... DC-voltage
iAC ... AC-currents 1..3
uAC ... AC-voltages 1..3 (distorted)
Try different load currents iDC = 0..approximately 500 A.

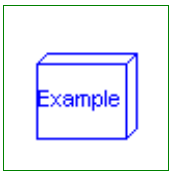
You may watch Losses (of the whole diode bridge) trying different diode parameters.

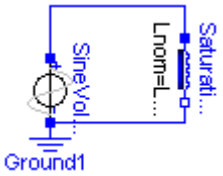
Parameters

Type	Name	Default	Description
Voltage	VAC	400	RMS line-to-line [V]
Frequency	f	50	line frequency [Hz]
Inductance	LAC	60E-6	line inductor [H]
Resistance	Ron	1E-3	diode forward resistance [Ohm]
Conductance	Goff	1E-3	diode backward conductance [S]
Voltage	Vknee	2	diode threshold voltage [V]
Capacitance	CDC	15E-3	DC capacitance [F]
Current	IDC	500	load current [A]

Modelica.Electrical.Analog.Examples.ShowSaturatingInductor

Simple demo to show behaviour of SaturatingInductor component



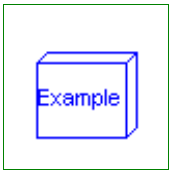


Information

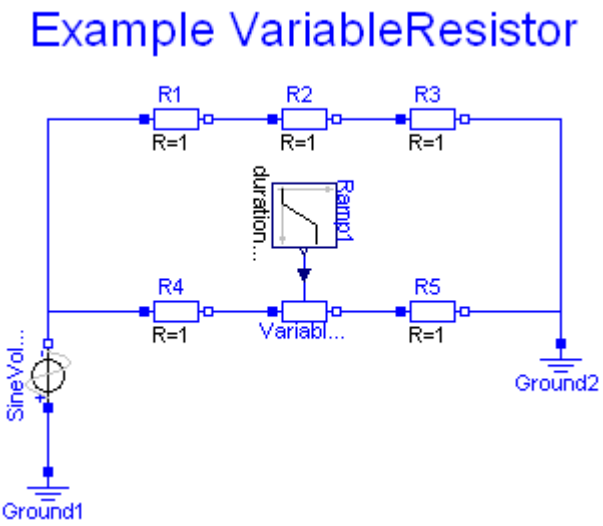
Parameters

Type	Name	Default	Description
Inductance	Lzer	2	[H]
Inductance	Lnom	1	[H]
Current	Inom	1	[A]
Inductance	Linf	0.5	[H]
Voltage	U	1.25	[V]
Frequency	f	1/(2*Modelica.Constants.pi)	[Hz]
Angle	phase	Modelica.Constants.pi/2	[rad]

Modelica.Electrical.Analog.Examples.ShowVariableResistor



Simple demo of a VariableResistor model



Information

It is a simple test circuit for the VariableResistor. The VariableResistor sould be compared with R2.

Simulate until T=1 s.

Modelica.Electrical.Analog.Examples.Utilities

Utility components used by package Examples

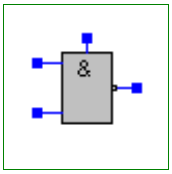
Information

This package contains utility components used by package Examples.

Package Content

Name	Description
Nand	CMOS NAND Gate (see Tietze/Schenk, page 157)
NonlinearResistor	Chua's resistor
RealSwitch	
Transistor	

Modelica.Electrical.Analog.Examples.Utilities.Nand



CMOS NAND Gate (see Tietze/Schenk, page 157)

Information

The nand gate is a basic CMOS building block. It consists of four CMOS transistors.

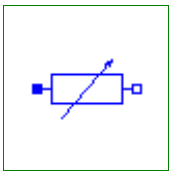
Reference:

Tietze, U.; Schenk, Ch.: Halbleiter-Schaltungstechnik. Springer-Verlag Berlin Heidelberg NewYork 1980, p. 157

Connectors

Type	Name	Description
Pin	x1	
Pin	x2	
Pin	Vdd	
Pin	y	

Modelica.Electrical.Analog.Examples.Utilities.NonlinearResistor



Chua's resistor

Information

Parameters

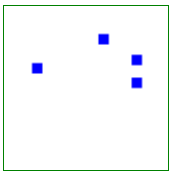
Type	Name	Default	Description
Conductance	Ga		[S]
Conductance	Gb		[S]
Voltage	Ve		[V]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop

		v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Examples.Utilities.RealSwitch

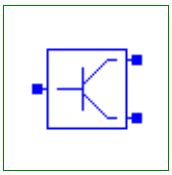


Information

Connectors

Type	Name	Description
Pin	p	
Pin	n1	
Pin	n2	
Pin	control	

Modelica.Electrical.Analog.Examples.Utilities.Transistor



Information

Connectors

Type	Name	Description
Pin	c	
Pin	b	
Pin	e	

Modelica.Electrical.Analog.Basic

Basic electrical components such as resistor, capacitor, transformer

Information

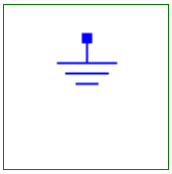
This package contains basic analog electrical components.

Package Content

Name	Description
Ground	Ground node
Resistor	Ideal linear electrical resistor
HeatingResistor	Temperature dependent electrical resistor
Conductor	Ideal linear electrical conductor
Capacitor	Ideal linear electrical capacitor
Inductor	Ideal linear electrical inductor
SaturatingInductor	Simple model of an inductor with saturation

Transformer	Transformer with two ports
Gyrator	Gyrator
EMF	Electromotoric force (electric/mechanic transformer)
VCV	Linear voltage-controlled voltage source
VCC	Linear voltage-controlled current source
CCV	Linear current-controlled voltage source
CCC	Linear current-controlled current source
OpAmp	Simple nonideal model of an OpAmp with limitation
VariableResistor	Ideal linear electrical resistor with variable resistance
VariableConductor	Ideal linear electrical conductor with variable conductance
VariableCapacitor	Ideal linear electrical capacitor with variable capacitance
VariableInductor	Ideal linear electrical inductor with variable inductance

Modelica.Electrical.Analog.Basic.Ground



Ground node

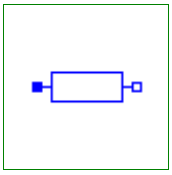
Information

Ground of an electrical circuit. The potential at the ground node is zero. Every electrical circuit has to contain at least one ground object.

Connectors

Type	Name	Description
Pin	p	

Modelica.Electrical.Analog.Basic.Resistor



Ideal linear electrical resistor

Information

The linear resistor connects the branch voltage *v* with the branch current *i* by *i*R* = *v*. The Resistance *R* is allowed to be positive, zero, or negative.

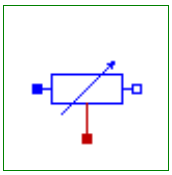
Parameters

Type	Name	Default	Description
Resistance	R	1	Resistance [Ohm]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Basic.HeatingResistor



Temperature dependent electrical resistor

Information

This is a model for an electrical resistor where the generated heat is dissipated to the environment via connector **heatPort** and where the resistance R is temperature dependent according to the following equation:

```
R = R_ref*(1 + alpha*(heatPort.T - T_ref))
```

alpha is the **temperature coefficient of resistance**, which is often abbreviated as **TCR**. In resistor catalogues, it is usually defined as **X [ppm/K]** (parts per million, similarly to per centage) meaning **X*1.e-6 [1/K]**. Resistors are available for 1 .. 7000 ppm/K, i.e., alpha = 1e-6 .. 7e-3 1/K;

When connector **heatPort** is **not** connected, the temperature dependent behaviour is switched off by setting heatPort.T = T_ref. Additionally, the equation `heatPort.Q_flow = 0` is implicitly present due to a special rule in Modelica that flow variables of not connected connectors are set to zero.

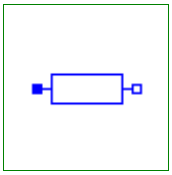
Parameters

Type	Name	Default	Description
Resistance	R_ref	1	Resistance at temperature T_ref [Ohm]
Temperature	T_ref	300	Reference temperature [K]
Real	alpha	0	Temperature coefficient of resistance [1/K]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
HeatPort_a	heatPort	

Modelica.Electrical.Analog.Basic.Conductor



Ideal linear electrical conductor

Information

The linear conductor connects the branch voltage *v* with the branch current *i* by *i* = *v***G*. The Conductance *G* is allowed to be positive, zero, or negative.

Parameters

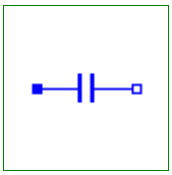
Type	Name	Default	Description
Conductance	G	1	Conductance [S]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop

		v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Basic.Capacitor



Ideal linear electrical capacitor

Information

The linear capacitor connects the branch voltage *v* with the branch current *i* by *i = C * dv/dt*. The Capacitance *C* is allowed to be positive, zero, or negative.

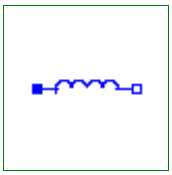
Parameters

Type	Name	Default	Description
Capacitance	C	1	Capacitance [F]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Basic.Inductor



Ideal linear electrical inductor

Information

The linear inductor connects the branch voltage *v* with the branch current *i* by *v = L * di/dt*. The Inductance *L* is allowed to be positive, zero, or negative.

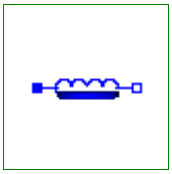
Parameters

Type	Name	Default	Description
Inductance	L	1	Inductance [H]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Basic.SaturatingInductor



Simple model of an inductor with saturation

Information

This model approximates the behaviour of an inductor with the influence of saturation, i.e. the value of the inductance depends on the current flowing through the inductor. The inductance decreases as current increases.
The parameters are:

- Inom...nominal current
- Lnom...nominal inductance at nominal current
- Lzer...inductance near current = 0; Lzer has to be greater than Lnom
- Linf...inductance at large currents; Linf has to be less than Lnom

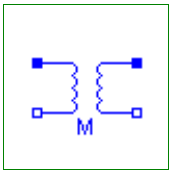
Parameters

Type	Name	Default	Description
Current	Inom	1	Nominal current [A]
Inductance	Lnom	1	Nominal inductance at Nominal current [H]
Inductance	Lzer	2*Lnom	Inductance near current=0 [H]
Inductance	Linf	Lnom/2	Inductance at large currents [H]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Basic.Transformer



Transformer with two ports

Information

The transformer is a two port. The left port voltage *v1*, left port current *i1*, right port voltage *v2* and right port current *i2* are connected by the following relation:

```
| v1 |          | L1   M  |   | i1' |
|    |    =     |         |   |     |
| v2 |          | M    L2 |   | i2' |
```

L1, *L2*, and *M* are the primary, secondary, and coupling inductances respectively.

Parameters

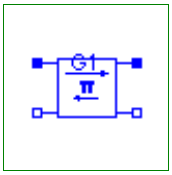
Type	Name	Default	Description
Inductance	L1	1	Primary inductance [H]
Inductance	L2	1	Secondary inductance [H]
Inductance	M	1	Coupling inductance [H]

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port

PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Basic.Gyrator



Gyrator

Information

A gyrator is a two-port element defined by the following equations:

```
i1 =  G2 * v2
i2 = -G1 * v1
```

where the constants *G1*, *G2* are called the gyration conductance.

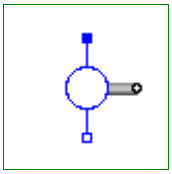
Parameters

Type	Name	Default	Description
Conductance	G1	1	Gyration conductance [S]
Conductance	G2	1	Gyration conductance [S]

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Basic.EMF



Electromotoric force (electric/mechanic transformer)

Information

EMF transforms electrical energy into rotational mechanical energy. It is used as basic building block of an electrical motor. The mechanical connector flange_b can be connected to elements of the Modelica.Mechanics.Rotational library. flange_b.tau is the cut-torque, flange_b.phi is the angle at the rotational connection.

Parameters

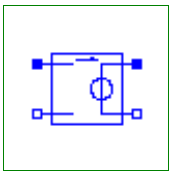
Type	Name	Default	Description
Real	k	1	Transformation coefficient [N.m/A]

Connectors

Type	Name	Description
PositivePin	p	

NegativePin	n	
Flange_b	flange_b	

Modelica.Electrical.Analog.Basic.VCV



Linear voltage-controlled voltage source

Information

The linear voltage-controlled voltage source is a TwoPort. The right port voltage v2 is controlled by the left port voltage v1 via

```
v2 = v1 * gain.
```

The left port current is zero. Any voltage gain can be chosen.

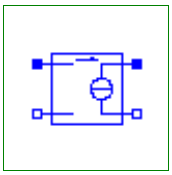
Parameters

Type	Name	Default	Description
Real	gain	1	Voltage gain

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Basic.VCC



Linear voltage-controlled current source

Information

The linear voltage-controlled current source is a TwoPort. The right port current i2 is controlled by the left port voltage v1 via

```
i2 = v1 * transConductance.
```

The left port current is zero. Any transConductance can be chosen.

Parameters

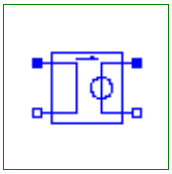
Type	Name	Default	Description
Conductance	transConductance	1	Transconductance [S]

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)

NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Basic.CCV



Linear current-controlled voltage source

Information

The linear current-controlled voltage source is a TwoPort. The right port voltage v2 is controlled by the left port current i1 via

```
v2 = i1 * transResistance.
```

The left port voltage is zero. Any transResistance can be chosen.

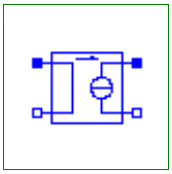
Parameters

Type	Name	Default	Description
Resistance	transResistance	1	Transresistance [Ohm]

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Basic.CCC



Linear current-controlled current source

Information

The linear current-controlled current source is a TwoPort. The right port current i2 is controlled by the left port current i1 via

```
i2 = i1 * gain.
```

The left port voltage is zero. Any current gain can be chosen.

Parameters

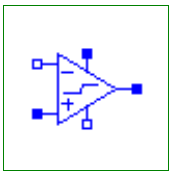
Type	Name	Default	Description
Real	gain	1	Current gain

Connectors

Type	Name	Description

PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Basic.OpAmp



Simple nonideal model of an OpAmp with limitation

Information

The OpAmp is a simle nonideal model with a smooth out.v = f(vin) characteristic, where "vin = in_p.v - in_n.v". The characteristic is limited by VMax.v and VMin.v. Its slope at vin=0 is the parameter Slope, which must be positive. (Therefore, the absolute value of Slope is taken into calculation.)

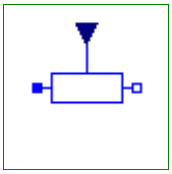
Parameters

Type	Name	Default	Description
Real	Slope	1	Slope of the out.v/vin characteristic at vin=0

Connectors

Type	Name	Description
PositivePin	in_p	Positive pin of the input port
NegativePin	in_n	Negative pin of the input port
PositivePin	out	Output pin
PositivePin	VMax	Positive output voltage limitation
NegativePin	VMin	Negative output voltage limitation

Modelica.Electrical.Analog.Basic.VariableResistor



Ideal linear electrical resistor with variable resistance

Information

The linear resistor connects the branch voltage *v* with the branch current *i* by

i*R = v

The Resistance *R* is given as input signal.

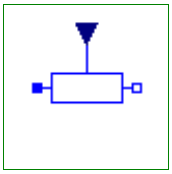
Attention!!!
It is recommended that the R signal should not cross the zero value. Otherwise depending on the surrounding circuit the probability of singularities is high.

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

input RealInput	R	

Modelica.Electrical.Analog.Basic.VariableConductor



Ideal linear electrical conductor with variable conductance

Information

The linear conductor connects the branch voltage *v* with the branch current *i* by

i = G*v

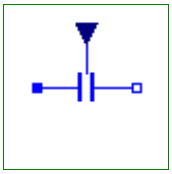
The Conductance *G* is given as input signal.

Attention!!!
It is recommended that the G signal should not cross the zero value. Otherwise depending on the surrounding circuit the probability of singularities is high.

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
input RealInput	G	

Modelica.Electrical.Analog.Basic.VariableCapacitor



Ideal linear electrical capacitor with variable capacitance

Information

The linear capacitor connects the branch voltage *v* with the branch current *i* by

i = dQ/dt with ***Q = C * v*** .

The capacitance *C* is given as input signal.

It is required that C ≥ 0, otherwise an assertion is raised. To avoid a variable index system, C = Cmin, if 0 ≤ C < Cmin, where Cmin is a parameter with default value Modelica.Constants.eps.

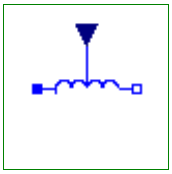
Parameters

Type	Name	Default	Description
Capacitance	Cmin	Modelica.Constants.eps	[F]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
input RealInput	C	

Modelica.Electrical.Analog.Basic.VariableInductor



Ideal linear electrical inductor with variable inductance

Information

The linear inductor connects the branch voltage *v* with the branch current *i* by

v = d Psi/dt with ***Psi = L * i*** .

The inductance *L* is as input signal.

It is required that L ≥ 0, otherwise an assertion is raised. To avoid a variable index system, L = Lmin, if 0 ≤ L < Lmin, where Lmin is a parameter with default value Modelica.Constants.eps.

Parameters

Type	Name	Default	Description
Inductance	Lmin	Modelica.Constants.eps	[H]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
input RealInput	L	

Modelica.Electrical.Analog.Ideal

Ideal electrical elements such as switches, diode, transformer, operational amplifier

Information

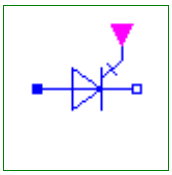
This package contains electrical components with idealized behaviour:

Package Content

Name	Description
IdealThyristor	Ideal thyristor
IdealGTOThyristor	Ideal GTO thyristor
IdealCommutingSwitch	Ideal commuting switch
IdealIntermediateSwitch	Ideal intermediate switch
ControlledIdealCommutingSwitch	Controlled ideal commuting switch
ControlledIdealIntermediateSwitch	Controlled ideal intermediate switch
IdealOpAmp	Ideal operational amplifier (norator-nullator pair)
IdealOpAmp3Pin	Ideal operational amplifier (norator-nullator pair), but 3 pins
IdealOpAmpLimited	Ideal operational amplifier with limitation
IdealDiode	Ideal diode
IdealTransformer	Ideal electrical transformer

IdealGyrator	Ideal gyrator
Idle	Idle branch
Short	Short cut branch
IdealOpeningSwitch	Ideal electrical opener
IdealClosingSwitch	Ideal electrical closer
ControlledIdealOpeningSwitch	Controlled ideal electrical opener
ControlledIdealClosingSwitch	Controlled ideal electrical closer

Modelica.Electrical.Analog.Ideal.IdealThyristor



Ideal thyristor

Information

This is an ideal thyristor model which is

open (off), if the voltage drop is less than 0 or fire is false
closed (on), if the voltage drop is greater or equal 0 and fire is true.

This is the behaviour if all parameters are exactly zero.

Note, there are circuits, where this ideal description with zero resistance and zero cinductance is not possible. In order to prevent singularities during switching, the opened thyristor has a small conductance *Goff* and the closed thyristor has a low resistance *Ron* which is default.

The parameter *Vknee* which is the forward threshold voltage, allows to displace the knee point along the *Goff*-characteristic until *v* = *Vknee*.

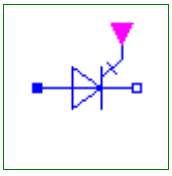
Parameters

Type	Name	Default	Description
Resistance	Ron	1.E-5	Closed thyristor resistance [Ohm]
Conductance	Goff	1.E-5	Opened thyristor conductance [S]
Voltage	Vknee	0	Forward threshold voltage [V]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
input BooleanInput	fire	

Modelica.Electrical.Analog.Ideal.IdealGTOThyristor



Ideal GTO thyristor

Information

This is an ideal GTO thyristor model which is

open (off), if the voltage drop is less than 0 or fire is false
closed (on), if the voltage drop is greater or equal 0 and fire is true.

This is the behaviour if all parameters are exactly zero.

Note, there are circuits, where this ideal description with zero resistance and zero cinductance is not possible. In order to prevent singularities during switching, the opened thyristor has a small conductance *Goff* and the closed thyristor has a low resistance *Ron* which is default.

The parameter *Vknee* which is the forward threshold voltage, allows to displace the knee point along the *Goff*-characteristic until *v* = *Vknee*.

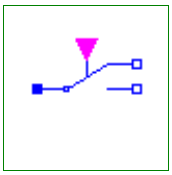
Parameters

Type	Name	Default	Description
Resistance	Ron	1.E-5	Closed thyristor resistance [Ohm]
Conductance	Goff	1.E-5	Opened thyristor conductance [S]
Voltage	Vknee	0	Forward threshold voltage [V]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
input BooleanInput	fire	

Modelica.Electrical.Analog.Ideal.IdealCommutingSwitch



Ideal commuting switch

Information

The commuting switch has a positive pin p and two negative pins n1 and n2. The switching behaviour is controlled by the inpug signal control. If control is true, the pin p is connected with the negative pin n2. Otherwise, the pin p is connected to the negative pin n1.

In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron. The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.

Parameters

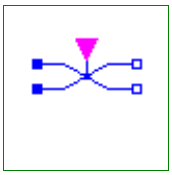
Type	Name	Default	Description
Resistance	Ron	1.E-5	Closed switch resistance [Ohm]
Conductance	Goff	1.E-5	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePin	p	
NegativePin	n2	

NegativePin	n1	
input BooleanInput	control	true => p--n2 connected, false => p--n1 connected

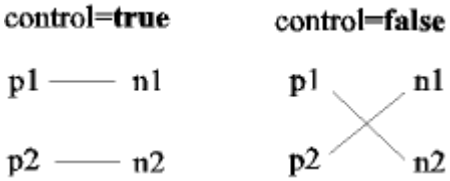
Modelica.Electrical.Analog.Ideal.IdealIntermediateSwitch



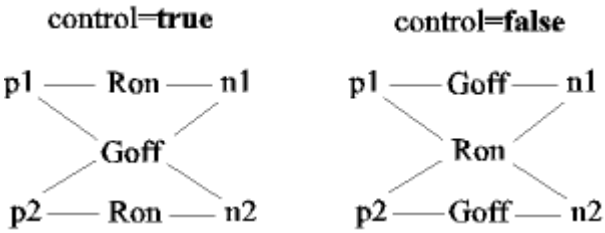
Ideal intermediate switch

Information

The intermediate switch has four switching contact pins p1, p2, n1, and n2. The switching behaviour is controlled by the input signal control. If control is true, the pin p1 is connected to pin n2, and the pin p2 is connected to the pin n2. Otherwise, the pin p1 is connected to n1, and p2 is connected to n2.



In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron.



The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.

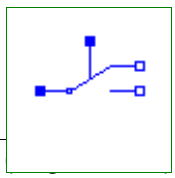
Parameters

Type	Name	Default	Description
Resistance	Ron	1.E-5	Closed switch resistance [Ohm]
Conductance	Goff	1.E-5	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePin	p1	
PositivePin	p2	
NegativePin	n1	
NegativePin	n2	
input BooleanInput	control	true => p1--n2, p2--n1 connected, otherwise p1--n1, p2--n2 connected

Modelica.Electrical.Analog.Ideal.ControlledIdealCommutingSwitch



Controlled ideal commuting switch

Information

The commuting switch has a positive pin p and two negative pins n1 and n2. The switching behaviour is controlled by the control pin. If its voltage exceeds the value of the parameter level, the pin p is connected with the negative pin n2. Otherwise, the pin p is connected the negative pin n1.

In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron. The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.

Parameters

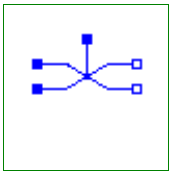
Type	Name	Default	Description
Voltage	level	0.5	Switch level [V]
Resistance	Ron	1.E-5	Closed switch resistance [Ohm]
Conductance	Goff	1.E-5	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePin	p	
NegativePin	n2	
NegativePin	n1	
Pin	control	Control pin: if control.v > level p--n2 connected, otherwise p--n1 connected

---

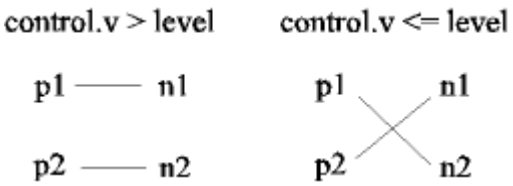
Modelica.Electrical.Analog.Ideal.ControlledIdealIntermediateSwitch



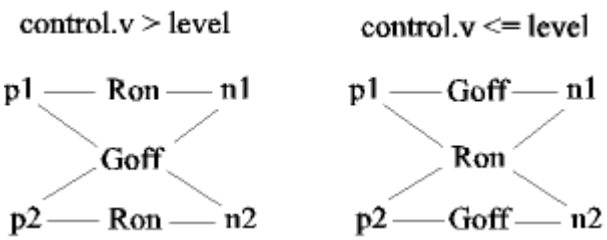
Controlled ideal intermediate switch

Information

The intermediate switch has four switching contact pins p1, p2, n1, and n2. The switching behaviour is controlled by the control pin. If its voltage exceeds the value of the parameter level, the pin p1 is connected to pin n2, and the pin p2 is connected to the pin n2. Otherwise, the pin p1 is connected to n1, and p2 is connected to n2.



In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron.



The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the

conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.

Parameters

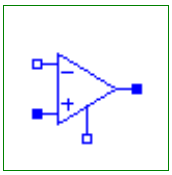
Type	Name	Default	Description
Voltage	level	0.5	Switch level [V]
Resistance	Ron	1.E-5	Closed switch resistance [Ohm]
Conductance	Goff	1.E-5	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePin	p1	
PositivePin	p2	
NegativePin	n1	
NegativePin	n2	
Pin	control	Control pin: if control.v > level p1--n2, p2--n1 connected, otherwise p1--n1, p2--n2 connected

---

Modelica.Electrical.Analog.Ideal.IdealOpAmp



Ideal operational amplifier (norator-nullator pair)

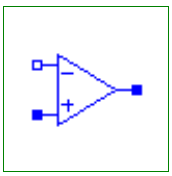
Information

The ideal OpAmp is a two-port. The left port is fixed to *v1=0* and *i1=0* (nullator). At the right port both any voltage *v2* and any current *i2* are possible (norator).

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Ideal.IdealOpAmp3Pin



Ideal operational amplifier (norator-nullator pair), but 3 pins

Information

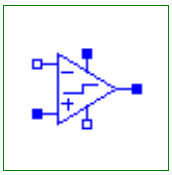
The ideal OpAmp with three pins is of exactly the same behaviour as the ideal OpAmp with four pins. Only the negative output pin is left out. Both the input voltage and current are fixed to zero (nullator). At the output pin both any voltage *v2* and any current *i2* are possible.

Connectors

Type	Name	Description
PositivePin	in_p	Positive pin of the input port

NegativePin	in_n	Negative pin of the input port
PositivePin	out	Output pin

Modelica.Electrical.Analog.Ideal.IdealOpAmpLimited



Ideal operational amplifier with limitation

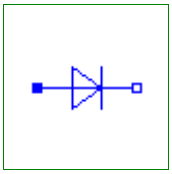
Information

The ideal OpAmp with limitation behaves like an ideal OpAmp without limitation, if the output voltage is within the limits (VMin < out.v < VMax). In this case the input voltage vin=in_p.v - in_n.v is zero. If the input voltage is vin < 0, the output voltage is out.v = VMin. If the input voltage is vin > 0, the output voltage is out.v = VMax.

Connectors

Type	Name	Description
PositivePin	in_p	Positive pin of the input port
NegativePin	in_n	Negative pin of the input port
PositivePin	out	Output pin
PositivePin	VMax	Positive output voltage limitation
NegativePin	VMin	Negative output voltage limitation

Modelica.Electrical.Analog.Ideal.IdealDiode



Ideal diode

Information

This is an ideal switch which is

open (off), if it is reversed biased (voltage drop less than 0)
closed (on), if it is conducting (current > 0).

This is the behaviour if all parameters are exactly zero.

Note, there are circuits, where this ideal description with zero resistance and zero cinductance is not possible. In order to prevent singularities during switching, the opened diode has a small conductance *Gon* and the closed diode has a low resistance *Roff* which is default.

The parameter *Vknee* which is the forward threshold voltage, allows to displace the knee point along the *Gon*-characteristic until *v* = *Vknee*.

Parameters

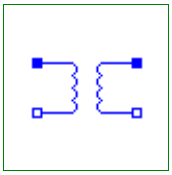
Type	Name	Default	Description
Resistance	Ron	1.E-5	Forward state-on differential resistance (closed diode resistance) [Ohm]
Conductance	Goff	1.E-5	Backward state-off conductance (opened diode conductance) [S]
Voltage	Vknee	0	Forward threshold voltage [V]

Connectors

Type	Name	Description

PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Ideal.IdealTransformer



Ideal electrical transformer

Information

The ideal transformer is an ideal two-port resistive circuit element which is characterized by the following two equations:

```
v1 =  n * v2
i2 = -n * i1
```

where *n* is a real number called the turns ratio.

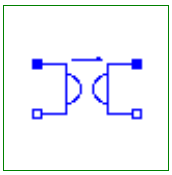
Parameters

Type	Name	Default	Description
Real	n	1	Turns ratio

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Ideal.IdealGyrator



Ideal gyrator

Information

A gyrator is an ideal two-port element defined by the following equations:

```
i1 =  G * v2
i2 = -G * v1
```

where the constant *G* is called the gyration conductance.

Parameters

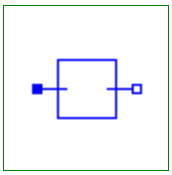
Type	Name	Default	Description
Conductance	G	1	Gyration conductance [S]

Connectors

Type	Name	Description

PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Ideal.Idle



Idle branch

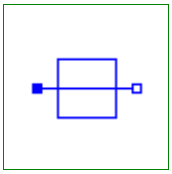
Information

The model Idle is a simple idle running branch.

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Ideal.Short



Short cut branch

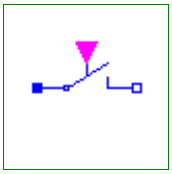
Information

The model Short is a simple short cut branch.

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Ideal.IdealOpeningSwitch



Ideal electrical opener

Information

The ideal opening switch has a positive pin p and a negative pin n. The switching behaviour is controlled by the input signal control. If control is true, pin p is not connected with negative pin n. Otherwise, pin p is connected with negative pin n.

In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron. The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.

Parameters

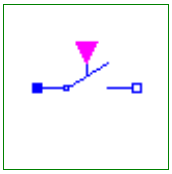
Type	Name	Default	Description
Resistance	Ron	1.E-5	Closed switch resistance [Ohm]
Conductance	Goff	1.E-5	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
input BooleanInput	control	true => switch open, false => p--n connected

---

Modelica.Electrical.Analog.Ideal.IdealClosingSwitch



Ideal electrical closer

Information

The ideal closing switch has a positive pin p and a negative pin n. The switching behaviour is controlled by input signal control. If control is true, pin p is connected with negative pin n. Otherwise, pin p is not connected with negative pin n.

In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron. The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.

Parameters

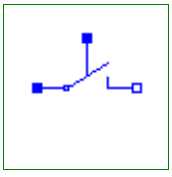
Type	Name	Default	Description
Resistance	Ron	1.E-5	Closed switch resistance [Ohm]
Conductance	Goff	1.E-5	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
input BooleanInput	control	true => p--n connected, false => switch open

---

Modelica.Electrical.Analog.Ideal.ControlledIdealOpeningSwitch



Controlled ideal electrical opener

Information

The ideal switch has a positive pin p and a negative pin n. The switching behaviour is controlled by the control pin. If its voltage exceeds the voltage of the parameter level, pin p is not connected with negative pin n. Otherwise, pin p is connected with negative pin n.

In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron. The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.

Parameters

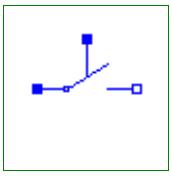
Type	Name	Default	Description
Voltage	level	0.5	Switch level [V]
Resistance	Ron	1.E-5	Closed switch resistance [Ohm]
Conductance	Goff	1.E-5	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePin	p	
NegativePin	n	
Pin	control	Control pin: control.v > level switch open, otherwise p--n connected

---

Modelica.Electrical.Analog.Ideal.ControlledIdealClosingSwitch



Controlled ideal electrical closer

Information

The closing ideal switch has a positive pin p and a negative pin n. The switching behaviour is controlled by the control pin. If its voltage exceeds the voltage of the parameter level, pin p is connected with negative pin n. Otherwise, pin p is not connected with negative pin n.

In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron. The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.

Parameters

Type	Name	Default	Description
Voltage	level	0.5	Switch level [V]
Resistance	Ron	1.E-5	Closed switch resistance [Ohm]
Conductance	Goff	1.E-5	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePin	p	
NegativePin	n	
Pin	control	Control pin: control.v > level switch closed, otherwise switch open

---

Modelica.Electrical.Analog.Interfaces

Connectors and partial models for Analog electrical components

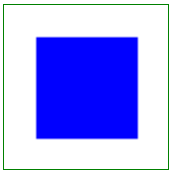
Information

This package contains connectors and interfaces (partial models) for analog electrical components.

Package Content

Name	Description
Pin	Pin of an electrical component
PositivePin	Positive pin of an electric component
NegativePin	Negative pin of an electric component
TwoPin	Component with one electrical port
OnePort	Component with two electrical pins p and n and current i from p to n
TwoPort	Component with two electrical ports, including current
AbsoluteSensor	Base class to measure the absolute value of a pin variable
RelativeSensor	Base class to measure a relative variable between two pins
VoltageSource	Interface for voltage sources
CurrentSource	Interface for current sources

Modelica.Electrical.Analog.Interfaces.Pin

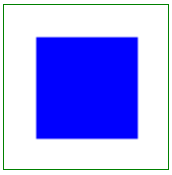


Pin of an electrical component

Contents

Type	Name	Description
Voltage	v	Potential at the pin [V]
flow Current	i	Current flowing into the pin [A]

Modelica.Electrical.Analog.Interfaces.PositivePin



Positive pin of an electric component

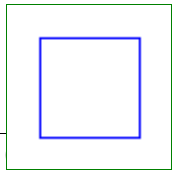
Information

Connectors PositivePin and NegativePin are nearly identical. The only difference is that the icons are different in order to identify more easily the pins of a component. Usually, connector PositivePin is used for the positive and connector NegativePin for the negative pin of an electrical component.

Contents

Type	Name	Description
Voltage	v	Potential at the pin [V]
flow Current	i	Current flowing into the pin [A]

Modelica.Electrical.Analog.Interfaces.NegativePin



Negative pin of an electric component

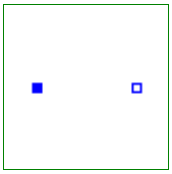
Information

Connectors PositivePin and NegativePin are nearly identical. The only difference is that the icons are different in order to identify more easily the pins of a component. Usually, connector PositivePin is used for the positive and connector NegativePin for the negative pin of an electrical component.

Contents

Type	Name	Description
Voltage	v	Potential at the pin [V]
flow Current	i	Current flowing into the pin [A]

Modelica.Electrical.Analog.Interfaces.TwoPin

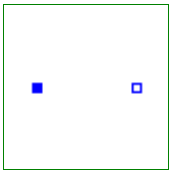


Component with one electrical port

Connectors

Type	Name	Description
PositivePin	p	Positive pin Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Interfaces.OnePort



Component with two electrical pins p and n and current i from p to n

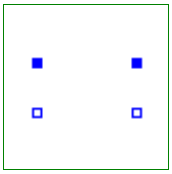
Information

Superclass of elements which have **two** electrical pins: the positive pin connector *p*, and the negative pin connector *n*. It is assumed that the current flowing into pin p is identical to the current flowing out of pin n. This current is provided explicitly as current i.

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Interfaces.TwoPort



Component with two electrical ports, including current

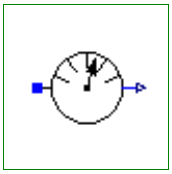
Information

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)

NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Interfaces.AbsoluteSensor

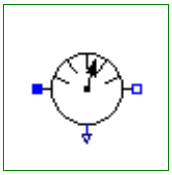


Base class to measure the absolute value of a pin variable

Connectors

Type	Name	Description
PositivePin	p	Pin to be measured
output RealOutput	y	Measured quantity as Real output signal

Modelica.Electrical.Analog.Interfaces.RelativeSensor

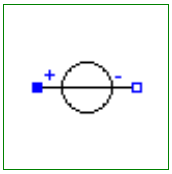


Base class to measure a relative variable between two pins

Connectors

Type	Name	Description
PositivePin	p	Positive pin
NegativePin	n	Negative pin
output RealOutput	y	Measured quantity as Real output signal

Modelica.Electrical.Analog.Interfaces.VoltageSource



Interface for voltage sources

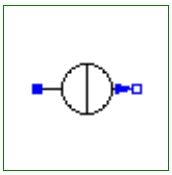
Parameters

Type	Name	Default	Description
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]
SignalSource	signalSource	redeclare Modelica.Blocks.In...	

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Interfaces.CurrentSource



Interface for current sources

Parameters

Type	Name	Default	Description
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]
SignalSource	signalSource	redeclare Modelica.Blocks.In...	

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Lines

Lossy and lossless segmented transmission lines, and LC distributed line models

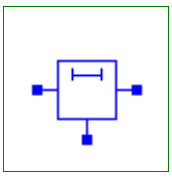
Information

This package contains lossy and lossless segmented transmission lines, and LC distributed line models.

Package Content

Name	Description
OLine	Lossy Transmission Line
ULine	Lossy RC Line
TLine1	Lossless transmission line with characteristic impedance Z0 and transmission delay TD
TLine2	Lossless transmission line with characteristic impedance Z0, frequency F and normalized length NL
TLine3	Lossless transmission line with characteristic impedance Z0 and frequency F

Modelica.Electrical.Analog.Lines.OLine



Lossy Transmission Line

Information

Lossy Transmission Line. The lossy transmission line OLine consists of segments of lumped resistances and inductances in series and conductances and capacitances that are connected with the reference pin p3. The precision of the model depends on the number N of lumped segments.

References:

Johnson, B.; Quarles, T.; Newton, A. R.; Pederson, D. O.; Sangiovanni-Vincentelli, A.: SPICE3 Version 3e User's Manual (April 1, 1991). Department of Electrical Engineering and Computer Sciences, University of California, Berkley p. 12, p. 106 - 107

Parameters

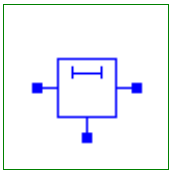
Type	Name	Default	Description

Real	r	1	Resistance per meter [Ohm/m]
Real	l	1	Inductance per meter [H/m]
Real	g	1	Conductance per meter [S/m]
Real	c	1	Capacitance per meter [F/m]
Length	length	1	Length of line [m]
Integer	N	1	Number of lumped segments

Connectors

Type	Name	Description
Pin	p1	
Pin	p2	
Pin	p3	

Modelica.Electrical.Analog.Lines.ULine



Lossy RC Line

Information

The lossy RC line ULine consists of segments of lumped series resistances and capacitances that are connected with the reference pin p3. The precision of the model depends on the number N of lumped segments.

References

Johnson, B.; Quarles, T.; Newton, A. R.; Pederson, D. O.; Sangiovanni-Vincentelli, A.
SPICE3 Version 3e User's Manual (April 1, 1991). Department of Electrical Engineering and Computer Sciences, University of California, Berkley p. 22, p. 124

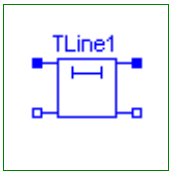
Parameters

Type	Name	Default	Description
Real	r	1	Resistance per meter [Ohm/m]
Real	c	1	Capacitance per meter [F/m]
Length	length	1	Length of line [m]
Integer	N	1	Number of lumped segments

Connectors

Type	Name	Description
Pin	p1	
Pin	p2	
Pin	p3	

Modelica.Electrical.Analog.Lines.TLine1



Lossless transmission line with characteristic impedance Z0 and transmission delay TD

Information

Lossless transmission line with characteristic impedance Z0 and transmission delay TD The lossless transmission line TLine1 is a two Port. Both port branches consist of a resistor with characteristic impedance Z0 and a controled voltage source that takes into consideration the transmission delay TD. For further details see Branin's article below. The model parameters can be derived from inductance and capacitance per length (L' resp. C'), i. e. Z0 = sqrt(L'/C') and TD = sqrt(L'*C')*length_of_line. Resistance R' and conductance C' per meter are assumed to be zero.

References:

Branin Jr., F. H.
Transient Analysis of Lossless Transmission Lines. Proceedings of the IEEE 55(1967), 2012 - 2013
Hoefer, E. E. E.; Nielinger, H.
SPICE : Analyseprogramm fuer elektronische Schaltungen. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.

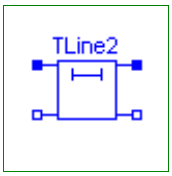
Parameters

Type	Name	Default	Description
Resistance	Z0	1	Characteristic impedance [Ohm]
Time	TD	1	Transmission delay [s]

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Lines.TLine2



Lossless transmission line with characteristic impedance Z0, frequency F and normalized length NL

Information

Lossless transmission line with characteristic impedance Z0, frequency F and normalized length NL The lossless transmission line TLine2 is a two Port. Both port branches consist of a resistor with the value of the characteristic impedance Z0 and a controled voltage source that takes into consideration the transmission delay. For further details see Branin's article below. Resistance R' and conductance C' per meter are assumed to be zero. The characteristic impedance Z0 can be derived from inductance and capacitance per length (L' resp. C'), i. e. Z0 = sqrt(L'/C'). The normalized length NL is equal to the length of the line divided by the wavelength corresponding to the frequency F, i. e. the transmission delay TD is the quotient of NL and F.

References:

Branin Jr., F. H.
Transient Analysis of Lossless Transmission Lines. Proceedings of the IEEE 55(1967), 2012 - 2013
Hoefer, E. E. E.; Nielinger, H.
SPICE : Analyseprogramm fuer elektronische Schaltungen. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.

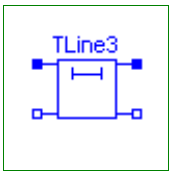
Parameters

Type	Name	Default	Description
Resistance	Z0	1	Characteristic impedance [Ohm]
Frequency	F	1	Frequency [Hz]
Real	NL	1	Normalized length

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Lines.TLine3



Lossless transmission line with characteristic impedance Z0 and frequency F

Information

Lossless transmission line with characteristic impedance Z0 and frequency F The lossless transmission line TLine3 is a two Port. Both port branches consist of a resistor with value of the characteristic impedance Z0 and a controled voltage source that takes into consideration the transmission delay. For further details see Branin's article below. Resistance R' and conductance C' per meter are assumed to be zero. The characteristic impedance Z0 can be derived from inductance and capacitance per length (L' resp. C'), i. e. Z0 = sqrt(L'/C'). The length of the line is equal to a quarter of the wavelength corresponding to the frequency F, i. e. the transmission delay is the quotient of 4 and F. In this case, the caracteristic impedance is called natural impedance.

References:

Branin Jr., F. H.
Transient Analysis of Lossless Transmission Lines. Proceedings of the IEEE 55(1967), 2012 - 2013
Hoefer, E. E. E.; Nielinger, H.
SPICE : Analyseprogramm fuer elektronische Schaltungen. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.

Parameters

Type	Name	Default	Description
Resistance	Z0	1	Natural impedance [Ohm]
Frequency	F	1	Frequency [Hz]

Connectors

Type	Name	Description
PositivePin	p1	Positive pin of the left port (potential p1.v > n1.v for positive voltage drop v1)
NegativePin	n1	Negative pin of the left port
PositivePin	p2	Positive pin of the right port (potential p2.v > n2.v for positive voltage drop v2)
NegativePin	n2	Negative pin of the right port

Modelica.Electrical.Analog.Semiconductors

Semiconductor devices such as diode, MOS and bipolar transistor

Information

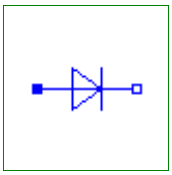
This package contains semiconductor devices:

- diode
- MOS transistors
- bipolar transistors
- diode, MOS and bipolar transistors with temperature dependent characteristic and a heatPort for connection to the thermal domain

Package Content

Name	Description
Diode	Simple diode
PMOS	Simple MOS Transistor
NMOS	Simple MOS Transistor
NPN	Simple BJT according to Ebers-Moll
PNP	Simple BJT according to Ebers-Moll
HeatingDiode	Simple diode with heating port
HeatingNMOS	Simple MOS Transistor with heating port
HeatingPMOS	Simple PMOS Transistor with heating port
HeatingNPN	Simple NPN BJT according to Ebers-Moll with heating port
HeatingPNP	Simple PNP BJT according to Ebers-Moll with heating port

Modelica.Electrical.Analog.Semiconductors.Diode



Simple diode

Information

The simple diode is a one port. It consists of the diode itself and an parallel ohmic resistance *R*. The diode formula is:

```
                 v/vt
i  =  ids ( e       - 1).
```

If the exponent *v/vt* reaches the limit *maxex*, the diode characterisic is linearly continued to avoid overflow.

Parameters

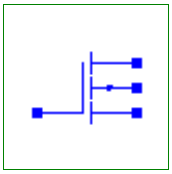
Type	Name	Default	Description
Current	Ids	1.e-6	Saturation current [A]
Voltage	Vt	0.04	Voltage equivalent of temperature (kT/qn) [V]
Real	Maxexp	15	Max. exponent for linear continuation

Resistance	R	1.e8	Parallel ohmic resistance [Ohm]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Semiconductors.PMOS



Simple MOS Transistor

Information

The PMOS model is a simple model of a p-channel metal-oxide semiconductor FET. It differs slightly from the device used in the SPICE simulator. For more details please care for H. Spiro.

The model does not consider capacitances. A high drain-source resistance RDS is included to avoid numerical difficulties.

References:

Spiro, H.: Simulation integrierter Schaltungen. R. Oldenbourg Verlag Muenchen Wien 1990.

Some typical parameter sets are:

W	L	Beta	Vt	K2	K5	DW	DL
m	m	A/V^2	V	-	-	m	m
50.e-6	8.e-6	.0085e-3	-.15	.41	.839	-3.8e-6	-4.0e-6
20.e-6	6.e-6	.0105e-3	-1.0	.41	.839	-2.5e-6	-2.1e-6
30.e-6	5.e-6	.0059e-3	-.3	.98	1.01	0	-3.9e-6
30.e-6	5.e-6	.0152e-3	-.69	.104	1.1	-.8e-6	-.4e-6
30.e-6	5.e-6	.0163e-3	-.69	.104	1.1	-.8e-6	-.4e-6
30.e-6	5.e-6	.0182e-3	-.69	.086	1.06	-.1e-6	-.6e-6
20.e-6	6.e-6	.0074e-3	-1.	.4	.59	0	0

Parameters

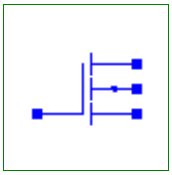
Type	Name	Default	Description
Length	W	20.0e-6	Width [m]
Length	L	6.0e-6	Length [m]
Transconductance	Beta	0.0105e-3	Transconductance parameter [A/V2]
Voltage	Vt	-1.0	Zero bias threshold voltage [V]
Real	K2	0.41	Bulk threshold parameter
Real	K5	0.839	Reduction of pinch-off region
Length	dW	-2.5e-6	Narrowing of channel [m]
Length	dL	-2.1e-6	Shortening of channel [m]
Resistance	RDS	1.e+7	Drain-Source-Resistance [Ohm]

Connectors

Type	Name	Description
Pin	D	Drain

Pin	G	Gate
Pin	S	Source
Pin	B	Bulk

Modelica.Electrical.Analog.Semiconductors.NMOS



Simple MOS Transistor

Information

The NMos model is a simple model of a n-channel metal-oxide semiconductor FET. It differs slightly from the device used in the SPICE simulator. For more details please care for H. Spiro.

The model does not consider capacitances. A high drain-source resistance RDS is included to avoid numerical difficulties.

```
  W        L        Beta         Vt        K2       K5        DW          DL
  m        m        A/V^2        V         -        -         m           m
  12.e-6   4.e-6   .062e-3    -4.5       .24      .61      -1.2e-6      -.9e-6
depletion
  60.e-6   3.e-6   .048e-3      .1       .08      .68      -1.2e-6      -.9e-6
enhancement
  12.e-6   4.e-6   .0625e-3    -.8       .21      .78      -1.2e-6      -.9e-6
zero
  50.e-6   8.e-6   .0299e-3     .24     1.144     .7311    -5.4e-6     -4.e-6
  20.e-6   6.e-6   .041e-3      .8      1.144     .7311    -2.5e-6     -1.5e-6
  30.e-6   9.e-6   .025e-3    -4.        .861     .878     -3.4e-6     -1.74e-6
  30.e-6   5.e-6   .031e-3      .6      1.5       .72       0          -3.9e-6
  50.e-6   6.e-6   .0414e-3   -3.8       .34      .8       -1.6e-6     -2.e-6
depletion
  50.e-6   5.e-6   .03e-3       .37      .23      .86      -1.6e-6     -2.e-6
enhancement
  50.e-6   6.e-6   .038e-3     -.9       .23      .707     -1.6e-6     -2.e-6
zero
  20.e-6   4.e-6   .06776e-3    .5409    .065     .71       -.8e-6      -.2e-6
  20.e-6   4.e-6   .06505e-3    .6209    .065     .71       -.8e-6      -.2e-6
  20.e-6   4.e-6   .05365e-3    .6909    .03      .8        -.3e-6      -.2e-6
  20.e-6   4.e-6   .05365e-3    .4909    .03      .8        -.3e-6      -.2e-6
  12.e-6   4.e-6   .023e-3    -4.5       .29      .6        0           0
depletion
  60.e-6   3.e-6   .022e-3      .1       .11      .65       0           0
enhancement
  12.e-6   4.e-6   .038e-3     -.8       .33      .6        0           0
zero
  20.e-6   6.e-6   .022e-3      .8      1         .66       0           0
```

References:

Spiro, H.: Simulation integrierter Schaltungen. R. Oldenbourg Verlag Muenchen Wien 1990.

Parameters

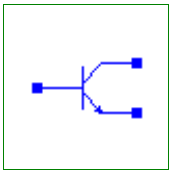
Type	Name	Default	Description
Length	W	20.e-6	Width [m]
Length	L	6.e-6	Length [m]
Transconductance	Beta	0.041e-3	Transconductance parameter [A/V2]

Voltage	Vt	0.8	Zero bias threshold voltage [V]
Real	K2	1.144	Bulk threshold parameter
Real	K5	0.7311	Reduction of pinch-off region
Length	dW	-2.5e-6	narrowing of channel [m]
Length	dL	-1.5e-6	shortening of channel [m]
Resistance	RDS	1.e+7	Drain-Source-Resistance [Ohm]

Connectors

Type	Name	Description
Pin	D	Drain
Pin	G	Gate
Pin	S	Source
Pin	B	Bulk

Modelica.Electrical.Analog.Semiconductors.NPN



Simple BJT according to Ebers-Moll

Information

This model is a simple model of a bipolar npn junction transistor according to Ebers-Moll.

A typical parameter set is:

```
  Bf   Br   Is      Vak   Tauf     Taur   Ccs    Cje      Cjc      Phie  Me    PHic   Mc
Gbc     Gbe     Vt
  -    -    A       V     s        s      F      F        F        V     -     V      -
mS      mS      V
  50   0.1 1e-16   0.02 0.12e-9 5e-9   1e-12 0.4e-12 0.5e-12 0.8   0.4   0.8
0.333   1e-15   1e-15   0.02585
```

References:

Vlach, J.; Singal, K.: Computer methods for circuit analysis and design. Van Nostrand Reinhold, New York 1983 on page 317 ff.

Parameters

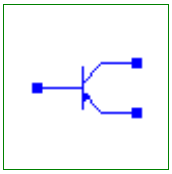
Type	Name	Default	Description
Real	Bf	50	Forward beta
Real	Br	0.1	Reverse beta
Current	Is	1.e-16	Transport saturation current [A]
InversePotential	Vak	0.02	Early voltage (inverse), 1/Volt [1/V]
Time	Tauf	0.12e-9	Ideal forward transit time [s]
Time	Taur	5e-9	Ideal reverse transit time [s]
Capacitance	Ccs	1e-12	Collector-substrat(ground) cap. [F]
Capacitance	Cje	0.4e-12	Base-emitter zero bias depletion cap. [F]
Capacitance	Cjc	0.5e-12	Base-coll. zero bias depletion cap. [F]
Voltage	Phie	0.8	Base-emitter diffusion voltage [V]
Real	Me	0.4	Base-emitter gradation exponent
Voltage	Phic	0.8	Base-collector diffusion voltage [V]

Real	Mc	0.333	Base-collector gradation exponent
Conductance	Gbc	1e-15	Base-collector conductance [S]
Conductance	Gbe	1e-15	Base-emitter conductance [S]
Voltage	Vt	0.02585	Voltage equivalent of temperature [V]
Real	EMin	-100	if x < EMin, the exp(x) function is linearized
Real	EMax	40	if x > EMax, the exp(x) function is linearized

Connectors

Type	Name	Description
Pin	C	Collector
Pin	B	Base
Pin	E	Emitter

Modelica.Electrical.Analog.Semiconductors.PNP



Simple BJT according to Ebers-Moll

Information

This model is a simple model of a bipolar pnp junction transistor according to Ebers-Moll.

A typical parameter set is:

```
  Bf   Br   Is      Vak   Tauf     Taur   Ccs    Cje      Cjc      Phie  Me    PHic   Mc
Gbc     Gbe     Vt
  -    -    A       V     s        s      F      F        F        V     -     V      -
mS      mS      V
  50   0.1 1e-16   0.02 0.12e-9 5e-9   1e-12 0.4e-12 0.5e-12 0.8   0.4   0.8
0.333   1e-15   1e-15   0.02585
```

References:

Vlach, J.; Singal, K.: Computer methods for circuit analysis and design. Van Nostrand Reinhold, New York 1983 on page 317 ff.

Parameters

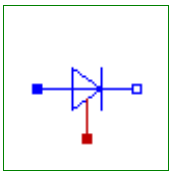
Type	Name	Default	Description
Real	Bf	50	Forward beta
Real	Br	0.1	Reverse beta
Current	Is	1.e-16	Transport saturation current [A]
InversePotential	Vak	0.02	Early voltage (inverse), 1/Volt [1/V]
Time	Tauf	0.12e-9	Ideal forward transit time [s]
Time	Taur	5e-9	Ideal reverse transit time [s]
Capacitance	Ccs	1e-12	Collector-substrat(ground) cap. [F]
Capacitance	Cje	0.4e-12	Base-emitter zero bias depletion cap. [F]
Capacitance	Cjc	0.5e-12	Base-coll. zero bias depletion cap. [F]
Voltage	Phie	0.8	Base-emitter diffusion voltage [V]
Real	Me	0.4	Base-emitter gradation exponent
Voltage	Phic	0.8	Base-collector diffusion voltage [V]
Real	Mc	0.333	Base-collector gradation exponent

Conductance	Gbc	1e-15	Base-collector conductance [S]
Conductance	Gbe	1e-15	Base-emitter conductance [S]
Voltage	Vt	0.02585	Voltage equivalent of temperature [V]
Real	EMin	-100	if x < EMin, the exp(x) function is linearized
Real	EMax	40	if x > EMax, the exp(x) function is linearized

Connectors

Type	Name	Description
Pin	C	Collector
Pin	B	Base
Pin	E	Emitter

Modelica.Electrical.Analog.Semiconductors.HeatingDiode



Simple diode with heating port

Information

The simple diode is an electrical one port, where a heat port is added, which is defined in the Modelica.Thermal library. It consists of the diode itself and an parallel ohmic resistance *R*. The diode formula is:

```
                 v/vt_t
i  =  ids ( e         - 1).
```

where vt_t depends on the temperature of the heat port:

```
vt_t = k*temp/q
```

If the exponent *v/vt_t* reaches the limit *maxex*, the diode characterisic is linearly continued to avoid overflow. The thermal power is calculated by *i*v*.

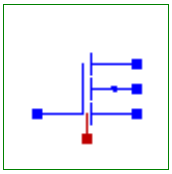
Parameters

Type	Name	Default	Description
Current	Ids	1.e-6	Saturation current [A]
Real	Maxexp	15	Max. exponent for linear continuation
Resistance	R	1.e8	Parallel ohmic resistance [Ohm]
Real	EG	1.11	activation energy
Real	N	1	Emission coefficient
Temperature	TNOM	300.15	Parameter measurement temperature [K]
Real	XTI	3	Temperature exponent of saturation current

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin
HeatPort_a	heatPort	

Modelica.Electrical.Analog.Semiconductors.HeatingNMOS



Simple MOS Transistor with heating port

Information

The NMos model is a simple model of a n-channel metal-oxide semiconductor FET. It differs slightly from the device used in the SPICE simulator. For more details please care for H. Spiro.

A heating port is added for thermal electric simulation. The heating port is defined in the Modelica.Thermal library.

The model does not consider capacitances. A high drain-source resistance RDS is included to avoid numerical difficulties.

```
  W        L        Beta         Vt        K2       K5        DW          DL
  m        m        A/V^2        V         -        -         m           m
  12.e-6   4.e-6   .062e-3    -4.5       .24      .61      -1.2e-6      -.9e-6
depletion
  60.e-6   3.e-6   .048e-3      .1       .08      .68      -1.2e-6      -.9e-6
enhancement
  12.e-6   4.e-6   .0625e-3    -.8       .21      .78      -1.2e-6      -.9e-6
zero
  50.e-6   8.e-6   .0299e-3     .24     1.144     .7311    -5.4e-6     -4.e-6
  20.e-6   6.e-6   .041e-3      .8      1.144     .7311    -2.5e-6     -1.5e-6
  30.e-6   9.e-6   .025e-3    -4.        .861     .878     -3.4e-6     -1.74e-6
  30.e-6   5.e-6   .031e-3      .6      1.5       .72       0          -3.9e-6
  50.e-6   6.e-6   .0414e-3   -3.8       .34      .8       -1.6e-6     -2.e-6
depletion
  50.e-6   5.e-6   .03e-3       .37      .23      .86      -1.6e-6     -2.e-6
enhancement
  50.e-6   6.e-6   .038e-3     -.9       .23      .707     -1.6e-6     -2.e-6
zero
  20.e-6   4.e-6   .06776e-3    .5409    .065     .71       -.8e-6      -.2e-6
  20.e-6   4.e-6   .06505e-3    .6209    .065     .71       -.8e-6      -.2e-6
  20.e-6   4.e-6   .05365e-3    .6909    .03      .8        -.3e-6      -.2e-6
  20.e-6   4.e-6   .05365e-3    .4909    .03      .8        -.3e-6      -.2e-6
  12.e-6   4.e-6   .023e-3    -4.5       .29      .6        0           0
depletion
  60.e-6   3.e-6   .022e-3      .1       .11      .65       0           0
enhancement
  12.e-6   4.e-6   .038e-3     -.8       .33      .6        0           0
zero
  20.e-6   6.e-6   .022e-3      .8      1         .66       0           0
```

References:

Spiro, H.: Simulation integrierter Schaltungen. R. Oldenbourg Verlag Muenchen Wien 1990.

Parameters

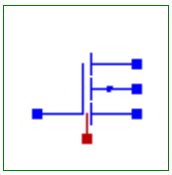
Type	Name	Default	Description
Length	W	20.e-6	Width [m]
Length	L	6.e-6	Length [m]
Transconductance	Beta	0.041e-3	Transconductance parameter [A/V2]
Voltage	Vt	0.8	Zero bias threshold voltage [V]
Real	K2	1.144	Bulk threshold parameter
Real	K5	0.7311	Reduction of pinch-off region

Length	dW	-2.5e-6	narrowing of channel [m]
Length	dL	-1.5e-6	shortening of channel [m]
Resistance	RDS	1.e+7	Drain-Source-Resistance [Ohm]
Temperature	Tnom	300.15	Parameter measurement temperature [K]
Real	kvt	-6.96e-3	fitting parameter for Vt
Real	kk2	6.0e-4	fitting parameter for K22

Connectors

Type	Name	Description
Pin	D	Drain
Pin	G	Gate
Pin	S	Source
Pin	B	Bulk
HeatPort_a	heatPort	

Modelica.Electrical.Analog.Semiconductors.HeatingPMOS



Simple PMOS Transistor with heating port

Information

The PMOS model is a simple model of a p-channel metal-oxide semiconductor FET. It differs slightly from the device used in the SPICE simulator. For more details please care for H. Spiro.

A heating port is added for thermal electric simulation. The heating port is defined in the Modelica.Thermal library.

The model does not consider capacitances. A high drain-source resistance RDS is included to avoid numerical difficulties.

References:

Spiro, H.: Simulation integrierter Schaltungen. R. Oldenbourg Verlag Muenchen Wien 1990.

Some typical parameter sets are:

W	L	Beta	Vt	K2	K5	DW	DL
m	m	A/V^2	V	-	-	m	m
50.e-6	8.e-6	.0085e-3	-.15	.41	.839	-3.8e-6	-4.0e-6
20.e-6	6.e-6	.0105e-3	-1.0	.41	.839	-2.5e-6	-2.1e-6
30.e-6	5.e-6	.0059e-3	-.3	.98	1.01	0	-3.9e-6
30.e-6	5.e-6	.0152e-3	-.69	.104	1.1	-.8e-6	-.4e-6
30.e-6	5.e-6	.0163e-3	-.69	.104	1.1	-.8e-6	-.4e-6
30.e-6	5.e-6	.0182e-3	-.69	.086	1.06	-.1e-6	-.6e-6
20.e-6	6.e-6	.0074e-3	-1.	.4	.59	0	0

Parameters

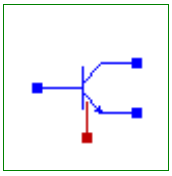
Type	Name	Default	Description
Length	W	20.0e-6	Width [m]
Length	L	6.0e-6	Length [m]
Transconductance	Beta	0.0105e-3	Transconductance parameter [A/V2]
Voltage	Vt	-1.0	Zero bias threshold voltage [V]

Real	K2	0.41	Bulk threshold parameter
Real	K5	0.839	Reduction of pinch-off region
Length	dW	-2.5e-6	Narrowing of channel [m]
Length	dL	-2.1e-6	Shortening of channel [m]
Resistance	RDS	1.e+7	Drain-Source-Resistance [Ohm]
Temperature	Tnom	300.15	Parameter measurement temperature [K]
Real	kvt	-2.9e-3	fitting parameter for Vt
Real	kk2	6.2e-4	fitting parameter for Kk2

Connectors

Type	Name	Description
Pin	D	Drain
Pin	G	Gate
Pin	S	Source
Pin	B	Bulk
HeatPort_a	heatPort	

Modelica.Electrical.Analog.Semiconductors.HeatingNPN



Simple NPN BJT according to Ebers-Moll with heating port

Information

This model is a simple model of a bipolar npn junction transistor according to Ebers-Moll.

A heating port is added for thermal electric simulation. The heating port is defined in the Modelica.Thermal library.

A typical parameter set is (the parameter Vt is no longer used):

```
  Bf   Br   Is      Vak   Tauf     Taur   Ccs    Cje      Cjc      Phie  Me    PHic   Mc
Gbc     Gbe
  -    -    A       V     s        s      F      F        F        V     -     V      -
mS      mS
  50   0.1 1e-16   0.02 0.12e-9 5e-9   1e-12 0.4e-12 0.5e-12 0.8   0.4   0.8
0.333   1e-15   1e-15
```

References:

Vlach, J.; Singal, K.: Computer methods for circuit analysis and design. Van Nostrand Reinhold, New York 1983 on page 317 ff.

Parameters

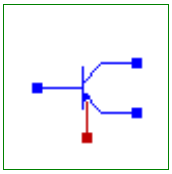
Type	Name	Default	Description
Real	Bf	50	Forward beta
Real	Br	0.1	Reverse beta
Current	Is	1.e-16	Transport saturation current [A]
InversePotential	Vak	0.02	Early voltage (inverse), 1/Volt [1/V]
Time	Tauf	0.12e-9	Ideal forward transit time [s]
Time	Taur	5e-9	Ideal reverse transit time [s]
Capacitance	Ccs	1e-12	Collector-substrat(ground) cap. [F]

Capacitance	Cje	0.4e-12	Base-emitter zero bias depletion cap. [F]
Capacitance	Cjc	0.5e-12	Base-coll. zero bias depletion cap. [F]
Voltage	Phie	0.8	Base-emitter diffusion voltage [V]
Real	Me	0.4	Base-emitter gradation exponent
Voltage	Phic	0.8	Base-collector diffusion voltage [V]
Real	Mc	0.333	Base-collector gradation exponent
Conductance	Gbc	1e-15	Base-collector conductance [S]
Conductance	Gbe	1e-15	Base-emitter conductance [S]
Real	EMin	-100	if x < EMin, the exp(x) function is linearized
Real	EMax	40	if x > EMax, the exp(x) function is linearized
Temperature	Tnom	300.15	Parameter measurement temperature [K]
Real	XTI	3	Temperature exponent for effect on Is
Real	XTB	0	Forward and reverse beta temperature exponent
Real	EG	1.11	Energy gap for temperature effect on Is
Real	NF	1.0	Forward current emission coefficient
Real	NR	1.0	Reverse current emission coefficient
Real	K	1.3806226e-23	Boltzmann's constant
Real	q	1.6021918e-19	Elementary electronic charge

Connectors

Type	Name	Description
Pin	C	Collector
Pin	B	Base
Pin	E	Emitter
HeatPort_a	heatPort	

Modelica.Electrical.Analog.Semiconductors.HeatingPNP



Simple PNP BJT according to Ebers-Moll with heating port

Information

This model is a simple model of a bipolar pnp junction transistor according to Ebers-Moll.

A heating port is added for thermal electric simulation. The heating port is defined in the Modelica.Thermal library.

A typical parameter set is (the parameter Vt is no longer used):

```
  Bf   Br   Is      Vak   Tauf     Taur   Ccs    Cje      Cjc      Phie  Me    PHic   Mc
Gbc     Gbe
  -    -    A       V     s        s      F      F        F        V     -     V      -
mS      mS
  50   0.1  1e-16   0.02  0.12e-9  5e-9   1e-12  0.4e-12  0.5e-12  0.8   0.4   0.8
0.333   1e-15   1e-15
```

References:

Vlach, J.; Singal, K.: Computer methods for circuit analysis and design. Van Nostrand Reinhold, New York 1983 on page 317 ff.

Parameters

Type	Name	Default	Description
Real	Bf	50	Forward beta
Real	Br	0.1	Reverse beta
Current	Is	1.e-16	Transport saturation current [A]
InversePotential	Vak	0.02	Early voltage (inverse), 1/Volt [1/V]
Time	Tauf	0.12e-9	Ideal forward transit time [s]
Time	Taur	5e-9	Ideal reverse transit time [s]
Capacitance	Ccs	1e-12	Collector-substrat(ground) cap. [F]
Capacitance	Cje	0.4e-12	Base-emitter zero bias depletion cap. [F]
Capacitance	Cjc	0.5e-12	Base-coll. zero bias depletion cap. [F]
Voltage	Phie	0.8	Base-emitter diffusion voltage [V]
Real	Me	0.4	Base-emitter gradation exponent
Voltage	Phic	0.8	Base-collector diffusion voltage [V]
Real	Mc	0.333	Base-collector gradation exponent
Conductance	Gbc	1e-15	Base-collector conductance [S]
Conductance	Gbe	1e-15	Base-emitter conductance [S]
Real	EMin	-100	if x < EMin, the exp(x) function is linearized
Real	EMax	40	if x > EMax, the exp(x) function is linearized
Temperature	Tnom	300.15	Parameter measurement temperature [K]
Real	XTI	3	Temperature exponent for effect on Is
Real	XTB	0	Forward and reverse beta temperature exponent
Real	EG	1.11	Energy gap for temperature effect on Is
Real	NF	1.0	Forward current emission coefficient
Real	NR	1.0	Reverse current emission coefficient
Real	K	1.3806226e-23	Boltzmann's constant
Real	q	1.6021918e-19	Elementary electronic charge

Connectors

Type	Name	Description
Pin	C	Collector
Pin	B	Base
Pin	E	Emitter
HeatPort_a	heatPort	

Modelica.Electrical.Analog.Sensors

Potential, voltage, current, and power sensors

Information

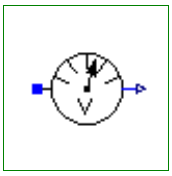
This package contains potential, voltage, and current sensors.

Package Content

Name	Description

PotentialSensor	Sensor to measure the potential
VoltageSensor	Sensor to measure the voltage between two pins
CurrentSensor	Sensor to measure the current in a branch
PowerSensor	Sensor to measure the power

Modelica.Electrical.Analog.Sensors.PotentialSensor

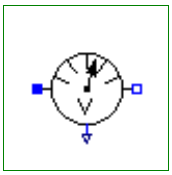


Sensor to measure the potential

Connectors

Type	Name	Description
PositivePin	p	pin to be measured
output RealOutput	phi	Absolute voltage potential as output signal

Modelica.Electrical.Analog.Sensors.VoltageSensor

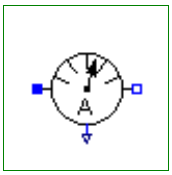


Sensor to measure the voltage between two pins

Connectors

Type	Name	Description
PositivePin	p	positive pin
NegativePin	n	negative pin
output RealOutput	v	Voltage between pin p and n (= p.v - n.v) as output signal

Modelica.Electrical.Analog.Sensors.CurrentSensor

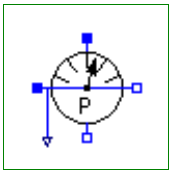


Sensor to measure the current in a branch

Connectors

Type	Name	Description
PositivePin	p	positive pin
NegativePin	n	negative pin
output RealOutput	i	current in the branch from p to n as output signal

Modelica.Electrical.Analog.Sensors.PowerSensor



Sensor to measure the power

Information

This power sensor measures instantaneous electrical power of a singlephase system and has a separated voltage and current path. The pins of the voltage path are `pv` and `nv`, the pins of the current path are `pc` and `nc`. The internal resistance of the current path is zero, the internal resistance of the voltage path is infinite.

Connectors

Type	Name	Description
PositivePin	pc	Positive pin, current path
NegativePin	nc	Negative pin, current path
PositivePin	pv	Positive pin, voltage path
NegativePin	nv	Negative pin, voltage path
output RealOutput	power	

Modelica.Electrical.Analog.Sources

Time-dependend and controlled voltage and current sources

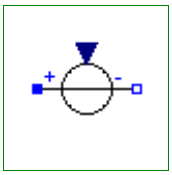
Information

This package contains time-dependend and controlled voltage and current sources.

Package Content

Name	Description
SignalVoltage	Generic voltage source using the input signal as source voltage
ConstantVoltage	Source for constant voltage
StepVoltage	Step voltage source
RampVoltage	Ramp voltage source
SineVoltage	Sine voltage source
ExpSineVoltage	Exponentially damped sine voltage source
ExponentialsVoltage	Rising and falling exponential voltage source
PulseVoltage	Pulse voltage source
SawToothVoltage	Saw tooth voltage source
TrapezoidVoltage	Trapezoidal voltage source
TableVoltage	Voltage source by linear interpolation in a table
SignalCurrent	Generic current source using the input signal as source current
ConstantCurrent	Source for constant current
StepCurrent	Step current source
RampCurrent	Ramp current source
SineCurrent	Sine current source
ExpSineCurrent	Exponentially damped sine current source
ExponentialsCurrent	Rising and falling exponential current source
PulseCurrent	Pulse current source
SawToothCurrent	Saw tooth current source
TrapezoidCurrent	Trapezoidal current source
TableCurrent	Current source by linear interpolation in a table

Modelica.Electrical.Analog.Sources.SignalVoltage

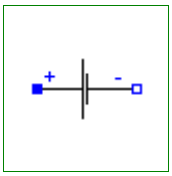


Generic voltage source using the input signal as source voltage

Connectors

Type	Name	Description
PositivePin	p	
NegativePin	n	
input RealInput	v	Voltage between pin p and n (= p.v - n.v) as input signal

Modelica.Electrical.Analog.Sources.ConstantVoltage



Source for constant voltage

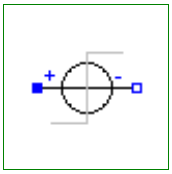
Parameters

Type	Name	Default	Description
Voltage	V	1	Value of constant voltage [V]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.StepVoltage



Step voltage source

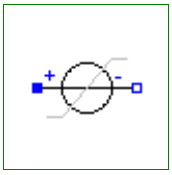
Parameters

Type	Name	Default	Description
Voltage	V	1	Height of step [V]
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.RampVoltage



Ramp voltage source

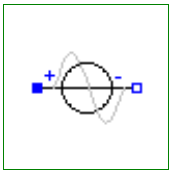
Parameters

Type	Name	Default	Description
Voltage	V	1	Height of ramp [V]
Time	duration	2	Duration of ramp [s]
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.SineVoltage



Sine voltage source

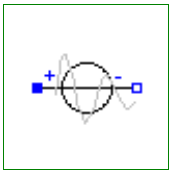
Parameters

Type	Name	Default	Description
Voltage	V	1	Amplitude of sine wave [V]
Angle	phase	0	Phase of sine wave [rad]
Frequency	freqHz	1	Frequency of sine wave [Hz]
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.ExpSineVoltage



Exponentially damped sine voltage source

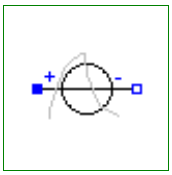
Parameters

Type	Name	Default	Description
Voltage	V	1	Amplitude of sine wave [V]
Frequency	freqHz	2	Frequency of sine wave [Hz]
Angle	phase	0	Phase of sine wave [rad]
Damping	damping	1	Damping coefficient of sine wave [s-1]
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.ExponentialsVoltage



Rising and falling exponential voltage source

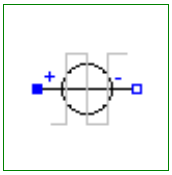
Parameters

Type	Name	Default	Description
Real	vMax	1	Upper bound for rising edge
Time	riseTime	0.5	Rise time [s]
Time	riseTimeConst	0.1	Rise time constant [s]
Time	fallTimeConst	riseTimeConst	Fall time constant [s]
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.PulseVoltage



Pulse voltage source

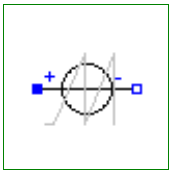
Parameters

Type	Name	Default	Description
Voltage	V	1	Amplitude of pulse [V]
Real	width	50	Width of pulse in % of period
Time	period	1	Time for one period [s]
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.SawToothVoltage



Saw tooth voltage source

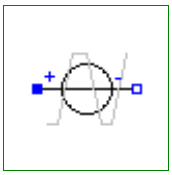
Parameters

Type	Name	Default	Description
Voltage	V	1	Amplitude of saw tooth [V]
Time	period	1	Time for one period [s]
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.TrapezoidVoltage



Trapezoidal voltage source

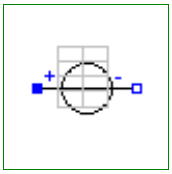
Parameters

Type	Name	Default	Description
Voltage	V	1	Amplitude of trapezoid [V]
Time	rising	0	Rising duration of trapezoid [s]
Time	width	0.5	Width duration of trapezoid [s]
Time	falling	0	Falling duration of trapezoid [s]
Time	period	1	Time for one period [s]
Integer	nperiod	-1	Number of periods (< 0 means infinite number of periods)
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.TableVoltage



Voltage source by linear interpolation in a table

Information

This block generates a voltage source by **linear interpolation** in a table. The time points and voltage values are stored in a matrix **table[i,j]**, where the first column table[:,1] contains the time points and the second column contains the voltage to be interpolated. The table interpolation has the following proporties:

- The time points need to be **monotonically increasing**.
- **Discontinuities** are allowed, by providing the same time point twice in the table.
- Values **outside** of the table range, are computed by **extrapolation** through the last or first two points of the table.
- If the table has only **one row**, no interpolation is performed and the voltage value is just returned independantly of the actual time instant, i.e., this is a constant voltage source.
- Via parameters **startTime** and **offset** the curve defined by the table can be shifted both in time and in the voltage.
- The table is implemented in a numerically sound way by generating **time events** at interval boundaries, in order to not integrate over a discontinuous or not differentiable points.

Example:

```
   table = [0  0
            1  0
            1  1
            2  4
            3  9
            4 16]
If, e.g., time = 1.0, the voltage v =  0.0 (before event), 1.0 (after event)
    e.g., time = 1.5, the voltage v =  2.5,
    e.g., time = 2.0, the voltage v =  4.0,
    e.g., time = 5.0, the voltage v = 23.0 (i.e. extrapolation).
```

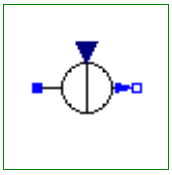
Parameters

Type	Name	Default	Description
Real	table[:, :]	[0, 0; 1, 1; 2, 4]	Table matrix (time = first column, voltage = second column)
Voltage	offset	0	Voltage offset [V]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.SignalCurrent



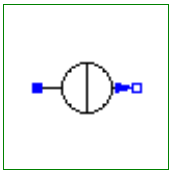
Generic current source using the input signal as source current

Connectors

Type	Name	Description
PositivePin	p	
NegativePin	n	
input RealInput	i	Current flowing from pin p to pin n as input signal

---

Modelica.Electrical.Analog.Sources.ConstantCurrent



Source for constant current

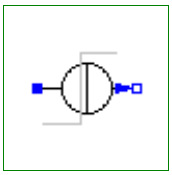
Parameters

Type	Name	Default	Description
Current	I	1	Value of constant current [A]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.StepCurrent



Step current source

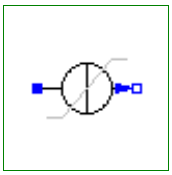
Parameters

Type	Name	Default	Description
Current	I	1	Height of step [A]
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.RampCurrent



Ramp current source

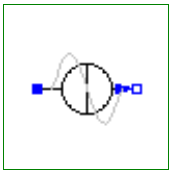
Parameters

Type	Name	Default	Description
Current	I	1	Height of ramp [A]
Time	duration	2	Duration of ramp [s]
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.SineCurrent



Sine current source

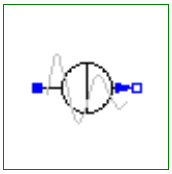
Parameters

Type	Name	Default	Description
Current	I	1	Amplitude of sine wave [A]
Angle	phase	0	Phase of sine wave [rad]
Frequency	freqHz	1	Frequency of sine wave [Hz]
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.ExpSineCurrent



Exponentially damped sine current source

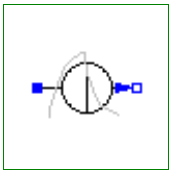
Parameters

Type	Name	Default	Description
Real	I	1	Amplitude of sine wave
Frequency	freqHz	2	Frequency of sine wave [Hz]
Angle	phase	0	Phase of sine wave [rad]
Damping	damping	1	Damping coefficient of sine wave [s-1]
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.ExponentialsCurrent



Rising and falling exponential current source

Parameters

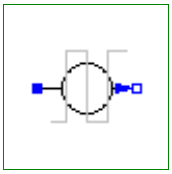
Type	Name	Default	Description
Real	iMax	1	Upper bound for rising edge
Time	riseTime	0.5	Rise time [s]

Time	riseTimeConst	0.1	Rise time constant [s]
Time	fallTimeConst	riseTimeConst	Fall time constant [s]
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.PulseCurrent



Pulse current source

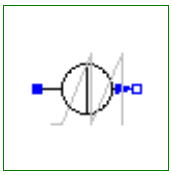
Parameters

Type	Name	Default	Description
Current	I	1	Amplitude of pulse [A]
Real	width	50	Width of pulse in % of period
Time	period	1	Time for one period [s]
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.SawToothCurrent



Saw tooth current source

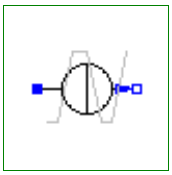
Parameters

Type	Name	Default	Description
Current	I	1	Amplitude of saw tooth [A]
Time	period	1	Time for one period [s]
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.TrapezoidCurrent



Trapezoidal current source

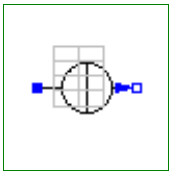
Parameters

Type	Name	Default	Description
Current	I	1	Amplitude of trapezoid [A]
Time	rising	0	Rising duration of trapezoid [s]
Time	width	0.5	Width duration of trapezoid [s]
Time	falling	0	Falling duration of trapezoid [s]
Time	period	1	Time for one period [s]
Integer	nperiod	-1	Number of periods (< 0 means infinite number of periods)
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Analog.Sources.TableCurrent



Current source by linear interpolation in a table

Information

This block generates a current source by **linear interpolation** in a table. The time points and current values are stored in a matrix **table[i,j]**, where the first column table[:,1] contains the time points and the second column contains the current to be interpolated. The table interpolation has the following proporties:

- The time points need to be **monotonically increasing**.
- **Discontinuities** are allowed, by providing the same time point twice in the table.
- Values **outside** of the table range, are computed by **extrapolation** through the last or first two points of the table.
- If the table has only **one row**, no interpolation is performed and the current value is just returned independantly of the actual time instant, i.e., this is a constant current source.
- Via parameters **startTime** and **offset** the curve defined by the table can be shifted both in time and in the current.
- The table is implemented in a numerically sound way by generating **time events** at interval boundaries, in order to not integrate over a discontinuous or not differentiable points.

Example:

```
   table = [0  0
            1  0
            1  1
            2  4
            3  9
            4 16]
If, e.g., time = 1.0, the current i =  0.0 (before event), 1.0 (after event)
    e.g., time = 1.5, the current i =  2.5,
```

```
e.g., time = 2.0, the current i =  4.0,
e.g., time = 5.0, the current i = 23.0 (i.e. extrapolation).
```

Parameters

Type	Name	Default	Description
Real	table[:, :]	[0, 0; 1, 1; 2, 4]	Table matrix (time = first column, current = second column)
Current	offset	0	Current offset [A]
Time	startTime	0	Time offset [s]

Connectors

Type	Name	Description
PositivePin	p	Positive pin (potential p.v > n.v for positive voltage drop v)
NegativePin	n	Negative pin

Modelica.Electrical.Digital

Library for digital electrical components based on the VHDL standard with 9-valued logic and conversion to 2-,3-,4-valued logic

Information

This library contains packages for digital electrical components. Both, type system and models are based on the VHDL standard (IEEE Std 1076-1987 VHDL, IEEE Std 1076-1993 VHDL, IEEE Std 1164 Multivalue Logic System):

- Interfaces: Definition of signals and interfaces
- Tables: All truth tables needed
- Delay: Transport and inertial delay
- Basic: Basic logic without delay
- Gates: Basic gates composed by basic components and inertial delay
- Tristate: (not yet available)
- FlipFlops: (not yet available)
- Latches: (not yet available)
- TransferGates: (not yet available)
- Multiplexers (not yet available)
- Memory: Ram, Rom, (not yet available)
- Sources: Time-dependend signal sources
- Converters
- Examples

The logic values are coded by integer values. The following code table is necessary for both setting of input and interpreting the output values.

Code Table:

Logic value	Integer code	Meaning
'U'	1	Uninitialized
'X'	2	Forcing Unknown
'0'	3	Forcing 0
'1'	4	Forcing 1
'Z'	5	High Impedance

'W'	6	Weak Unknown
'L'	7	Weak 0
'H'	8	Weak 1
'-'	9	Don't care

The library will be developed in two main steps. The first step contains the basic components and the gates. In the next step the more complicated devices will be added. Currently the first step of the library is implemented and released for public use.

Copyright © 1998-2007, Modelica Association and Fraunhofer-Gesellschaft.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
UsersGuide	User's Guide
Examples	Examples that demonstrate the usage of the Digital electrical components
Interfaces	Connectors for Digital electrical components
Tables	Truth tables for all components of package Digital
Delay	Transport and inertial delay blocks
Basic	Basic logic blocks without delays
Gates	Logic gates including delays
Sources	Time-dependend digital signal sources
Converters	Converters between 2-,3-,4- and 9-valued logic

Modelica.Electrical.Digital.UsersGuide

User's Guide

Library **Electrical.Digital** is a **free** Modelica package providing components to model **digital** electronic systems based on combinational and sequential logic in a convenient way. This package contains the **User's Guide** for the library and has the following content:

1. Overview of library gives an overview of the library.
2. A first example demonstrates at hand of a first example how to use this library.
3. An application example demonstrates a generic n-bit adder. .
4. Release Notes summarizes the differences between different versions of this library.
5. Literature provides references that have been used to design and implement this library.
6. Contact provides information about the authors of the library as well as acknowledgments.

Package Content

Name	Description
OverView	Overview of library
FirstExample	A first example
ApplicationExample	An application example
ReleaseNotes	Release notes

Literature	Literature
Contact	Contact

Modelica.Electrical.Digital.UsersGuide.OverView

Overview of library

In this section, an overview of the most important features of this library is given.

(will be added as soon as possible).

Modelica.Electrical.Digital.UsersGuide.FirstExample

A first example

A first example will be given here (not yet done).

Modelica.Electrical.Digital.UsersGuide.ApplicationExample

An application example

An application example will be given here (not yet done).

Modelica.Electrical.Digital.UsersGuide.ReleaseNotes

Release notes

Version 1.0.7, 2005-07-01

- xxxx

Version 1.0.6, 2004-10-18

- Missing HTML tags added (problems with mismatched pre tags fixed).
- CVS ID string deleted.

Version 1.0.5, 2004-10-01

- Wrong identifiers x0 and Tdel in HalfAdder example fixed.
- Experiment command in FlipFlop example deleted.
- Known issue: Pulse source causes a warning in Dymola. It is recommended to use Clock source.

Version 1.0.4, 2004-09-30

- Documentation improved.

Version 1.0.3, 2004-09-21

- Table names changed from "map" to "Table".
- Icons for converters modified.
- LogicValueType renamed to Logic. For the Electrical.Digital library the type Logic has a fundamental meaning. Logic is similar to Real, Integer or Boolean in other packages. Names for converters are

now more consistent (LogicToBoolean, RealToLogic etc.).
- Icons for gates and sources improved.
- New examples added.
- Internal names for signals and ports unified.
- Simple Clock source added in addition to Pulse source (for convenience reasons).

Version 1.0.2, 2004-09-13

- First prerelease for discussions at the 40th Modelica Design Meeting.

Version 1.0.1, 2004-06-01

- Packages Tables, Basic, and Gates implemented.
- Transport and inertial delay implemented and successfully tested.

Version 1.0.0, 2003-05-01

- A first version has been implemented for case studies.

Modelica.Electrical.Digital.UsersGuide.Literature

Literature

The Electrical.Digital library is based on the following references:

Ashenden, P. J.:
The Designer's Guide to VHDL. San Francisco: Morgan Kaufmann, 1995, 688 p. ISBN 1-55860-270-4.

IEEE 1076-1993:
IEEE Standard VHDL Language Reference Manual (ANSI). 288 p. ISBN 1-55937-376-8. IEEE Ref. SH16840-NYF.

IEEE 1164-1993:
IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164). 24 p. ISBN 1-55937-299-0. IEEE Ref. SH16097-NYF.

Lipsett, R.; Schaefer, C.; Ussery, C.:
VHDL: Hardware Description and Design. Boston: Kluwer, 1989, 299 p. ISBN 079239030X.

Navabi, Z:
VHDL: Analysis and Modeling of Digital Systems. New York: McGraw-Hill, 1993, 375 p. ISBN 0070464723.

Modelica.Electrical.Digital.UsersGuide.Contact

Contact

Main Authors:
Christoph Clauß <Christoph.Clauss@eas.iis.fraunhofer.de>
André Schneider <Andre.Schneider@eas.iis.fraunhofer.de>
Fraunhofer Institute for Integrated Circuits (IIS)
Design Automation Department (EAS)
Zeunerstraße 38
D-01069 Dresden
Germany

Acknowledgements:
We thank our colleague Ulrich Donath <Ulrich.Donath@eas.iis.fraunhofer.de> for his support and fruitful discussions regarding all questions on VHDL and the IEEE 1164 standard logic libraries. Furthermore, we thank our students Teresa Schlegel and Enrico Weber for implementing and carefully testing many models and examples.

Modelica.Electrical.Digital.Examples

Examples that demonstrate the usage of the Digital electrical components

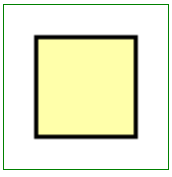
Information

This package contains examples that demonstrate the usage of the components of the Electrical.Digital library.

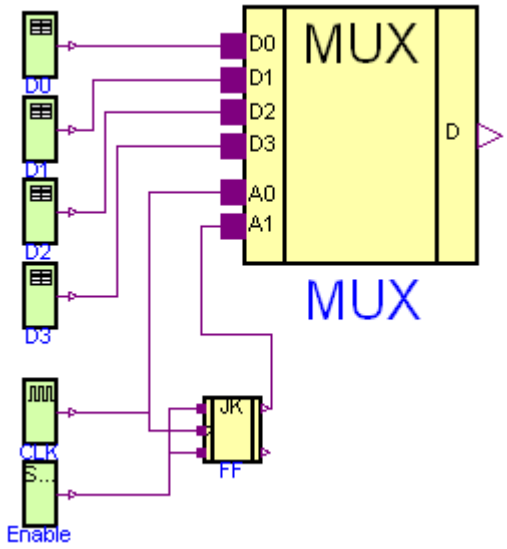
Package Content

Name	Description
Multiplexer	4 to 1 Bit Multiplexer Example
FlipFlop	Pulse Triggered Master Slave Flip-Flop
HalfAdder	adding circuit for binary numbers without input carry bit
FullAdder	Full 1 Bit Adder Example
Adder4	4 Bit Adder Example
Counter3	3 Bit Counter Example
Counter	Generic N Bit Counter Example
Utilities	Utility components used by package Examples

Modelica.Electrical.Digital.Examples.Multiplexer



4 to 1 Bit Multiplexer Example

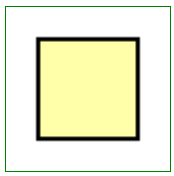


Information

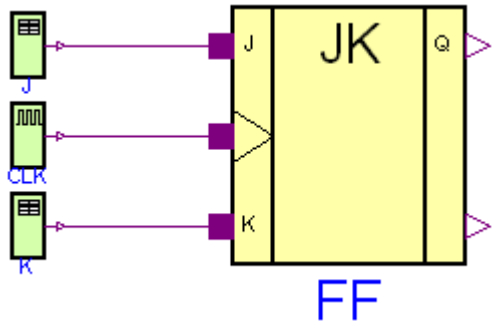
4 to 1 Bit Multiplexer

The multiplexer converts a parallel 4 bit signal in a sequential 1 bit stream.

Modelica.Electrical.Digital.Examples.FlipFlop



Pulse Triggered Master Slave Flip-Flop

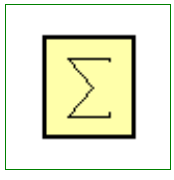


Information

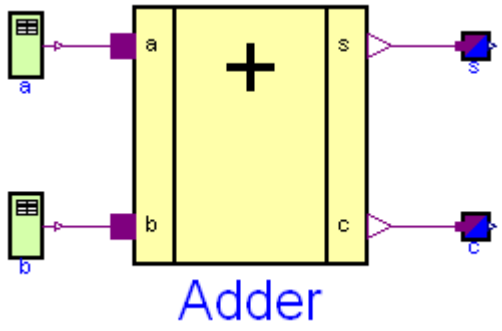
FlipFlop

Pulse-triggered master-slave flip-flop.

Modelica.Electrical.Digital.Examples.HalfAdder



adding circuit for binary numbers without input carry bit



Information

This example demonstrates an adding circuit for binary numbers, which internally realizes the interconnection to And and to Xor in the final sum.

1 + 0 = 1
0 + 1 = 1
1 + 1 = 10
0 + 0 = 0

a + **b** = **s**
(The carry of this adding is **c**.)

and

a * **b** = **s**
(It is an interconnection to And.)

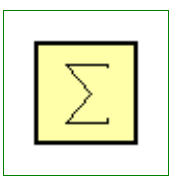
a * **b** + **a** * **b** = **a** Xor **b** = **c**
(It is an interconnection to Xor.)

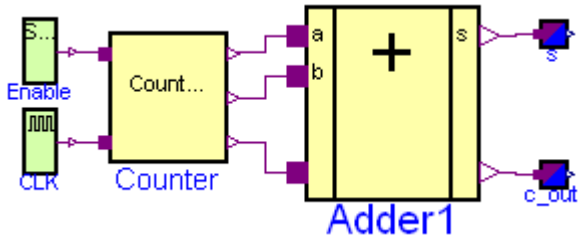
a	b	c	s	t
1	0	1	0	1
0	1	1	0	2
1	1	0	1	3
0	0	0	0	4

t is the pick-up instant of the next bit(s) in the simulation. The simulation stop time should be 5 seconds.

Modelica.Electrical.Digital.Examples.FullAdder

Full 1 Bit Adder Example





Information

It is an adding circuit for binary numbers with input carry bit, which consists of two HalfAdders.

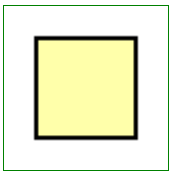
a.y, **b**.y and **c**.y are the inputs of the FullAdder.
cout = **Or1**.y and **h**.s are the outputs of the Fulladder.

t is the pick-up instant of the next bit(s) in the simulation.

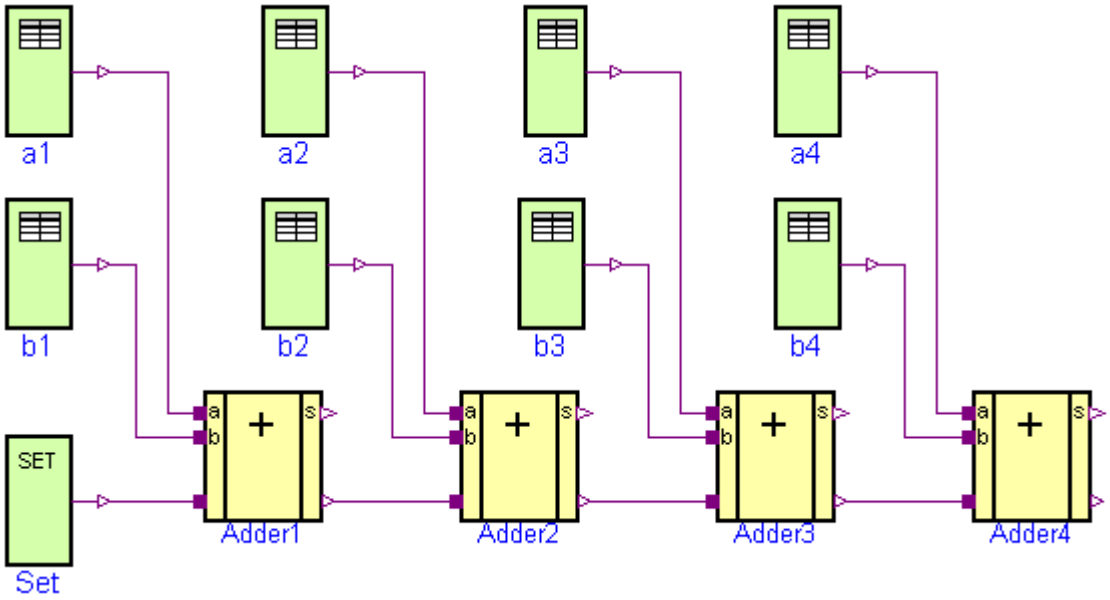
a.y	**b**.y	**c**.y	**c**out	**h**.s	**t**
1	0	0	0	1	1
0	1	0	0	1	2
0	0	1	0	1	3
1	1	0	1	0	4
0	1	1	1	0	5
1	0	1	1	0	6
1	1	1	1	1	7
0	0	0	0	0	8

The simulation stop time should be 10 seconds.

Modelica.Electrical.Digital.Examples.Adder4



4 Bit Adder Example



Information

Four Fulladders are combined to built a four bit adder unit.

In dependence on time five additions are carried out:

```
at t = 0
 a        0 0 0 0
 b    +   0 0 0 0
 s      0 0 0 0 0
at t = 2
 a        0 1 1 0
 b    +   0 0 1 1
 s      1 0 1 0 0

at t = 4
 a      1 1 0 0
 b   +  1 1 1 0
 s    0 0 1 0 1
```

```
at t = 1
   a      1 1 1 0
   b   +  1 0 1 1
   s    1 0 0 1 0
 at t = 3
   a      1 1 1 0
   b   +  1 0 1 0
   s    0 0 0 1 1
```

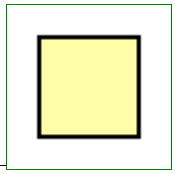
To show the influence of delay a large delay time of 0.1s is choosen. Furthermore, all signals are initialized with U, the unitialized value. Please remember, that the nine logic values are coded by the numbers 1,...,9. The summands a and b can be found at the output signals of the taba and tabb sources. The result can be seen in the output signals of the Fulladders according to:

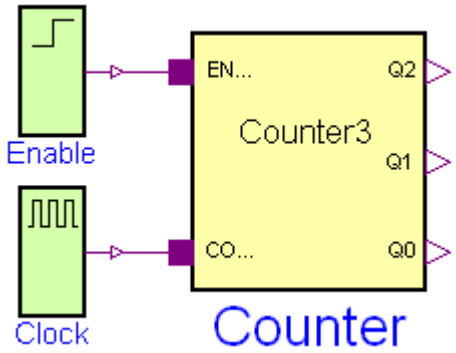
```
a                          a4.y       a3.y       a2.y       a1.y
b                          b4.y       b3.y       b2.y       b1.y
sum    Adder4.c_out  Adder4.s   Adder3.s   Adder2.s   Adder1.s
```

The simulation stop time has to be 5s.

Modelica.Electrical.Digital.Examples.Counter3

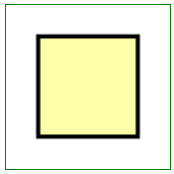
3 Bit Counter Example



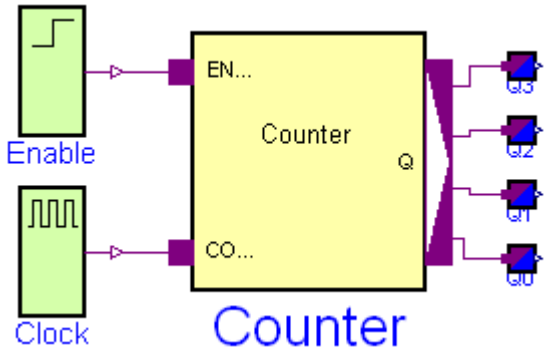


Information

Modelica.Electrical.Digital.Examples.Counter



Generic N Bit Counter Example



Information

Modelica.Electrical.Digital.Examples.Utilities

Utility components used by package Examples

Information

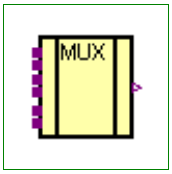
This package contains utility components used by package Examples.

Package Content

Name	Description
MUX4	4 to 1 Bit Multiplexer
RS	Unclocked RS FlipFlop
RSFF	Unclocked RS FlipFlop
DFF	D FlipFlop
JKFF	JK FlipFlop

HalfAdder	
FullAdder	adding circuit for binary numbers with input carry bit
Adder	Generic N Bit Adder
Counter3	3 Bit Counter
Counter	Generic N Bit Counter

Modelica.Electrical.Digital.Examples.Utilities.MUX4



4 to 1 Bit Multiplexer

Information

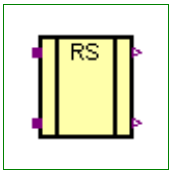
Parameters

Type	Name	Default	Description
Time	delayTime	0.001	[s]
Logic	q0	L.'0'	

Connectors

Type	Name	Description
input DigitalInput	d0	
input DigitalInput	d1	
input DigitalInput	d2	
input DigitalInput	d3	
input DigitalInput	a0	
input DigitalInput	a1	
output DigitalOutput	d	

Modelica.Electrical.Digital.Examples.Utilities.RS



Unclocked RS FlipFlop

Information

Parameters

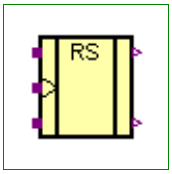
Type	Name	Default	Description
Time	delayTime	0	delay time [s]
Logic	q0	L.'U'	initial value of output

Connectors

Type	Name	Description
input DigitalInput	s	
input DigitalInput	r	
output DigitalOutput	q	

output DigitalOutput	qn	

Modelica.Electrical.Digital.Examples.Utilities.RSFF



Unclocked RS FlipFlop

Information

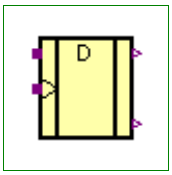
Parameters

Type	Name	Default	Description
Time	delayTime	0.01	[s]
Logic	q0	L.'U'	

Connectors

Type	Name	Description
input DigitalInput	s	
input DigitalInput	r	
output DigitalOutput	q	
output DigitalOutput	qn	not Q
input DigitalInput	clk	

Modelica.Electrical.Digital.Examples.Utilities.DFF



D FlipFlop

Information

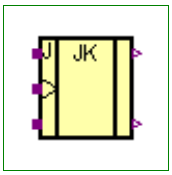
Parameters

Type	Name	Default	Description
Time	Tdel	0.01	[s]
Logic	QInit	L.'U'	

Connectors

Type	Name	Description
input DigitalInput	d	
output DigitalOutput	q	
output DigitalOutput	qn	not Q
input DigitalInput	clk	

Modelica.Electrical.Digital.Examples.Utilities.JKFF



JK FlipFlop

Information

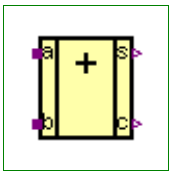
Parameters

Type	Name	Default	Description
Time	delayTime	0.001	[s]
Logic	q0	L.'0'	

Connectors

Type	Name	Description
input DigitalInput	j	
output DigitalOutput	q	
output DigitalOutput	qn	not Q
input DigitalInput	clk	
input DigitalInput	k	

Modelica.Electrical.Digital.Examples.Utilities.HalfAdder



Information

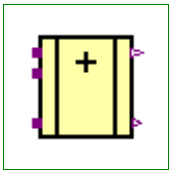
Parameters

Type	Name	Default	Description
Real	delayTime	0	

Connectors

Type	Name	Description
input DigitalInput	b	
output DigitalOutput	s	
input DigitalInput	a	
output DigitalOutput	c	

Modelica.Electrical.Digital.Examples.Utilities.FullAdder



adding circuit for binary numbers with input carry bit

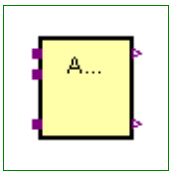
Information

a	b	c in	c out	s
1	1	1	0	
0	0	0	0	
1	0	0	1	
0	1	0	1	

Connectors

Type	Name	Description
input DigitalInput	a	
input DigitalInput	b	
input DigitalInput	c_in	
output DigitalOutput	s	
output DigitalOutput	c_out	

Modelica.Electrical.Digital.Examples.Utilities.Adder



Generic N Bit Adder

Information

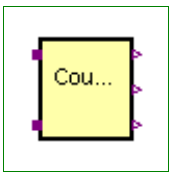
Parameters

Type	Name	Default	Description
Integer	n	2	

Connectors

Type	Name	Description
input DigitalInput	a[n]	
input DigitalInput	b[n]	
input DigitalInput	c_in	
output DigitalOutput	s[n]	
output DigitalOutput	c_out	

Modelica.Electrical.Digital.Examples.Utilities.Counter3



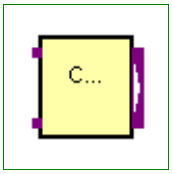
3 Bit Counter

Information

Connectors

Type	Name	Description
input DigitalInput	enable	
output DigitalOutput	q2	
input DigitalInput	count	
output DigitalOutput	q1	
output DigitalOutput	q0	

Modelica.Electrical.Digital.Examples.Utilities.Counter



Generic N Bit Counter

Information

Parameters

Type	Name	Default	Description
Integer	n	3	
Time	delayTime	0.001	[s]
Logic	q0	L.'0'	

Connectors

Type	Name	Description
input DigitalInput	enable	
input DigitalInput	count	
output DigitalOutput	q[n]	

Modelica.Electrical.Digital.Interfaces

Connectors for Digital electrical components

Information

This package contains interface definitions (connectors) digital electrical components.

Package Content

Name	Description
Logic	Signal type in package Digital according to the IEEE 1164 STD_ULOGIC type
LogicValue	Logic values and their coding
DigitalSignal	Digital port (both input/output possible)
DigitalInput	input DigitalSignal as connector
DigitalOutput	output DigitalSignal as connector
SISO	Single input, single output
MISO	Multiple input - single output

Types and constants

```
type Logic = Integer(min=1,max=9)
"Signal type in package Digital according to the IEEE 1164 STD_ULOGIC type";
```

Modelica.Electrical.Digital.Interfaces.LogicValue

Logic values and their coding

Information

Code Table:

Logic value	Integer code	Meaning

'U'	1	Uninitialized
'X'	2	Forcing Unknown
'0'	3	Forcing 0
'1'	4	Forcing 1
'Z'	5	High Impedance
'W'	6	Weak Unknown
'L'	7	Weak 0
'H'	8	Weak 1
'-'	9	Don't care

Modelica definition

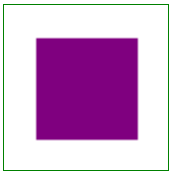
```
record LogicValue "Logic values and their coding"
  extends Modelica.Icons.Enumeration;

  constant Integer min=1;
  constant Integer max=9;
  constant Integer 'U'=1 "Uninitialized";
  constant Integer 'X'=2 "Forcing Unknown";
  constant Integer '0'=3 "Forcing 0";
  constant Integer '1'=4 "Forcing 1";
  constant Integer 'Z'=5 "High Impedance";
  constant Integer 'W'=6 "Weak    Unknown";
  constant Integer 'L'=7 "Weak    0";
  constant Integer 'H'=8 "Weak    1";
  constant Integer '-'=9 "Don't care";
end LogicValue;
```

Modelica.Electrical.Digital.Interfaces.DigitalSignal

Digital port (both input/output possible)

Information

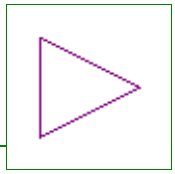
Modelica.Electrical.Digital.Interfaces.DigitalInput



input DigitalSignal as connector

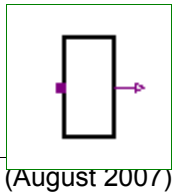
Information

Modelica.Electrical.Digital.Interfaces.DigitalOutput



output DigitalSignal as connector

Modelica.Electrical.Digital.Interfaces.SISO



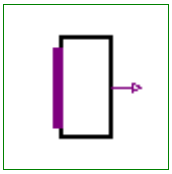
Single input, single output

Information

Connectors

Type	Name	Description
input DigitalInput	x	Connector of Digital input signal
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Interfaces.MISO



Multiple input - single output

Information

Parameters

Type	Name	Default	Description
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Tables

Truth tables for all components of package Digital

Information

Package Content

Name	Description
AndTable=[L.'U', L.'U', L.'0', L.'U', L.'U', L.'U', L.'0', L.'U', L.'U'; L.'U', L.'X', L.'0', L.'X', L.'X', L.'X', L.'0', L.'X', L.'X'; L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0'; L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X'; L.'U', L.'X', L.'0', L.'X', L.'X', L.'X', L.'0', L.'X', L.'X'; L.'U', L.'X', L.'0', L.'X', L.'X', L.'X', L.'0', L.'X', L.'X'; L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0'; L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X'; L.'U', L.'X', L.'0', L.'X', L.'X', L.'X', L.'0', L.'X', L.'X']	9-value logic for 'and'
OrTable=[L.'U', L.'U', L.'U', L.'1', L.'U', L.'U', L.'U', L.'1', L.'U'; L.'U', L.'X', L.'X', L.'1', L.'X', L.'X', L.'X', L.'1', L.'X'; L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X'; L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1'; L.'U', L.'X', L.'X', L.'1', L.'X', L.'X', L.'X', L.'1', L.'X'; L.'U', L.'X', L.'X', L.'1', L.'X', L.'X', L.'X', L.'1', L.'X'; L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X'; L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1'; L.'U', L.'X', L.'X', L.'1', L.'X', L.'X', L.'X', L.'1', L.'X']	9-value logic for 'or'
NotTable={L.'U',L.'X',L.'1',L.'0',L.'X',L.'X',L.'1',L.'0',L.'X'}	9-value logic for 'not'
XorTable=[L.'U', L.'U', L.'U', L.'U', L.'U', L.'U', L.'U', L.'U', L.'U'; L.'U', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X'; L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X'; L.'U', L.'X', L.'1', L.'0', L.'X', L.'X', L.'1', L.'0', L.'X'; L.'U', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X'; L.'U', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X'; L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X'; L.'U', L.'X', L.'1', L.'0', L.'X', L.'X', L.'1', L.'0', L.'X'; L.'U', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X']	9-value logic for 'xor'

X01Table={L.'X',L.'X',L.'0',L.'1',L.'X',L.'X',L.'0',L.'1',L.'X'}	
X01ZTable={L.'X',L.'X',L.'0',L.'1',L.'Z',L.'X',L.'0',L.'1',L.'Z'}	
UX01Table={L.'U',L.'X',L.'0',L.'1',L.'X',L.'X',L.'0',L.'1',L.'X'}	
DelayTable=[0, 0, -1, 1, 0, 0, -1, 1, 0; 0, 0, -1, 1, 0, 0, -1, 1, 0; 1, 1, 0, 1, 1, 1, 0, 1, 1; -1, -1, -1, 0, -1, -1, -1, 0, -1; 0, 0, -1, 1, 0, 0, -1, 1, 0; 0, 0, -1, 1, 0, 0, -1, 1, 0; 1, 1, 0, 1, 1, 1, 0, 1, 1; -1, -1, -1, 0, -1, -1, -1, 0, -1; 0, 0, -1, 1, 0, 0, -1, 1, 0]	

Types and constants

```
constant D.Interfaces.Logic AndTable[L.max, L.max]=[
    L.'U', L.'U', L.'0', L.'U', L.'U', L.'U', L.'0', L.'U', L.'U';
    L.'U', L.'X', L.'0', L.'X', L.'X', L.'X', L.'0', L.'X', L.'X';
    L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0';
    L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X';
    L.'U', L.'X', L.'0', L.'X', L.'X', L.'X', L.'0', L.'X', L.'X';
    L.'U', L.'X', L.'0', L.'X', L.'X', L.'X', L.'0', L.'X', L.'X';
    L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0', L.'0';
    L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X';
    L.'U', L.'X', L.'0', L.'X', L.'X', L.'X', L.'0', L.'X', L.'X']
"9-value logic for 'and'";

constant D.Interfaces.Logic OrTable[L.max, L.max]=[
    L.'U', L.'U', L.'U', L.'1', L.'U', L.'U', L.'U', L.'1', L.'U';
    L.'U', L.'X', L.'X', L.'1', L.'X', L.'X', L.'X', L.'1', L.'X';
    L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X';
    L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1';
    L.'U', L.'X', L.'X', L.'1', L.'X', L.'X', L.'X', L.'1', L.'X';
    L.'U', L.'X', L.'X', L.'1', L.'X', L.'X', L.'X', L.'1', L.'X';
    L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X';
    L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1', L.'1';
    L.'U', L.'X', L.'X', L.'1', L.'X', L.'X', L.'X', L.'1', L.'X']
"9-value logic for 'or'";

constant D.Interfaces.Logic NotTable[L.max]={
    L.'U',L.'X',L.'1',L.'0',L.'X',L.'X',L.'1',L.'0',L.'X'}
"9-value logic for 'not'";

constant D.Interfaces.Logic XorTable[L.max, L.max]=[
    L.'U', L.'U', L.'U', L.'U', L.'U', L.'U', L.'U', L.'U', L.'U';
    L.'U', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X';
    L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X';
    L.'U', L.'X', L.'1', L.'0', L.'X', L.'X', L.'1', L.'0', L.'X';
    L.'U', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X';
    L.'U', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X';
    L.'U', L.'X', L.'0', L.'1', L.'X', L.'X', L.'0', L.'1', L.'X';
    L.'U', L.'X', L.'1', L.'0', L.'X', L.'X', L.'1', L.'0', L.'X';
    L.'U', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X', L.'X']
"9-value logic for 'xor'";

constant D.Interfaces.Logic X01Table[L.max]={
    L.'X',L.'X',L.'0',L.'1',L.'X',L.'X',L.'0',L.'1',L.'X'};

constant D.Interfaces.Logic X01ZTable[L.max]={
    L.'X',L.'X',L.'0',L.'1',L.'Z',L.'X',L.'0',L.'1',L.'Z'};

constant D.Interfaces.Logic UX01Table[L.max]={
```

```
      L.'U',L.'X',L.'0',L.'1',L.'X',L.'X',L.'0',L.'1',L.'X'};

  constant Integer DelayTable[9, 9]=[
      0, 0, -1, 1, 0, 0, -1, 1, 0;
      0, 0, -1, 1, 0, 0, -1, 1, 0;
      1, 1, 0, 1, 1, 1, 0, 1, 1;
      -1, -1, -1, 0, -1, -1, -1, 0, -1;
      0, 0, -1, 1, 0, 0, -1, 1, 0;
      0, 0, -1, 1, 0, 0, -1, 1, 0;
      1, 1, 0, 1, 1, 1, 0, 1, 1;
      -1, -1, -1, 0, -1, -1, -1, 0, -1;
      0, 0, -1, 1, 0, 0, -1, 1, 0];
```

Modelica.Electrical.Digital.Delay

Transport and inertial delay blocks

Information

Package Content

Name	Description
DelayParams	Definition of delay parameters
TransportDelay	Transport delay with initial parameter
InertialDelay	Inertial delay with initial parameter
InertialDelaySensitive	Provide the input as output if it holds its value for a specific amount of time

Modelica.Electrical.Digital.Delay.DelayParams

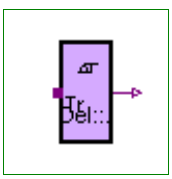
Definition of delay parameters

Information

Parameters

Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output

Modelica.Electrical.Digital.Delay.TransportDelay



Transport delay with initial parameter

Information

Provide the input as output exactly delayed by *Tdel*. If time less than *Tdel* the initial value *initout* holds.

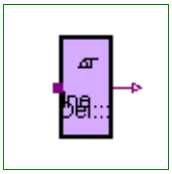
Parameters

Type	Name	Default	Description
Time	delayTime	0	delay time [s]
Logic	y0	L.'U'	initial value of output

Connectors

Type	Name	Description
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Delay.InertialDelay



Inertial delay with initial parameter

Information

Provides the input as output delayed by *Tdel* if the input holds its value for a longer time than *Tdel*. If time is less than *Tdel* the initial value *initout* holds.

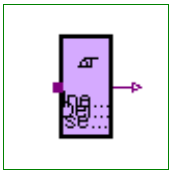
Parameters

Type	Name	Default	Description
Time	delayTime	0	Minimum time to hold value [s]
Logic	y0	L.'U'	Initial value of output y

Connectors

Type	Name	Description
input DigitalInput	x	Connector of Digital input signal
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Delay.InertialDelaySensitive



Provide the input as output if it holds its value for a specific amount of time

Information

Provides the input as output delayed by *Tdel* if the input holds its value for a longer time than *Tdel*. If the time is less than *Tdel* the initial value *initout* holds.
The delay *Tdel* depends on the values of the signal change. To calculate *Tdel*, the delaymap specified in Digital.Tables is used. If the corresponding value is 1, then *tLH* is used, if it is -1, then *tHL* is used, if it is zero, the input is not delayed.

Parameters

Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output

Connectors

Type	Name	Description
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Basic

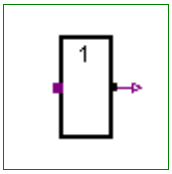
Basic logic blocks without delays

Information

Package Content

Name	Description
Not	Not Logic
And	And logic with multiple input and one output
Nand	Nand logic with multiple input and one output
Or	Or logic with multiple input and one output
Nor	Nor logic with multiple input and one output
Xor	Xor logic with multiple input and one output
Xnor	Xnor logic with multiple input and one output

Modelica.Electrical.Digital.Basic.Not



Not Logic

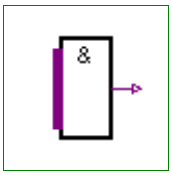
Information

Not with 1 input value, without delay.

Connectors

Type	Name	Description
input DigitalInput	x	Connector of Digital input signal
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Basic.And



And logic with multiple input and one output

Information

And with n input values, without delay.

Parameters

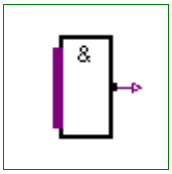
Type	Name	Default	Description

Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Basic.Nand



Nand logic with multiple input and one output

Information

Nand with n input values, without delay.

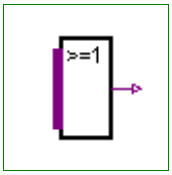
Parameters

Type	Name	Default	Description
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Basic.Or



Or logic with multiple input and one output

Information

Or with n input values, without delay.

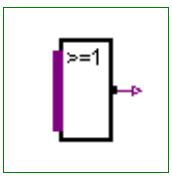
Parameters

Type	Name	Default	Description
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Basic.Nor



Nor logic with multiple input and one output

Information

Nor with n input values, without delay.

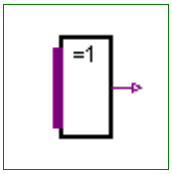
Parameters

Type	Name	Default	Description
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Basic.Xor



Xor logic with multiple input and one output

Information

Xor with n input values, without delay.

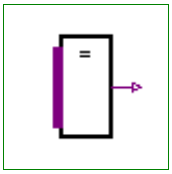
Parameters

Type	Name	Default	Description
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Basic.Xnor



Xnor logic with multiple input and one output

Information

XNor with n input values, without delay.

Parameters

Type	Name	Default	Description
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Gates

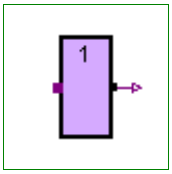
Logic gates including delays

Information

Package Content

Name	Description
InvGate	InvGate with 1 input value, composed by Not and sensitive intertial delay
AndGate	AndGate with multiple input
NandGate	NandGate with multiple input
OrGate	OrGate with multiple input
NorGate	NorGate with multiple input
XorGate	XorGate with multiple input
XnorGate	XnorGate with multiple input
BufGate	BufGate with 1 input value, composed by Not and sensitive intertial delay

Modelica.Electrical.Digital.Gates.InvGate



InvGate with 1 input value, composed by Not and sensitive intertial delay

Information

InvGate with 1 input value, composed by Not and sensitive intertial delay.

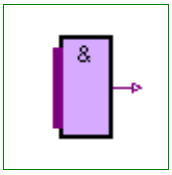
Parameters

Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output

Connectors

Type	Name	Description
input DigitalInput	x	Connector of Digital input signal
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Gates.AndGate



AndGate with multiple input

Information

AndGate with n input values, composed by And and sensitive intertial delay.

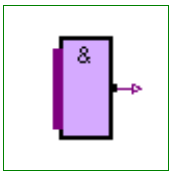
Parameters

Type	Name	Default	Description
Integer	n	2	Number of inputs
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Gates.NandGate



NandGate with multiple input

Information

NandGate with n input values, composed by Nand and sensitive intertial delay.

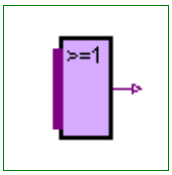
Parameters

Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Gates.OrGate



OrGate with multiple input

Information

OrGate with n input values, composed by Or and sensitive intertial delay.

Parameters

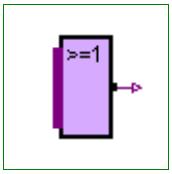
Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output

Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Gates.NorGate



NorGate with multiple input

Information

NorGate with n input values, composed by Nor and sensitive intertial delay.

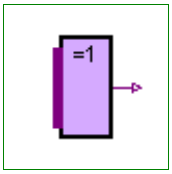
Parameters

Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Gates.XorGate



XorGate with multiple input

Information

XorGate with n input values, composed by Xor and sensitive intertial delay.

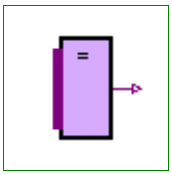
Parameters

Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Gates.XnorGate



XnorGate with multiple input

Information

XNorGate with n input values, composed by XNor and sensitive intertial delay.

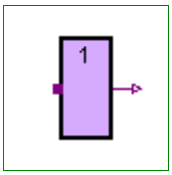
Parameters

Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output
Integer	n	2	Number of inputs

Connectors

Type	Name	Description
input DigitalInput	x[n]	Connector of Digital input signal vector
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Gates.BufGate



BufGate with 1 input value, composed by Not and sensitive intertial delay

Information

BufGate with 1 input value, composed by Not and sensitive intertial delay.

Parameters

Type	Name	Default	Description
Time	tLH	0	rise inertial delay [s]
Time	tHL	0	fall inertial delay [s]
Logic	y0	L.'U'	initial value of output

Connectors

Type	Name	Description
input DigitalInput	x	Connector of Digital input signal
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Sources

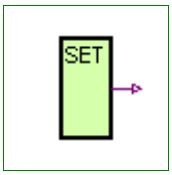
Time-dependend digital signal sources

Information

Package Content

Name	Description
Set	Digital Set Source
Step	Digital Step Source
Table	Digital Tabular Source
Pulse	Digital Pulse Source
Clock	Digital Clock Source

Modelica.Electrical.Digital.Sources.Set



Digital Set Source

Information

Sets a nine valued digital signal, which is specified by the *setval* parameter.

To specify *setval*, the integer code has to be used.

Code Table

Logic value	Integer code	Meaning
'U'	1	Uninitialized
'X'	2	Forcing Unknown
'0'	3	Forcing 0
'1'	4	Forcing 1
'Z'	5	High Impedance
'W'	6	Weak Unknown
'L'	7	Weak 0
'H'	8	Weak 1
'-'	9	Don't care

If the logic values are imported by
import L = Modelica.Electrical.Digital.Interfaces.LogicValue;
they can be used to specify the parameter, e.g. **L.'0'** for forcing 0.

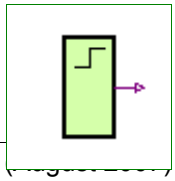
Parameters

Type	Name	Default	Description
Logic	x	L.'1'	Logic value to be set

Connectors

Type	Name	Description
output DigitalOutput	y	

Modelica.Electrical.Digital.Sources.Step



Digital Step Source

Information

The step source output signal steps from the value *before* to the value *after* at the time *stepTime*.

To specify the logic value parameters, the integer code has to be used.

Code Table

Logic value	Integer code	Meaning
'U'	1	Uninitialized
'X'	2	Forcing Unknown
'0'	3	Forcing 0
'1'	4	Forcing 1
'Z'	5	High Impedance
'W'	6	Weak Unknown
'L'	7	Weak 0
'H'	8	Weak 1
'-'	9	Don't care

If the logic values are imported by
import L = Modelica.Electrical.Digital.Interfaces.LogicValue;
they can be used to specify the parameter, e.g. **L.'0'** for forcing 0.

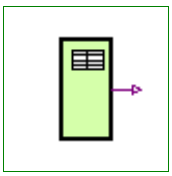
Parameters

Type	Name	Default	Description
Logic	before	L.'0'	Logic value before step
Logic	after	L.'1'	Logic value after step
Real	stepTime	1	step time

Connectors

Type	Name	Description
output DigitalOutput	y	

Modelica.Electrical.Digital.Sources.Table



Digital Tabular Source

Information

The table source output signal *y* steps to the values of the *x* table at the corresponding timepoints in the *t* table.
The initial value is specified by *y0*.

To specify the logic value parameters, the integer code has to be used.

Code Table

Logic value	Integer code	Meaning
'U'	1	Uninitialized
'X'	2	Forcing Unknown
'0'	3	Forcing 0
'1'	4	Forcing 1
'Z'	5	High Impedance

'W'	6	Weak Unknown
'L'	7	Weak 0
'H'	8	Weak 1
'-'	9	Don't care

If the logic values are imported by
import L = Modelica.Electrical.Digital.Interfaces.LogicValue;
they can be used to specify the parameter, e.g. **L.'0'** for forcing 0.

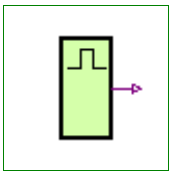
Parameters

Type	Name	Default	Description
Logic	x[:]	{1}	
Real	t[size(x, 1)]	{1}	
Logic	y0	L.'U'	

Connectors

Type	Name	Description
output DigitalOutput	y	

Modelica.Electrical.Digital.Sources.Pulse



Digital Pulse Source

Information

The pulse source forms pulses between the *quiet* value and the *pulse* value. The pulse length *width* is specified in percent of the period length *period*. The number of periods is specified by *nperiod*. If *nperiod* is less than zero, the number of periods is unlimited.

To specify the logic value parameters, the integer code has to be used.

Code Table

Logic value	Integer code	Meaning
'U'	1	Uninitialized
'X'	2	Forcing Unknown
'0'	3	Forcing 0
'1'	4	Forcing 1
'Z'	5	High Impedance
'W'	6	Weak Unknown
'L'	7	Weak 0
'H'	8	Weak 1
'-'	9	Don't care

If the logic values are imported by
import L = Modelica.Electrical.Digital.Interfaces.LogicValue;
they can be used to specify the parameter, e.g. **L.'0'** for forcing 0.

Parameters

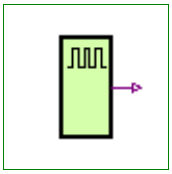
Type	Name	Default	Description
Real	width	50	Widths of pulses in % of periods

Time	period	1	Time for one period [s]
Time	startTime	0	Output = offset for time < startTime [s]
Logic	pulse	L.'0'	
Logic	quiet	L.'1'	
Integer	nperiod	-1	Number of periods (< 0 means infinite number of periods)

Connectors

Type	Name	Description
output DigitalOutput	y	

Modelica.Electrical.Digital.Sources.Clock



Digital Clock Source

Information

The clock source forms pulses between the *'0'* value (forcing 0) and the *'1'* value (forcing 1). The pulse length *width* is specified in percent of the period length *period*. The number of periods is unlimited. The first pulse starts at startTime.

The clock source is a special but often used variant of the pulse source.

Parameters

Type	Name	Default	Description
Time	startTime	0	Output = offset for time < startTime [s]
Time	period	1	Time for one period [s]
Real	width	50	Width of pulses in % of period

Connectors

Type	Name	Description
output DigitalOutput	y	Connector of Digital output signal

Modelica.Electrical.Digital.Converters

Converters between 2-,3-,4- and 9-valued logic

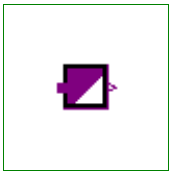
Information

Package Content

Name	Description
LogicToXO1	Conversion to XO1
LogicToXO1Z	Conversion to XO1Z
LogicToUX01	Conversion to UXO1
BooleanToLogic	Boolean to Logic converter
LogicToBoolean	Logic to Boolean converter

RealToLogic	Real to Logic converter
LogicToReal	Logic to Real converter

Modelica.Electrical.Digital.Converters.LogicToXO1



Conversion to XO1

Information

Conversion of a nine valued digital input into a X01 digital output without any delay according to IEEE 1164 To_X01 function.

Conversion Table:

```
input                  output
'U' (coded by 1)       'X'  (coded by 2)
'X' (coded by 2)       'X'  (coded by 2)
'0' (coded by 3)       '0'  (coded by 3)
'1' (coded by 4)       '1'  (coded by 4)
'Z' (coded by 5)       'X'  (coded by 2)
'W' (coded by 6)       'X'  (coded by 2)
'L' (coded by 7)       '0'  (coded by 3)
'H' (coded by 8)       '1'  (coded by 4)
'-' (coded by 9)       'X'  (coded by 2)
```

If the signal width is greater than 1 this conversion is done for each signal.

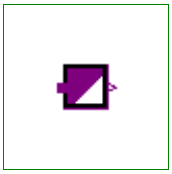
Parameters

Type	Name	Default	Description
Integer	n	1	signal width

Connectors

Type	Name	Description
input DigitalInput	x[n]	
output DigitalOutput	y[n]	

Modelica.Electrical.Digital.Converters.LogicToXO1Z



Conversion to XO1Z

Information

Conversion of a nine valued digital input into a X01Z digital output without any delay according to IEEE 1164 To_X01Z function.

Conversion Table:

```
input                  output
'U' (coded by 1)       'X'  (coded by 2)
'X' (coded by 2)       'X'  (coded by 2)
'0' (coded by 3)       '0'  (coded by 3)
'1' (coded by 4)       '1'  (coded by 4)
'Z' (coded by 5)       'Z'  (coded by 5)
```

```
'W' (coded by 6)        'X'  (coded by 2)
'L' (coded by 7)        '0'  (coded by 3)
'H' (coded by 8)        '1'  (coded by 4)
'-' (coded by 9)        'X'  (coded by 2)
```

If the signal width is greater than 1 this conversion is done for each signal.

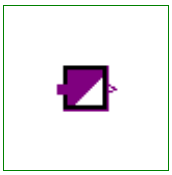
Parameters

Type	Name	Default	Description
Integer	n	1	signal width

Connectors

Type	Name	Description
input DigitalInput	x[n]	
output DigitalOutput	y[n]	

Modelica.Electrical.Digital.Converters.LogicToUX01



Conversion to UXO1

Information

Conversion of a nine valued digital input into a UX01 digital output without any delay according to IEEE 1164 To_UX01 function.

Conversion Table:

```
input                   output
'U' (coded by 1)        'U'  (coded by 1)
'X' (coded by 2)        'X'  (coded by 2)
'0' (coded by 3)        '0'  (coded by 3)
'1' (coded by 4)        '1'  (coded by 4)
'Z' (coded by 5)        'X'  (coded by 2)
'W' (coded by 6)        'X'  (coded by 2)
'L' (coded by 7)        '0'  (coded by 3)
'H' (coded by 8)        '1'  (coded by 4)
'-' (coded by 9)        'X'  (coded by 2)
```

If the signal width is greater than 1 this conversion is done for each signal.

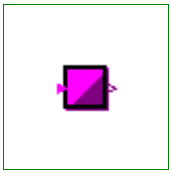
Parameters

Type	Name	Default	Description
Integer	n	1	signal width

Connectors

Type	Name	Description
input DigitalInput	x[n]	
output DigitalOutput	y[n]	

Modelica.Electrical.Digital.Converters.BooleanToLogic



Boolean to Logic converter

Information

Conversion of a Boolean input into a digital output without any delay according to:

```
input      output
true       '1'   (coded by 4)
false      '0'   (coded by 3)
```

If the signal width is greater than 1 this conversion is done for each signal.

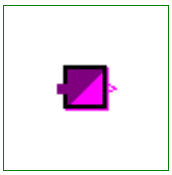
Parameters

Type	Name	Default	Description
Integer	n	1	signal width

Connectors

Type	Name	Description
input BooleanInput	x[n]	
output DigitalOutput	y[n]	

Modelica.Electrical.Digital.Converters.LogicToBoolean



Logic to Boolean converter

Information

Conversion of a digital input into a Boolean output without any delay according to:

```
input                  output
'U'   (coded by 1)     false
'X'   (coded by 2)     false
'0'   (coded by 3)     false
'1'   (coded by 4)     true
'Z'   (coded by 5)     false
'W'   (coded by 6)     false
'L'   (coded by 7)     false
'H'   (coded by 8)     true
'-'   (coded by 9)     false
```

If the signal width is greater than 1 this conversion is done for each signal.

Parameters

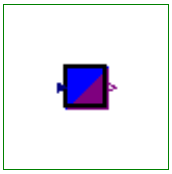
Type	Name	Default	Description
Integer	n	1	signal width

Connectors

Type	Name	Description
input DigitalInput	x[n]	

output BooleanOutput	y[n]	

Modelica.Electrical.Digital.Converters.RealToLogic



Real to Logic converter

Information

Conversion of a real input into a digital output without any delay according to:

```
                              condition            output
      first check:            input greater upp    lupp
      second check:           input larger low     llow
                              else                 lmid
```

If the signal width is greater than 1 this conversion is done for each signal.

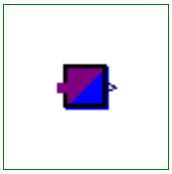
Parameters

Type	Name	Default	Description
Integer	n	1	signal width
Real	upper_limit	1	upper limit
Real	lower_limit	0	lower limit
Logic	upper_value	L.'1'	output if input > upper_limit
Logic	lower_value	L.'0'	output if input < lower_limit
Logic	middle_value	L.'X'	output else

Connectors

Type	Name	Description
input RealInput	x[n]	
output DigitalOutput	y[n]	

Modelica.Electrical.Digital.Converters.LogicToReal



Logic to Real converter

Information

Conversion of a digital input into a Real output without any delay according to:

```
input                  output
'U'   (coded by 1)     val_U
'X'   (coded by 2)     val_X
'0'   (coded by 3)     val_0
'1'   (coded by 4)     val_1
'Z'   (coded by 5)     val_Z
'W'   (coded by 6)     val_W
'L'   (coded by 7)     val_L
'H'   (coded by 8)     val_H
'-'   (coded by 9)     val_m
```

The values val... are given by parameters.

If the signal width is greater than 1 this conversion is done for each signal.

Parameters

Type	Name	Default	Description
Integer	n	1	signal width
Real	value_U	0.5	value for digital U (uninitialized)
Real	value_X	0.5	value for digital X (Forcing Unknown)
Real	value_0	0	value for digital 0 (Forcing 0)
Real	value_1	1	value for digital 1 (Forcing 1)
Real	value_Z	0.5	value for digital Z (High Impedance)
Real	value_W	0.5	value for digital W (Weak Unknown)
Real	value_L	0	value for digital L (Weak 0)
Real	value_H	1	value for digital H (Weak 1)
Real	value_m	0.5	value for digital m (Don´t care)

Connectors

Type	Name	Description
input DigitalInput	x[n]	
output RealOutput	y[n]	

Modelica.Electrical.Machines

Library for electric machines

Information

This package contains components to model electrical machines:

- Examples: test examples
- BasicMachines: basic machine models
- Sensors: sensors, usefull when modelling machines
- SpacePhasors: an independent library for using space phasors
- Interfaces: Space phasor connector and partial machine models

Limitations and assumptions:

- number of phases (of induction machines) is limited to 3, therefore definition as a constant m=3
- phase symmetric windings as well as symmetry of the whole machine structure
- all values are used in physical units, no scaling to p.u. is done
- only basic harmonics (in space) are taken into account
- waveform (with respect to time) of voltages and currents is not restricted
- constant parameters, i.e. no saturation, no skin effect
- no iron losses, eddy currents, friction losses;
  only ohmic losses in stator and rotor winding

You may have a look at a short summary of space phasor theory at
http://www.haumer.at/refimg/SpacePhasors.pdf

Further development:

- generalizing space phasor theory to m phases with arbitrary spatial angle of the coils
- generalizing space phasor theory to arbitrary number of windings and winding factor of the coils
- MachineModels: other machine types
- effects: saturation, skin-effect, other losses than ohmic, ...

Main Authors:
Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern
Austria
email: a.haumer@haumer.at

Copyright © 1998-2007, Modelica Association and Anton Haumer.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Examples	Test examples
BasicMachines	Basic machine models
Sensors	Sensors for machine modelling
SpacePhasors	Library with space phasor-models
Interfaces	SpacePhasor connector and PartialMachines

Modelica.Electrical.Machines.Examples

Test examples

Information

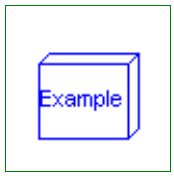
This package contains test examples of electric machines,
and a package utilities with components used for the examples.

Package Content

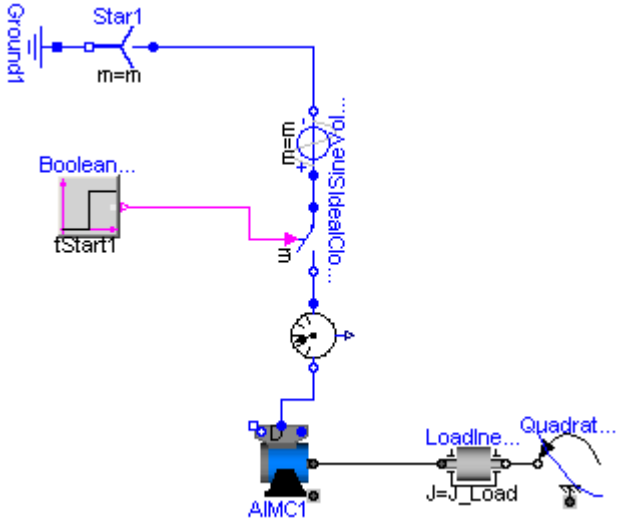
Name	Description
AIMC_DOL	Test example 1: AsynchronousInductionMachineSquirrelCage direct-on-line
AIMC_YD	Test example 2: AsynchronousInductionMachineSquirrelCage Y-D
AIMS_start	Test example 3: AsynchronousInductionMachineSlipRing
AIMC_Inverter	Test example 4: AsynchronousInductionMachineSquirrelCage with inverter
SMR_Inverter	Test example 5: SynchronousInductionMachineReluctanceRotor with inverter
SMPM_Inverter	Test example 6: PermanentMagnetSynchronousInductionMachine with inverter
SMEE_Gen	Test example 7: ElectricalExcitedSynchronousInductionMachine as Generator
DCPM_start	Test example 8: DC with permanent magnet starting with voltage ramp
DCEE_start	Test example 9: DC with electrical ecxitation starting with voltage ramp
DCSE_start	Test example 10: DC with serial excitation starting with voltage ramp
TransformerTestbench	Transformer Testbench
Rectifier6pulse	6-pulse rectifier with 1 transformer
Rectifier12pulse	12-pulse rectifier with 2 transformers

AIMC_Steinmetz	AsynchronousInductionMachineSquirrelCage Steinmetz-connection
Utilities	Library with auxiliary models for testing

Modelica.Electrical.Machines.Examples.AIMC_DOL



Test example 1: AsynchronousInductionMachineSquirrelCage direct-on-line



Information

1st Test example: Asynchronous induction machine with squirrel cage - direct on line starting
At start time tStart three phase voltage is supplied to the asynchronous induction machine with squirrel cage; the machine starts from standstill, accelerating inertias against load torque quadratic dependent on speed, finally reaching nominal speed.
Simulate for 1.5 seconds and plot (versus time):

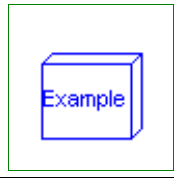
- CurrentRMSsensor1.I: stator current RMS
- AIMC1.rpm_mechanical: motor's speed
- AIMC1.tau_electrical: motor's torque

Default machine parameters of model *AIM_SquirrelCage* are used.

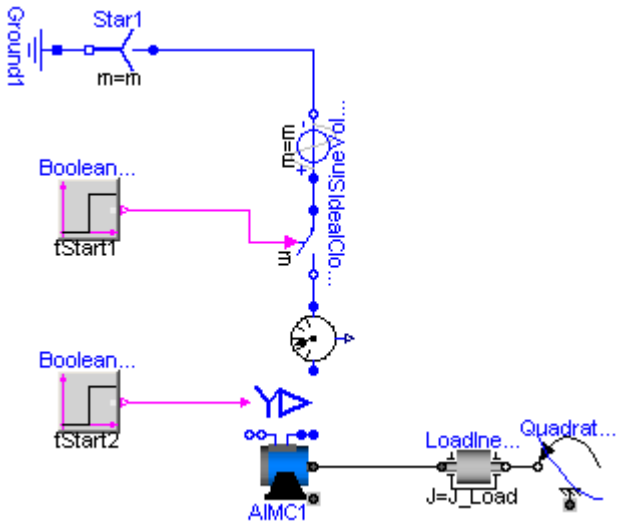
Parameters

Type	Name	Default	Description
Voltage	VNominal	100	nominal RMS voltage per phase [V]
Frequency	fNominal	50	nominal frequency [Hz]
Time	tStart1	0.1	start time [s]
Torque	T_Load	161.4	nominal load torque [N.m]
AngularVelocity_rpm	rpmLoad	1440.45	nominal load speed [rev/min]
Inertia	J_Load	0.29	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.AIMC_YD



Test example 2: AsynchronousInductionMachineSquirrelCage Y-D



Information

2nd Test example: Asynchronous induction machine with squirrel cage - Y-D starting
At start time tStart three phase voltage is supplied to the asynchronous induction machine with squirrel cage, first star-connected, then delta-connetced; the machine starts from standstill, accelerating inertias against load torque quadratic dependent on speed, finally reaching nominal speed.
Simulate for 2.5 seconds and plot (versus time):

- CurrentRMSsensor1.I: stator current RMS
- AIMC1.rpm_mechanical: motor's speed
- AIMC1.tau_electrical: motor's torque

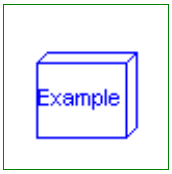
Default machine parameters of model *AIM_SquirrelCage* are used.

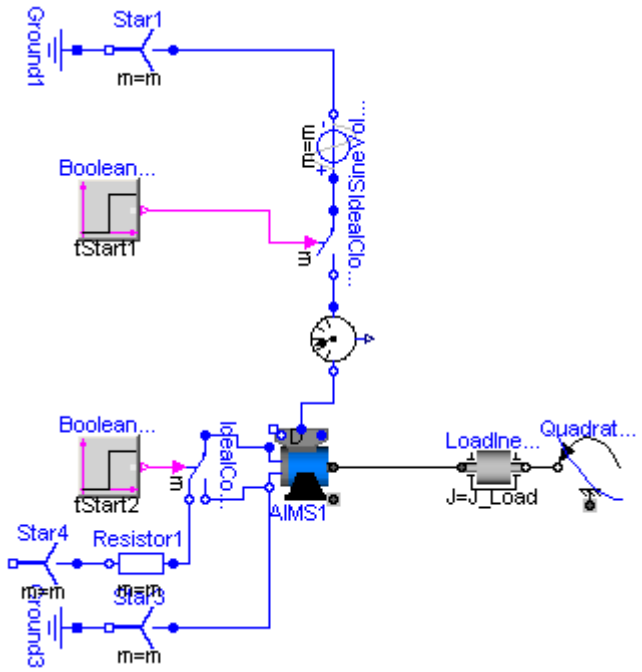
Parameters

Type	Name	Default	Description
Voltage	VNominal	100	nominal RMS voltage per phase [V]
Frequency	fNominal	50	nominal frequency [Hz]
Time	tStart1	0.1	start time [s]
Time	tStart2	2.0	2nd start time [s]
Torque	T_Load	161.4	nominal load torque [N.m]
AngularVelocity_rpm	rpmLoad	1440.45	nominal load speed [rev/min]
Inertia	J_Load	0.29	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.AIMS_start

Test example 3: AsynchronousInductionMachineSlipRing





Information

3rd Test example: Asynchronous induction machine with slipring rotor - resistance starting
At start time tStart1 three phase voltage is supplied to the asynchronous induction machine with sliprings; the machine starts from standstill, accelerating inertias against load torque quadratic dependent on speed, using a starting resistance. At time tStart2 external rotor resistance is shortened, finally reaching nominal speed. Simulate for 1.5 seconds and plot (versus time):

- CurrentRMSsensor1.I: stator current RMS
- AIMS1.rpm_mechanical: motor's speed
- AIMS1.tau_electrical: motor's torque

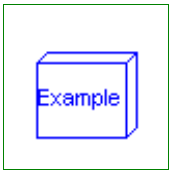
Default machine parameters of model *AIM_SlipRing* are used.

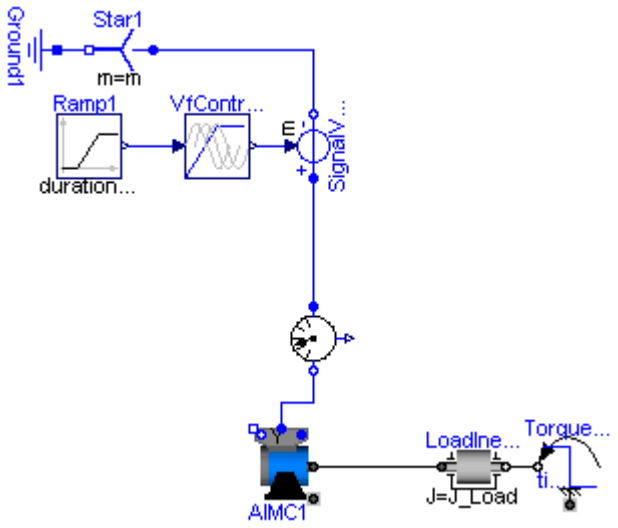
Parameters

Type	Name	Default	Description
Voltage	VNominal	100	nominal RMS voltage per phase [V]
Frequency	fNominal	50	nominal frequency [Hz]
Time	tStart1	0.1	1st start time [s]
Resistance	Rstart	0.16	starting resistance [Ohm]
Time	tStart2	1.0	2nd start time [s]
Torque	T_Load	161.4	nominal load torque [N.m]
AngularVelocity_rpm	rpmLoad	1440.45	nominal load speed [rev/min]
Inertia	J_Load	0.29	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.AIMC_Inverter

Test example 4: AsynchronousInductionMachineSquirrelCage with inverter





Information

4th Test example: Asynchronous induction machine with squirrel cage fed by an ideal inverter
An ideal frequency inverter is modeled by using a VfController and a threephase SignalVoltage.
Frequency is raised by a ramp, causing the asynchronous induction machine with squirrel cage to start, and accelerating inertias.
At time tStep a load step is applied.
Simulate for 1.5 seconds and plot (versus time):

- CurrentRMSsensor1.I: stator current RMS
- AIMC1.rpm_mechanical: motor's speed
- AIMC1.tau_electrical: motor's torque

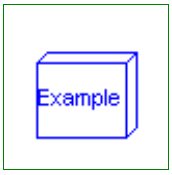
Default machine parameters of model *AIM_SquirrelCage* are used.

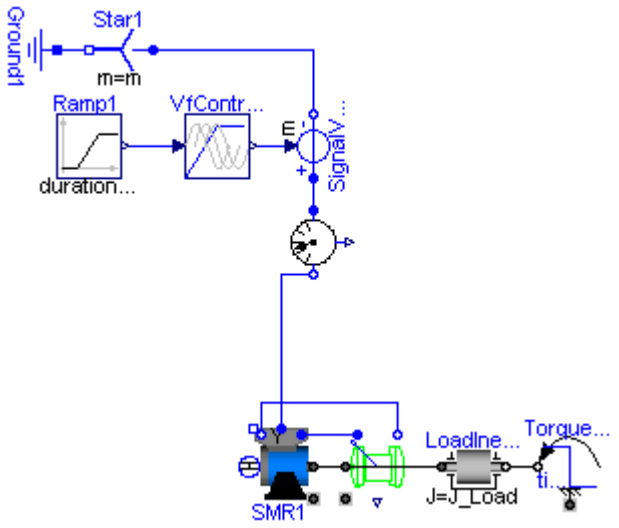
Parameters

Type	Name	Default	Description
Voltage	VNominal	100	nominal RMS voltage per phase [V]
Frequency	fNominal	50	nominal frequency [Hz]
Frequency	f	50	actual frequency [Hz]
Time	tRamp	1	frequency ramp [s]
Torque	T_Load	161.4	nominal load torque [N.m]
Time	tStep	1.2	time of load torque step [s]
Inertia	J_Load	0.29	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.SMR_Inverter

Test example 5: SynchronousInductionMachineReluctanceRotor with inverter





Information

5th Test example: Synchronous induction machine with reluctance rotor fed by an ideal inverter
An ideal frequency inverter is modeled by using a VfController and a threephase SignalVoltage.
Frequency is raised by a ramp, causing the reluctance machine to start, and accelerating inertias.
At time tStep a load step is applied.
Simulate for 1.5 seconds and plot (versus time):

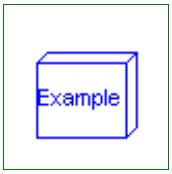
- CurrentRMSsensor1.I: stator current RMS
- SMRD1.rpm_mechanical: motor's speed
- SMRD1.tau_electrical: motor's torque
- RotorAngle.rotorAngle: rotor displacement angle

Default machine parameters of model *SM_ReluctanceRotorDamperCage* are used.

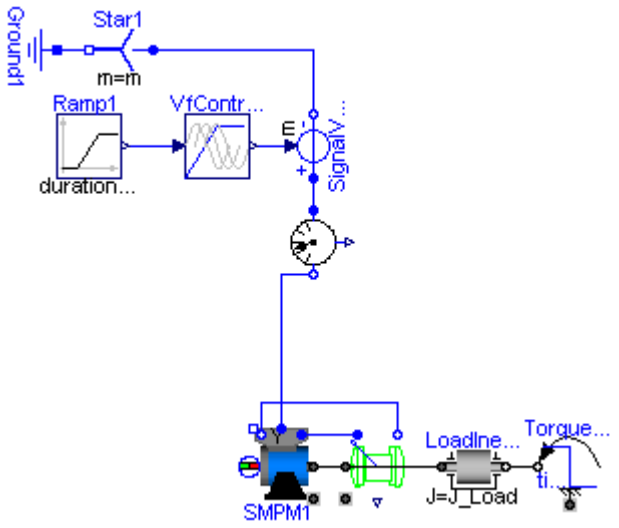
Parameters

Type	Name	Default	Description
Voltage	VNominal	100	nominal RMS voltage per phase [V]
Frequency	fNominal	50	nominal frequency [Hz]
Frequency	f	50	actual frequency [Hz]
Time	tRamp	1	frequency ramp [s]
Torque	T_Load	46	nominal load torque [N.m]
Time	tStep	1.2	time of load torque step [s]
Inertia	J_Load	0.29	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.SMPM_Inverter



Test example 6: PermanentMagnetSynchronousInductionMachine with inverter



Information

6th Test example: Permanent magnet synchronous induction machine fed by an ideal inverter
An ideal frequency inverter is modeled by using a VfController and a threephase SignalVoltage.
Frequency is raised by a ramp, causing the permanent magnet synchronous induction machine to start, and accelerating inertias.
At time tStep a load step is applied.
Simulate for 1.5 seconds and plot (versus time):

- CurrentRMSsensor1.I: stator current RMS
- PMSMD1.rpm_mechanical: motor's speed
- PMSMD1.tau_electrical: motor's torque
- RotorAngle.rotorAngle: rotor displacement angle

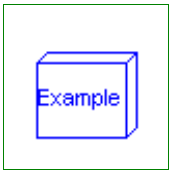
Default machine parameters of model *SM_PermanentMagnetDamperCage* are used.

In practice it is nearly impossible to drive a PMSMD without current controller.

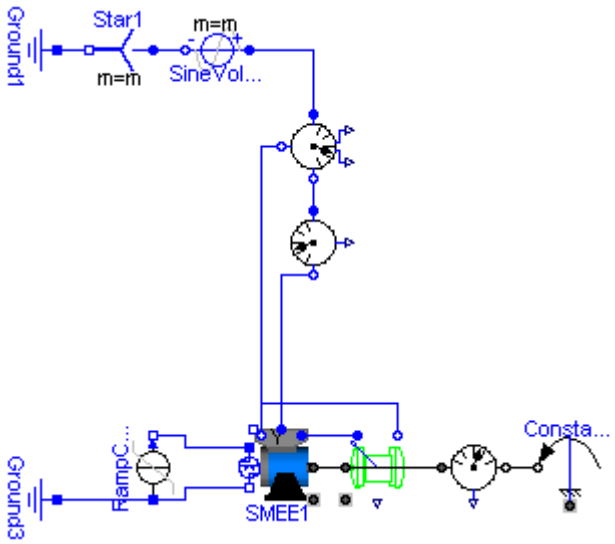
Parameters

Type	Name	Default	Description
Voltage	VNominal	100	nominal RMS voltage per phase [V]
Frequency	fNominal	50	nominal frequency [Hz]
Frequency	f	50	actual frequency [Hz]
Time	tRamp	1	frequency ramp [s]
Torque	T_Load	181.4	nominal load torque [N.m]
Time	tStep	1.2	time of load torque step [s]
Inertia	J_Load	0.29	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.SMEE_Gen



Test example 7: ElectricalExcitedSynchronousInductionMachine as Generator



Information

7th Test example: Electrical excited synchronous induction machine as generator
An electrically excited synchronous generator is connected to the grid and driven with constant speed. Since speed is slightly smaller than synchronous speed corresponding to mains frequency, rotor angle is very slowly increased. This allows to see several charactersistics dependent on rotor angle. Simulate for 30 seconds and plot (versus RotorAngle1.rotorAngle):

- SMEED1.tau_electrical
- CurrentRMSsensor1.I
- ElectricalPowerSensor1.P
- ElectricalPowerSensor1.Q
- MechanicalPowerSensor1.P

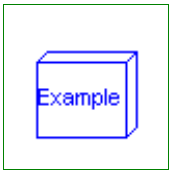
Default machine parameters of model *SM_ElectricalExcitedDamperCage* are used.

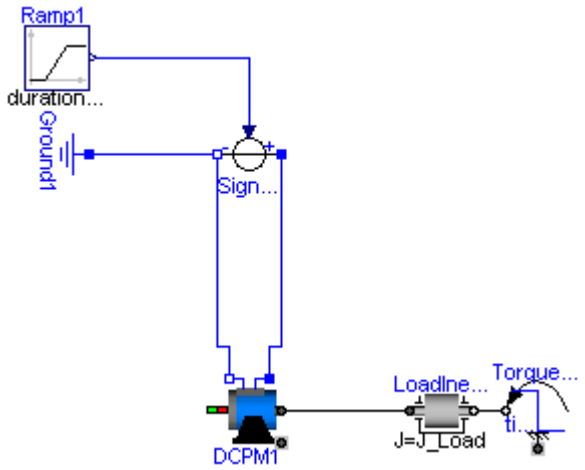
Parameters

Type	Name	Default	Description
Voltage	VNominal	100	nominal RMS voltage per phase [V]
Frequency	fNominal	50	nominal frequency [Hz]
AngularVelocity_rpm	rpm	1499	nominal speed [rev/min]
Current	Ie	19	excitation current [A]
Current	Ie0	10	initial excitation current [A]
Angle_deg	gamma0	0	initial rotor displacement angle [deg]

Modelica.Electrical.Machines.Examples.DCPM_start

Test example 8: DC with permanent magnet starting with voltage ramp





Information

8th Test example: Permanent magnet DC machine started with an armature voltage ramp
A voltage ramp is applied to the armature, causing the DC machine to start, and accelerating inertias.
At time tStep a load step is applied.
Simulate for 2 seconds and plot (versus time):

- DCPM1.ia: armature current
- DCPM1.rpm_mechanical: motor's speed
- DCPM1.tau_electrical: motor's torque

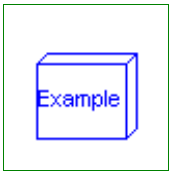
Default machine parameters of model *DC_PermanentMagnet* are used.

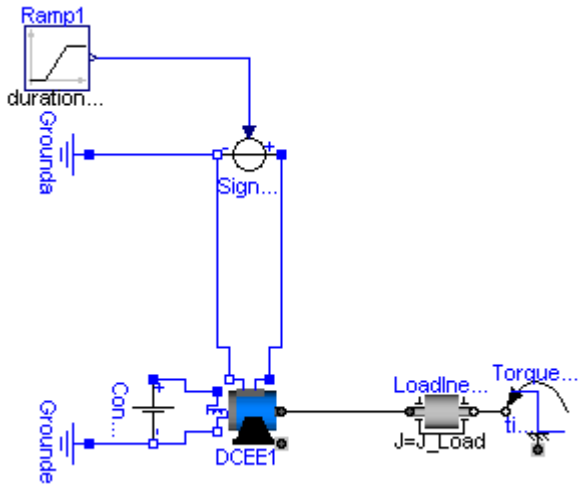
Parameters

Type	Name	Default	Description
Voltage	Va	100	actual armature voltage [V]
Time	tStart	0.2	armature voltage ramp [s]
Time	tRamp	0.8	armature voltage ramp [s]
Torque	T_Load	63.66	nominal load torque [N.m]
Time	tStep	1.5	time of load torque step [s]
Inertia	J_Load	0.15	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.DCEE_start

Test example 9: DC with electrical ecxitation starting with voltage ramp





Information

9th Test example: Electrically separate excited DC machine started with an armature voltage ramp
A voltage ramp is applied to the armature, causing the DC machine to start, and accelerating inertias.
At time tStep a load step is applied.
Simulate for 2 seconds and plot (versus time):

- DCEE1.ia: armature current
- DCEE1.rpm_mechanical: motor's speed
- DCEE1.tau_electrical: motor's torque
- DCEE1.ie: excitation current

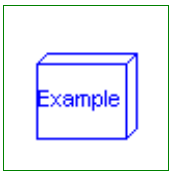
Default machine parameters of model *DC_ElectricalExcited* are used.

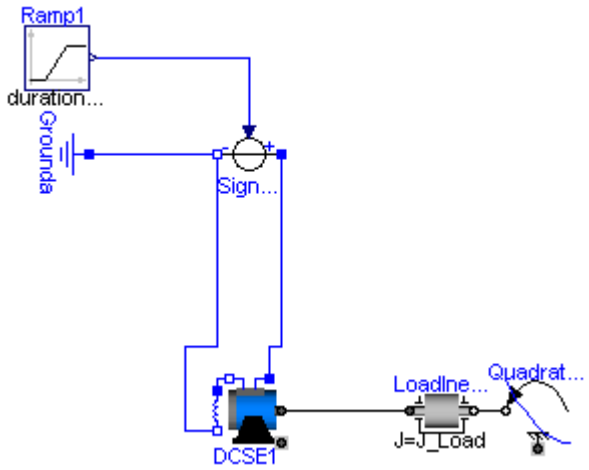
Parameters

Type	Name	Default	Description
Voltage	Va	100	actual armature voltage [V]
Time	tStart	0.2	armature voltage ramp [s]
Time	tRamp	0.8	armature voltage ramp [s]
Voltage	Ve	100	actual excitation voltage [V]
Torque	T_Load	63.66	nominal load torque [N.m]
Time	tStep	1.5	time of load torque step [s]
Inertia	J_Load	0.15	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.DCSE_start

Test example 10: DC with serial excitation starting with voltage ramp





Information

10th Test example: Series excited DC machine started with an armature voltage ramp
A voltage ramp is applied to the armature, causing the DC machine to start, and accelerating inertiasagainst load torque quadratic dependent on speed, finally reaching nominal speed.
Simulate for 2 seconds and plot (versus time):

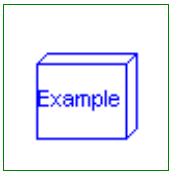
- DCSE1.ia: armature current
- DCSE1.rpm_mechanical: motor's speed
- DCSE1.tau_electrical: motor's torque

Default machine parameters of model *DC_SeriesExcited* are used.

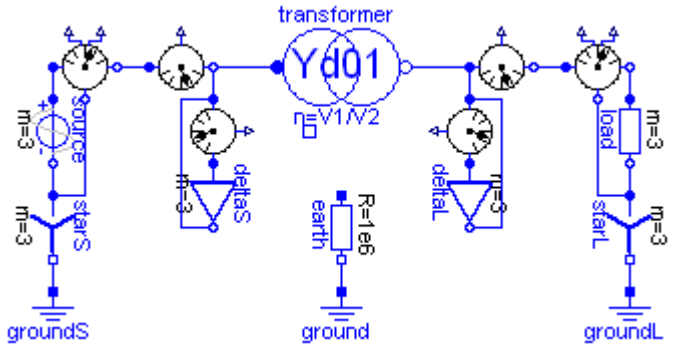
Parameters

Type	Name	Default	Description
Voltage	Va	100	actual armature voltage [V]
Time	tStart	0.2	armature voltage ramp [s]
Time	tRamp	0.8	armature voltage ramp [s]
Torque	T_Load	63.66	nominal load torque [N.m]
AngularVelocity_rpm	rpmLoad	1410	nominal load speed [rev/min]
Inertia	J_Load	0.15	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.TransformerTestbench



Transformer Testbench



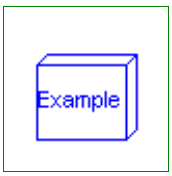
Information

Transformer testbench:
You may choose different connections as well as vary the load (even not symmetrical).
Please pay attention to proper grounding of the primary and secondary part of the whole circuit.
The primary and secondary starpoint are available as connectors, if the connection is not delta (D or d).
In some cases it may be necessary to ground the transformer's starpoint even though the source's or load's starpoint are grounded; you may use a reasonable high earthing resistance.

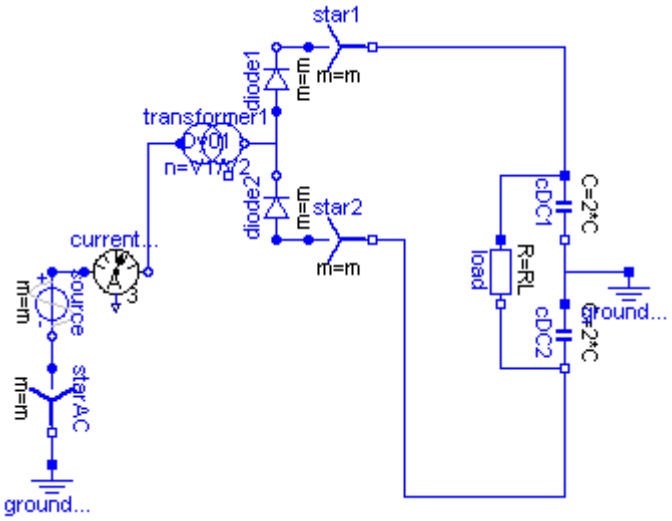
Parameters

Type	Name	Default	Description
Resistance	RL[3]	fill(1/3, 3)	Load resistance [Ohm]

Modelica.Electrical.Machines.Examples.Rectifier6pulse



6-pulse rectifier with 1 transformer



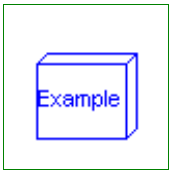
Information

Test example with multiphase components:
Star-connected voltage source feeds via a transformer a diode bridge rectifier with a DC burden.
Using f=50 Hz, simulate for 0.1 seconds (5 periods) and compare voltages and currents of source and DC burden, neglecting initial transient.

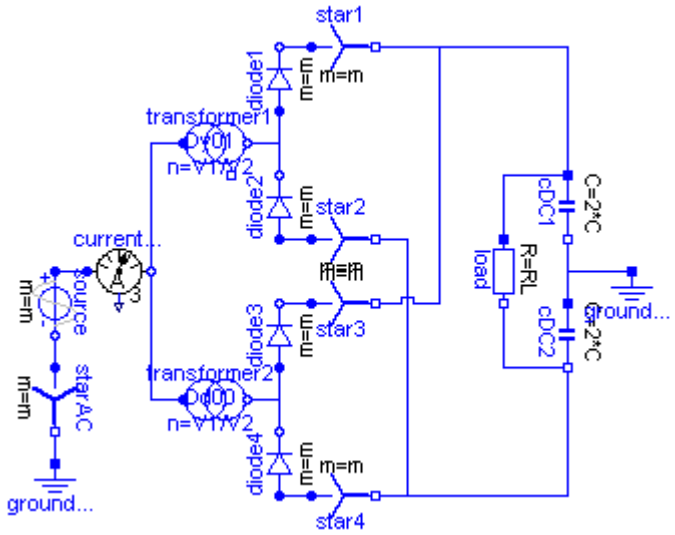
Parameters

Type	Name	Default	Description
Voltage	V	100*sqrt(2/3)	Amplitude of star-voltage [V]
Frequency	f	50	Frequency [Hz]
Resistance	RL	0.4	Load resistance [Ohm]
Capacitance	C	0.005	Total DC-capacitance [F]
Voltage	VC0	sqrt(3)*V	Initial voltage of capacitance [V]

Modelica.Electrical.Machines.Examples.Rectifier12pulse



12-pulse rectifier with 2 transformers



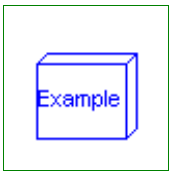
Information

Test example with multiphase components:
Star-connected voltage source feeds via two transformers (Dd0 and Dy1) two diode bridge rectifiers with a single DC burden.
Using f=50 Hz, simulate for 0.1 seconds (5 periods) and compare voltages and currents of source and DC burden, neglecting initial transient.

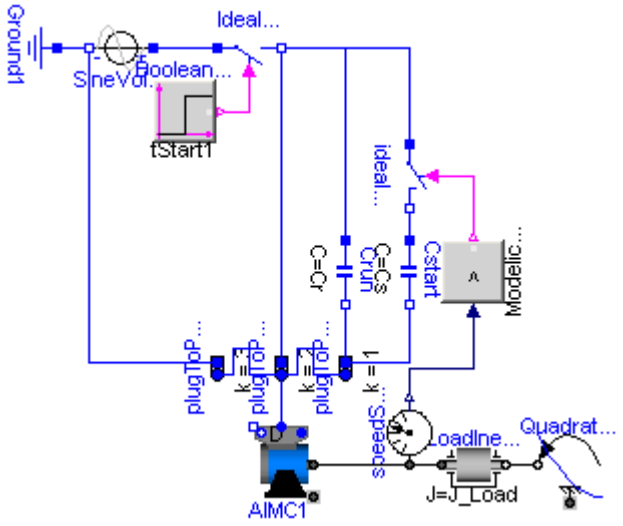
Parameters

Type	Name	Default	Description
Voltage	V	100*sqrt(2/3)	Amplitude of star-voltage [V]
Frequency	f	50	Frequency [Hz]
Resistance	RL	0.2	Load resistance [Ohm]
Capacitance	C	0.005	Total DC-capacitance [F]
Voltage	VC0	sqrt(3)*V	Initial voltage of capacitance [V]

Modelica.Electrical.Machines.Examples.AIMC_Steinmetz



AsynchronousInductionMachineSquirrelCage Steinmetz-connection



Information

Asynchronous induction machine with squirrel cage - Steinmetz-connection
At start time tStart single phase voltage is supplied to the asynchronous induction machine with squirrel cage; the machine starts from standstill, accelerating inertias against load torque quadratic dependent on speed, finally reaching nominal speed.
Default machine parameters of model *AIM_SquirrelCage* are used.

Parameters

Type	Name	Default	Description
Voltage	VNominal	100	nominal RMS voltage per phase [V]
Frequency	fNominal	50	nominal frequency [Hz]
Time	tStart1	0.1	start time [s]
Capacitance	Cr	0.0035	motor's running capacitor [F]
Capacitance	Cs	5*Cr	motor's (additional) starting capacitor [F]
AngularVelocity_rpm	rpmSwitch	1350	speed for switching off the starting capacitor [rev/min]
Torque	T_Load	2/3*161.4	nominal load torque [N.m]
AngularVelocity_rpm	rpmLoad	1462.5	nominal load speed [rev/min]
Inertia	J_Load	0.29	load's moment of inertia [kg.m2]

Modelica.Electrical.Machines.Examples.Utilities

Library with auxiliary models for testing

Information

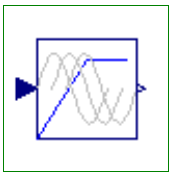
This package contains components utility components for testing examples.

Package Content

Name	Description
VfController	Voltage-Frequency-Controller

SwitchYD	Y-D-switch
TerminalBox	terminal box Y/D-connection

Modelica.Electrical.Machines.Examples.Utilities.VfController



Voltage-Frequency-Controller

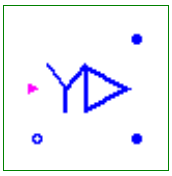
Information

Simple Voltage-Frequency-Controller.
Amplitude of voltage is linear dependent (VNominal/fNominal) on frequency (input signal "u"), but limited by VNominal (nominal RMS voltage per phase).
m sine-waves with amplitudes as described above are provided as output signal "y".
The sine-waves are intended to feed a m-phase SignalVoltage.
Phase shifts between sine-waves may be choosen by the user; default values are *(k-1)/m*pi for k in 1:m*.

Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Voltage	VNominal		nominal RMS voltage per phase [V]
Frequency	fNominal		nominal frequency [Hz]
Angle	BasePhase	0	common phase shift [rad]

Modelica.Electrical.Machines.Examples.Utilities.SwitchYD



Y-D-switch

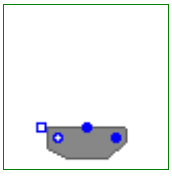
Information

Simple Star-Delta-switch.
If *control* is false, plug_PS and plug_NS are star connected and plug_PS connected to plug_P.
If *control* is true, plug_PS and plug_NS are delta connected and they are connected to plug_P.

Connectors

Type	Name	Description
PositivePlug	plug_P	
PositivePlug	pug_PS	
NegativePlug	plug_NS	
input BooleanInput	control[m]	

Modelica.Electrical.Machines.Examples.Utilities.TerminalBox



terminal box Y/D-connection

Information

TerminalBox: at the bottom connected to both machine plugs, connect at the top to the grid as usual, choosing Y-connection (StarDelta=Y) or D-connection (StarDelta=D).

Parameters

Type	Name	Default	Description
String	StarDelta	"Y"	Choose Y=star/D=delta

Connectors

Type	Name	Description
PositivePlug	positiveMachinePlug	
NegativePlug	negativeMachinePlug	
PositivePlug	plugToGrid	
NegativePin	starpoint	

Modelica.Electrical.Machines.BasicMachines

Basic machine models

Information

This package contains components for modeling electrical machines, specially threephase induction machines, based on space phasor theory:

- package AsynchronousInductionMachines: models of three phase asynchronous induction machines
- package SynchronousInductionMachines: models of three phase synchronous induction machines
- package DCMachines: models of DC machines with different excitation
- package Transformers: Threephase transformers (see detailled documentation in subpackage)
- package Components: components for modeling machines and transformers

The induction machine models use package SpacePhasors.

Package Content

Name	Description
AsynchronousInductionMachines	Models of asynchronous induction machines
SynchronousInductionMachines	Models of synchronous induction machines
DCMachines	Models of DC machines
Transformers	Library for technical 3phase transformers
Components	Machine components like AirGaps

Modelica.Electrical.Machines.BasicMachines.AsynchronousInductionMachines

Models of asynchronous induction machines

Information

This package contains models of asynchronous induction machines, based on space phasor theory:

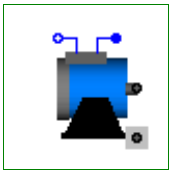
- AIM_SquirrelCage: asynchronous induction machine with squirrel cage
- AIM_SlipRing: asynchronous induction machine with wound rotor

These models use package SpacePhasors.

Package Content

Name	Description
AIM_SquirrelCage	Asynchronous induction machine with squirrel cage rotor
AIM_SlipRing	Asynchronous induction machine with slipring rotor

Modelica.Electrical.Machines.BasicMachines.AsynchronousInductionMachines.AIM_SquirrelCage



Asynchronous induction machine with squirrel cage rotor

Information

Model of a three phase asynchronous induction machine with squirrel cage.
Resistance and stray inductance of stator is modeled directly in stator phases, then using space phasor transformation. Resistance and stray inductance of rotor's squirrel cage is modeled in two axis of the rotor-fixed ccordinate system. Both together connected via a stator-fixed *AirGap* model. Only losses in stator and rotor resistance are taken into account.
Default values for machine's parameters (a realistic example) are:

number of pole pairs p	2	
stator's moment of inertia	0.29	kg.m2
rotor's moment of inertia	0.29	kg.m2
nominal frequency fNominal	50	Hz
nominal voltage per phase	100	V RMS
nominal current per phase	100	A RMS
nominal torque	161.4	Nm
nominal speed	1440.45	rpm
nominal mechanical output	24.346	kW
efficiency	92.7	%
power factor	0.875	
stator resistance	0.03	Ohm per phase in warm condition
rotor resistance	0.04	Ohm in warm condition
stator reactance Xs	3	Ohm per phase
rotor reactance Xr	3	Ohm
total stray coefficient sigma	0.0667	
These values give the following inductances, assuming equal stator and rotor stray inductances:		
stator stray inductance per phase	Xs * (1 - sqrt(1-sigma))/(2*pi*fNominal)	
rotor stray inductance	Xr * (1 - sqrt(1-sigma))/(2*pi*fNominal)	
main field inductance per phase	sqrt(Xs*Xr * (1-sigma))/(2*pi*fNominal)	

Parameters

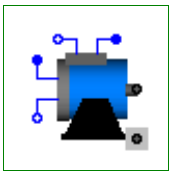
Type	Name	Default	Description

Inertia	J_Rotor	0.29	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Integer	p	2	number of pole pairs (Integer)
Frequency	fNominal	50	nominal frequency [Hz]
Nominal resistances and inductances			
Resistance	Rs	0.03	warm stator resistance per phase [Ohm]
Inductance	Lssigma	3*(1 - sqrt(1 - 0.0667))/(2*...	stator stray inductance per phase [H]
Inductance	Lm	3*sqrt(1 - 0.0667)/(2*pi*fNo...	main field inductance [H]
Inductance	Lrsigma	3*(1 - sqrt(1 - 0.0667))/(2*...	rotor stray inductance (equivalent three phase winding) [H]
Resistance	Rr	0.04	warm rotor resistance (equivalent three phase winding) [Ohm]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePlug	plug_sp	
NegativePlug	plug_sn	

Modelica.Electrical.Machines.BasicMachines.AsynchronousInductionMachines.AIM_SlipRing



Asynchronous induction machine with slipring rotor

Information

Model of a three phase asynchronous induction machine with slipring rotor.
Resistance and stray inductance of stator and rotor are modeled directly in stator respectively rotor phases, then using space phasor transformation and a stator-fixed *AirGap* model. Only losses in stator and rotor resistance are taken into account.
Default values for machine's parameters (a realistic example) are:

number of pole pairs p	2	
stator's moment of inertia	0.29	kg.m2
rotor's moment of inertia	0.29	kg.m2
nominal frequency fNominal	50	Hz
nominal voltage per phase	100	V RMS
nominal current per phase	100	A RMS
nominal torque	161.4	Nm
nominal speed	1440.45	rpm
nominal mechanical output	24.346	kW
efficiency	92.7	%
power factor	0.875	
stator resistance	0.03	Ohm per phase in warm condition
rotor resistance	0.04	Ohm per phase in warm condition
stator reactance Xs	3	Ohm per phase

rotor reactance Xr	3	Ohm per phase
total stray coefficient sigma	0.0667	
TurnsRatio	1	effective ratio of stator and rotor current (ws*xis) / (wr*xir)
These values give the following inductances:		
stator stray inductance per phase	Xs * (1 - sqrt(1-sigma))/(2*pi*fNominal)	
rotor stray inductance	Xr * (1 - sqrt(1-sigma))/(2*pi*fNominal)	
main field inductance per phase	sqrt(Xs*Xr * (1-sigma))/(2*pi*f)	

Parameter TurnsRatio could be obtained from the following relationship at standstill with open rotor circuit at nominal voltage and nominal frequency,
using the locked-rotor voltage VR, no-load stator current I0 and powerfactor PF0:
TurnsRatio * $\underline{V}_R = \underline{V}_s - (R_s + j\, X_{s,sigma})\, \underline{I}_0$

Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.29	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Integer	p	2	number of pole pairs (Integer)
Frequency	fNominal	50	nominal frequency [Hz]
Boolean	useTurnsRatio	true	use TurnsRatio or calculate from locked-rotor voltage?
Real	TurnsRatio	1	(ws*xis) / (wr*xir)
Voltage	VsNom	100	Nominal stator voltage per phase [V]
Voltage	Vr_LR	100*(2*pi*fNominal*Lm)/sqrt(...	Locked-rotor voltage per phase [V]
Nominal resistances and inductances			
Resistance	Rs	0.03	warm stator resistance per phase [Ohm]
Inductance	Lssigma	3*(1 - sqrt(1 - 0.0667))/(2*...	stator stray inductance per phase [H]
Inductance	Lm	3*sqrt(1 - 0.0667)/(2*pi*fNo...	main field inductance [H]
Inductance	Lrsigma	3*(1 - sqrt(1 - 0.0667))/(2*...	rotor stray inductance per phase [H]
Resistance	Rr	0.04	warm rotor resistance per phase [Ohm]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePlug	plug_sp	
NegativePlug	plug_sn	
PositivePlug	plug_rp	
NegativePlug	plug_rn	

Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines

Models of synchronous induction machines

Information

This package contains models of synchronous induction machines, based on space phasor theory:

- SM_PermanentMagnetDamperCage: synchronous induction machine with permanent magnet excitation, with damper cage
- SM_ElectricalExcitedDamperCage: synchronous induction machine with electrical excitation and damper cage
- SM_ReluctanceRotorDamperCage: induction machine with reluctance rotor and damper cage i.e. a squirrel cage rotor with magnetic poles due to different airgap width

These models use package SpacePhasors.
Please keep in mind:

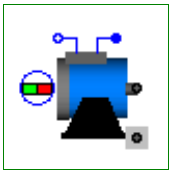
- We keep the same reference system as for motors, i.e.:
  Positive RotorDisplacementAngle means acting as motor,
  with positive electric power consumption and positive mechanical power output.
- ElectricalAngle = p * MechanicalAngle
- real axis = d-axis
  imaginary= q-axis
- Voltage induced by the magnet wheel (d-axis) is located in the q-axis.

Package Content

Name	Description
SM_PermanentMagnetDamperCage	Permanent magnet synchronous induction machine
SM_ElectricalExcitedDamperCage	Electrical excited synchronous induction machine with damper cage
SM_ReluctanceRotorDamperCage	Synchronous induction machine with reluctance rotor and damper cage

---

Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines.SM_PermanentMagnetDamperCage



Permanent magnet synchronous induction machine

Information

Model of a three phase permanent magnet synchronous induction machine.
Resistance and stray inductance of stator is modeled directly in stator phases, then using space phasor transformation and a rotor-fixed *AirGap* model. Resistance and stray inductance of rotor's squirrel cage is modeled in two axis of the rotor-fixed ccordinate system. Permanent magnet excitation is modelled by a constant equivalent excitation current feeding the d-axis. Only losses in stator and damper resistance are taken into account.
Whether a damper cage is present or not, can be selected with Boolean parameter DamperCage (default = true).
Default values for machine's parameters (a realistic example) are:

number of pole pairs p	2	
stator's moment of inertia	0.29	kg.m2
rotor's moment of inertia	0.29	kg.m2
nominal frequency fNominal	50	Hz

nominal voltage per phase	100	V RMS
no-load voltage per phase	112.3	V RMS @ nominal speed
nominal current per phase	100	A RMS
nominal torque	181.4	Nm
nominal speed	1500	rpm
nominal mechanical output	28.5	kW
nominal rotor angle	20.75	degree
efficiency	95.0	%
power factor	0.98	
stator resistance	0.03	Ohm per phase in warm condition
stator reactance Xd	0.4	Ohm per phase in d-axis
stator reactance Xq	0.4	Ohm per phase in q-axis
stator stray reactance Xss	0.1	Ohm per phase
damper resistance in d-axis	0.04	Ohm in warm condition
damper resistance in q-axis	same as d-axis	
damper stray reactance in d-axis XDds	0.05	Ohm
damper stray reactance in q-axis XDqs	same as d-axis	
These values give the following inductances:		
main field inductance in d-axis	(Xd - Xss)/(2*pi*fNominal)	
main field inductance in q-axis	(Xq - Xss)/(2*pi*fNominal)	
stator stray inductance per phase	Xss/(2*pi*fNominal)	
damper stray inductance in d-axis	XDds/(2*pi*fNominal)	
damper stray inductance in q-axis	XDqs/(2*pi*fNominal)	

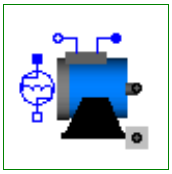
Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.29	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Integer	p	2	number of pole pairs (Integer)
Excitation			
Frequency	fNominal	50	nominal frequency [Hz]
Voltage	V0	112.3	no-load RMS voltage per phase @ fNominal [V]
Nominal resistances and inductances			
Resistance	Rs	0.03	warm stator resistance per phase [Ohm]
Inductance	Lssigma	0.1/(2*pi*fNominal)	stator stray inductance per phase [H]
Inductance	Lmd	0.3/(2*pi*fNominal)	main field inductance in d-axis [H]
Inductance	Lmq	0.3/(2*pi*fNominal)	main field inductance in q-axis [H]
DamperCage			
Boolean	DamperCage	true	damper cage is present?
Inductance	Lrsigma	0.05/(2*pi*fNominal)	damper stray inductance in d-axis [H]
Inductance	Lrsigmaq	Lrsigma	damper stray inductance in q-axis [H]
Resistance	Rr	0.04	warm damper resistance in d-axis [Ohm]
Resistance	Rrq	Rr	warm damper resistance in q-axis [Ohm]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePlug	plug_sp	
NegativePlug	plug_sn	

Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines.SM_ElectricalExcitedDamperCage



Electrical excited synchronous induction machine with damper cage

Information

Model of a three phase electrical excited synchronous induction machine with damper cage.
Resistance and stray inductance of stator is modeled directly in stator phases, then using space phasor transformation and a rotor-fixed *AirGap* model. Resistance and stray inductance of rotor's squirrel cage is modeled in two axis of the rotor-fixed ccordinate system. Electrical excitation is modelled by converting excitation current and voltage to d-axis space phasors. Only losses in stator, damper and excitation resistance are taken into account.
Whether a damper cage is present or not, can be selected with Boolean parameter DamperCage (default = true).

Default values for machine's parameters (a realistic example) are:

number of pole pairs p	2	
stator's moment of inertia	0.29	kg.m2
rotor's moment of inertia	0.29	kg.m2
nominal frequency fNominal	50	Hz
nominal voltage per phase	100	V RMS
no-load excitation current @ nominal voltage and frequency	10	A DC
warm excitation resistance	2.5	Ohm
nominal current per phase	100	A RMS
nominal apparent power	-30000	VA
power factor	-1.0	ind./cap.
nominal excitation current	19	A
efficiency w/o excitation	97.1	%
nominal torque	-196.7	Nm
nominal speed	1500	rpm
nominal rotor angle	-57.23	degree
stator resistance	0.03	Ohm per phase in warm condition
stator reactance Xd	1.6	Ohm per phase in d-axis
giving Kc	0.625	
stator reactance Xq	1.6	Ohm per phase in q-axis
stator stray reactance Xss	0.1	Ohm per phase
damper resistance in d-axis	0.04	Ohm in warm condition
damper resistance in q-axis	same as d-axis	
damper stray reactance in d-axis XDds	0.1	Ohm
damper stray reactance in q-axis XDqs	same as d-axis	

excitation stray inductance	2.5	% of total excitation inductance
These values give the following inductances:		
main field inductance in d-axis	(Xd - Xss)/(2*pi*fNominal)	
main field inductance in q-axis	(Xq - Xss)/(2*pi*fNominal)	
stator stray inductance per phase	Xss/(2*pi*fNominal)	
damper stray inductance in d-axis	XDds/(2*pi*fNominal)	
damper stray inductance in q-axis	XDqs/(2*pi*fNominal)	

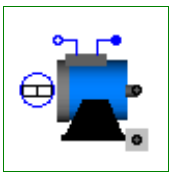
Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.29	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Integer	p	2	number of pole pairs (Integer)
Excitation			
Frequency	fNominal	50	nominal frequency [Hz]
Voltage	VNominal	100	nominal stator RMS voltage per phase [V]
Current	Ie0	10	no-load excitation current @ nominal voltage and frequency [A]
Resistance	Re	2.5	warm excitation resistance [Ohm]
Real	sigmae	0.025	stray fraction of total excitation inductance
Nominal resistances and inductances			
Resistance	Rs	0.03	warm stator resistance per phase [Ohm]
Inductance	Lssigma	0.1/(2*pi*fNominal)	stator stray inductance per phase [H]
Inductance	Lmd	1.5/(2*pi*fNominal)	main field inductance in d-axis [H]
Inductance	Lmq	1.5/(2*pi*fNominal)	main field inductance in q-axis [H]
DamperCage			
Boolean	DamperCage	true	damper cage is present?
Inductance	Lrsigma	0.05/(2*pi*fNominal)	damper stray inductance in d-axis [H]
Inductance	Lrsigmaq	Lrsigma	damper stray inductance in q-axis [H]
Resistance	Rr	0.04	warm damper resistance in d-axis [Ohm]
Resistance	Rrq	Rr	warm damper resistance in q-axis [Ohm]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePlug	plug_sp	
NegativePlug	plug_sn	
PositivePin	pin_ep	
NegativePin	pin_en	

Modelica.Electrical.Machines.BasicMachines.SynchronousInductionMachines.SM_ReluctanceRotorDamperCage



Synchronous induction machine with reluctance rotor and damper cage

Information

Model of a three phase synchronous induction machine with reluctance rotor and damper cage.
Resistance and stray inductance of stator is modeled directly in stator phases, then using space phasor transformation. Resistance and stray inductance of rotor's squirrel cage is modeled in two axis of the rotor-fixed ccordinate system. Both together connected via a rotor-fixed *AirGap* model. Only losses in stator and rotor resistance are taken into account.
Whether a damper cage is present or not, can be selected with Boolean parameter DamperCage (default = true).

Default values for machine's parameters (a realistic example) are:

number of pole pairs p	2	
stator's moment of inertia	0.29	kg.m2
rotor's moment of inertia	0.29	kg.m2
nominal frequency fNominal	50	Hz
nominal voltage per phase	100	V RMS
nominal current per phase	50	A RMS
nominal torque	46	Nm
nominal speed	1500	rpm
nominal mechanical output	7.23	kW
efficiency	96.98	%
power factor	0.497	
stator resistance	0.03	Ohm per phase in warm condition
rotor resistance in d-axis	0.04	Ohm in warm condition
rotor resistance in q-axis	same as d-axis	
stator reactance Xsd in d-axis	3	Ohm per phase
stator reactance Xsq in q-axis	1	Ohm
stator stray reactance Xss	0.1	Ohm per phase
rotor stray reactance in d-axis Xrds	0.1	Ohm per phase
rotor stray reactance in q-axis Xrqs	same as d-axis	
These values give the following inductances:		
stator stray inductance per phase	Xss/(2*pi*fNominal)	
rotor stray inductance in d-axis	Xrds/(2*pi*fNominal)	
rotor stray inductance in q-axis	Xrqs/(2*pi*fNominal)	
main field inductance per phase in d-axis	(Xsd-Xss)/(2*pi*fNominal)	
main field inductance per phase in q-axis	(Xsq-Xss)/(2*pi*fNominal)	

Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.29	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Integer	p	2	number of pole pairs (Integer)
Excitation			
Frequency	fNominal	50	nominal frequency [Hz]

Nominal resistances and inductances			
Resistance	Rs	0.03	warm stator resistance per phase [Ohm]
Inductance	Lssigma	0.1/(2*pi*fNominal)	stator stray inductance per phase [H]
Inductance	Lmd	2.9/(2*pi*fNominal)	main field inductance in d-axis [H]
Inductance	Lmq	0.9/(2*pi*fNominal)	main field inductance in q-axis [H]
DamperCage			
Boolean	DamperCage	true	damper cage is present?
Inductance	Lrsigma	0.05/(2*pi*fNominal)	damper stray inductance in d-axis [H]
Inductance	Lrsigmaq	Lrsigma	damper stray inductance in q-axis [H]
Resistance	Rr	0.04	warm damper resistance in d-axis [Ohm]
Resistance	Rrq	Rr	warm damper resistance in q-axis [Ohm]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePlug	plug_sp	
NegativePlug	plug_sn	

Modelica.Electrical.Machines.BasicMachines.DCMachines

Models of DC machines

Information

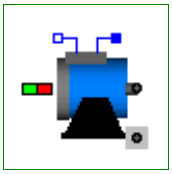
This package contains models of DC machines:

- DC_PermanentMagnet: DC machine with permanent magnet excitation
- DC_ElectricalExcited: DC machine with electrical shunt or separate excitation
- DC_SeriesExcited: DC machine with series excitation

Package Content

Name	Description
DC_PermanentMagnet	Permanent magnet DC machine
DC_ElectricalExcited	Electrical shunt/separate excited linear DC machine
DC_SeriesExcited	Series excited linear DC machine

Modelica.Electrical.Machines.BasicMachines.DCMachines.DC_PermanentMagnet



Permanent magnet DC machine

Information

Model of a DC Machine with Permanent magnet.
Armature resistance and inductance are modeled directly after the armature pins, then using a *AirGapDC* model. Permanent magnet excitation is modelled by a constant equivalent excitation current feeding

AirGapDC. Only losses in armature resistance are taken into account. No saturation is modelled.
Default values for machine's parameters (a realistic example) are:

stator's moment of inertia	0.29	kg.m2
rotor's moment of inertia	0.15	kg.m2
nominal armature voltage	100	V
nominal armature current	100	A
nominal speed	1425	rpm
nominal torque	63.66	Nm
nominal mechanical output	9.5	kW
efficiency	95.0	%
armature resistance	0.05	Ohm in warm condition
armature inductance	0.0015	H

Armature resistance resp. inductance include resistance resp. inductance of commutating pole winding and compensation windig, if present.

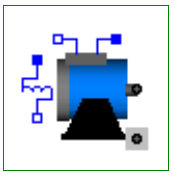
Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.15	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Nominal parameters			
Voltage	VaNominal	100	nominal armature voltage [V]
Current	IaNominal	100	nominal armature current [A]
AngularVelocity_rpm	rpmNominal	1425	nominal speed [rev/min]
Nominal resistances and inductances			
Resistance	Ra	0.05	warm armature resistance [Ohm]
Inductance	La	0.0015	armature inductance [H]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePin	pin_ap	
NegativePin	pin_an	

Modelica.Electrical.Machines.BasicMachines.DCMachines.DC_ElectricalExcited



Electrical shunt/separate excited linear DC machine

Information

Model of a DC Machine with Electrical shunt or separate excitation.
Armature resistance and inductance are modeled directly after the armature pins, then using a *AirGapDC* model.
Only losses in armature and excitation resistance are taken into account. No saturation is modelled.
Shunt or separate excitation is defined by the user's external circuit.
Default values for machine's parameters (a realistic example) are:

stator's moment of inertia	0.29	kg.m2
rotor's moment of inertia	0.15	kg.m2

nominal armature voltage	100	V
nominal armature current	100	A
nominal torque	63.66	Nm
nominal speed	1425	rpm
nominal mechanical output	9.5	kW
efficiency	95.0	% only armature
efficiency	94.06	% including excitation
armature resistance	0.05	Ohm in warm condition
aramture inductance	0.0015	H
nominal excitation voltage	100	V
nominal excitation current	1	A
excitation resistance	100	Ohm in warm condition
excitation inductance	1	H

Armature resistance resp. inductance include resistance resp. inductance of commutating pole winding and compensation windig, if present.
Armature current does not cover excitation current of a shunt excitation; in this case total current drawn from the grid = armature current + excitation current.

Parameters

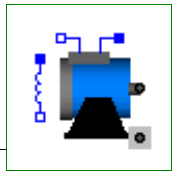
Type	Name	Default	Description
Inertia	J_Rotor	0.15	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Nominal parameters			
Voltage	VaNominal	100	nominal armature voltage [V]
Current	IaNominal	100	nominal armature current [A]
AngularVelocity_rpm	rpmNominal	1425	nominal speed [rev/min]
Nominal resistances and inductances			
Resistance	Ra	0.05	warm armature resistance [Ohm]
Inductance	La	0.0015	armature inductance [H]
Excitation			
Current	IeNominal	1	nominal excitation current [A]
Resistance	Re	100	warm field excitation resistance [Ohm]
Inductance	Le	1	total field excitation inductance [H]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePin	pin_ap	
NegativePin	pin_an	
PositivePin	pin_ep	
NegativePin	pin_en	

Modelica.Electrical.Machines.BasicMachines.DCMachines.DC_SeriesExcited

Series excited linear DC machine



Information

Model of a DC Machine with Series excitation.
Armature resistance and inductance are modeled directly after the armature pins, then using a *AirGapDC* model.
Only losses in armature and excitation resistance are taken into account. No saturation is modelled.
Series excitation has to be connected by the user's external circuit.
Default values for machine's parameters (a realistic example) are:

stator's moment of inertia	0.29	kg.m2
rotor's moment of inertia	0.15	kg.m2
nominal armature voltage	100	V
nominal armature current	100	A
nominal torque	63.66	Nm
nominal speed	1410	rpm
nominal mechanical output	9.4	kW
efficiency	94.0	% only armature
armature resistance	0.05	Ohm in warm condition
aramture inductance	0.0015	H
excitation resistance	0.01	Ohm in warm condition
excitation inductance	0.0005	H

Armature resistance resp. inductance include resistance resp. inductance of commutating pole winding and compensation windig, if present.
Parameter nominal armature voltage includes voltage drop of series excitation;
but for output the voltage is splitted into:
va = armature voltage without voltage drop of series excitation
ve = voltage drop of series excitation

Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.15	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Nominal parameters			
Voltage	VaNominal	100	nominal armature voltage [V]
Current	IaNominal	100	nominal armature current [A]
AngularVelocity_rpm	rpmNominal	1410	nominal speed [rev/min]
Nominal resistances and inductances			
Resistance	Ra	0.05	warm armature resistance [Ohm]
Inductance	La	0.0015	armature inductance [H]
Excitation			
Resistance	Re	0.01	warm field excitation resistance [Ohm]
Inductance	Le	0.0005	total field excitation inductance [H]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePin	pin_ap	
NegativePin	pin_an	

PositivePin	pin_ep	
NegativePin	pin_en	

Modelica.Electrical.Machines.BasicMachines.Transformers

Library for technical 3phase transformers

Information

This package contains components to model technical threephase transformers:

- Transformer: transformer model to choose connection / vector group
- Yy: Transformers with primary primary Y / secondary y
- Yd: Transformers with primary primary Y / secondary d
- Yz: Transformers with primary primary Y / secondary zig-zag
- Dy: Transformers with primary primary D / secondary y
- Dd: Transformers with primary D / secondary d
- Dz: Transformers with primary D / secondary zig-zag

Transformers are modeled by an ideal transformer, adding primary and secondary winding resistances and stray inductances.
All transformers extend from the base model *PartialTransformer*, adding the primary and secondary connection.
VectorGroup defines the phase shift between primary and secondary voltages, expressed by a number phase shift/30 degree (i.e. the hour on a clock face). Therefore each transformer is identified by two characters and a two-digit number, e.g. Yd11 ... primary connection Y (star), secondfary connection d (delta), vector group 11 (phase shift 330 degree)
With the "supermodel" *Tranformer* the user may choose primary and secondary connection as well as the vector group.
It calculates winding ratio as well as primary and secondary winding resistances and stray inductances, distributing them equally to primary and secondary winding, from the following parameters:

- nominal frequency
- primary voltage (RMS line-to-line)
- secondary voltage (RMS line-to-line)
- nominal apparent power
- impedance voltage drop
- short-circuit copper losses

The **impedance voltage drop** indicates the (absolute value of the) voltage drop at nominal load (current) as well as the voltage we have to apply to the primary winding to achieve nominal current in the short-circuited secondary winding.

Please pay attention to proper grounding of the primary and secondary part of the whole circuit.
The primary and secondary starpoint are available as connectors, if the connection is not delta (D or d).
In some cases (Yy or Yz) it may be necessary to ground one of the transformer's starpoints even though the source's and/or load's starpoint are grounded; you may use a reasonable high earthing resistance.

Limitations and assumptions:

- number of phases is limited to 3, therefore definition as a constant m=3
- symmetry of the 3 phases resp. limbs
- temperature dependency is neglected, i.e. resistances are constant
- saturation is neglected, i.e. inductances are constant
- magnetizing current is neglected
- magnetizing losses are neglected
- additional (stray) losses are neglected

Further development:

- modeling magnetizing current, including saturation
- temperature dependency of winding resistances

Main Authors:
Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern
Austria
email: a.haumer@haumer.at

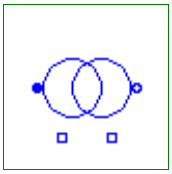
Copyright © 1998-2007, Modelica Association and Anton Haumer.

*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Transformer	Transformer: choose connection/vector group
Yy	Transformers: primary Y / secondary y
Yd	Transformers: primary Y / secondary d
Yz	Transformers: primary Y / secondary zig-zag
Dy	Transformers: primary D / secondary y
Dd	Transformers: primary D / secondary d
Dz	Transformers: primary D / secondary ziag-zag

Modelica.Electrical.Machines.BasicMachines.Transformers.Transformer



Transformer: choose connection/vector group

Information

"Supertransformer": lets the user choose connection/vector group

Parameters

Type	Name	Default	Description
Frequency	f	50	Nominal frequency [Hz]
Voltage	V1	100	Primary nominal line-to-line voltage (RMS) [V]
Voltage	V2	100	Secondary open circuit line-to-line voltage (RMS) @ primary nominal voltage [V]
ApparentPower	SNominal	30E3	Nominal apparent power [VA]
Real	v_sc	0.05	Impedance voltage drop pu
Power	P_sc	300	Short-circuit (copper) losses [W]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description

PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yy

Transformers: primary Y / secondary y

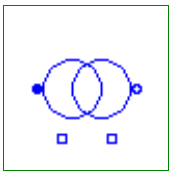
Information

This package contains transformers primary Y connected / secondary y connected in all possbile vector groups.

Package Content

Name	Description
Yy00	Transformer Yy0
Yy02	Transformer Yy2
Yy04	Transformer Yy4
Yy06	Transformer Yy6
Yy08	Transformer Yy8
Yy10	Transformer Yy10

Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy00



Transformer Yy0

Information

Transformer Yy0

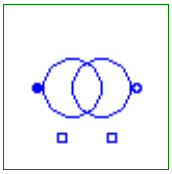
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy02



Transformer Yy2

Information

Transformer Yy2

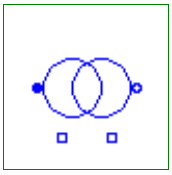
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy04



Transformer Yy4

Information

Transformer Yy4

Parameters

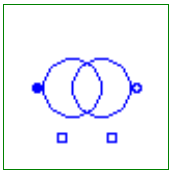
Type	Name	Default	Description

Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy06



Transformer Yy6

Information

Transformer Yy6

Parameters

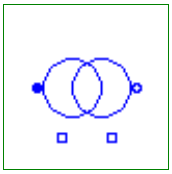
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	

NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy08



Transformer Yy8

Information

Transformer Yy8

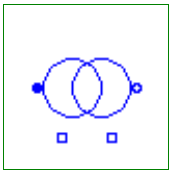
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yy.Yy10



Transformer Yy10

Information

Transformer Yy10

Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]

Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yd

Transformers: primary Y / secondary d

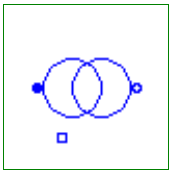
Information

This package contains transformers primary Y connected / secondary d connected in all possbile vector groups.

Package Content

Name	Description
Yd01	Transformer Yd1
Yd03	Transformer Yd3
Yd05	Transformer Yd5
Yd07	Transformer Yd7
Yd09	Transformer Yd9
Yd11	Transformer Yd11

Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd01



Transformer Yd1

Information

Transformer Yd1

Parameters

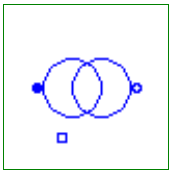
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]

Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd03



Transformer Yd3

Information

Transformer Yd3

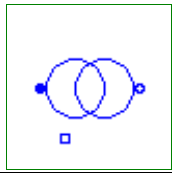
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd05



Transformer Yd5

Information

Transformer Yd5

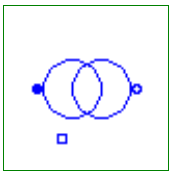
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd07



Transformer Yd7

Information

Transformer Yd7

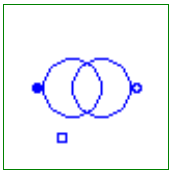
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd09



Transformer Yd9

Information

Transformer Yd9

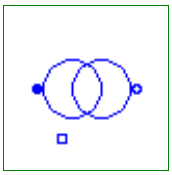
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yd.Yd11



Transformer Yd11

Information

Transformer Yd11

Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)

Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yz

Transformers: primary Y / secondary zig-zag

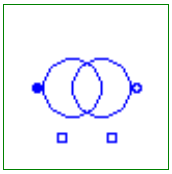
Information

This package contains transformers primary Y connected / secondary zig-zag connected in all possbile vector groups.

Package Content

Name	Description
Yz01	Transformer Yz1
Yz03	Transformer Yz3
Yz05	Transformer Yz5
Yz07	Transformer Yz7
Yz09	Transformer Yz9
Yz11	Transformer Yz11

Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz01



Transformer Yz1

Information

Transformer Yz1

Parameters

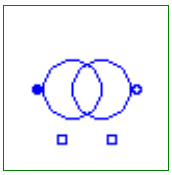
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-

			line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz03



Transformer Yz3

Information

Transformer Yz3

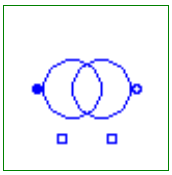
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz05



Transformer Yz5

Information

Transformer Yz5

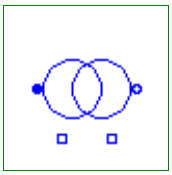
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz07



Transformer Yz7

Information

Transformer Yz7

Parameters

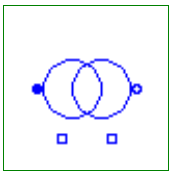
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]

Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz09



Transformer Yz9

Information

Transformer Yz9

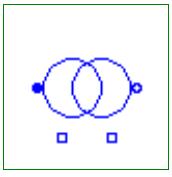
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Yz.Yz11



Transformer Yz11

Information

Transformer Yz11

Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint1	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dy

Transformers: primary D / secondary y

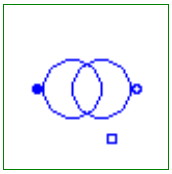
Information

This package contains transformers primary D connected / secondary y connected in all possbile vector groups.

Package Content

Name	Description
Dy01	Transformer Dy1
Dy03	Transformer Dy3
Dy05	Transformer Dy5
Dy07	Transformer Dy7
Dy09	Transformer Dy9
Dy11	Transformer Dy11

Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy01



Transformer Dy1

Information

Transformer Dy1

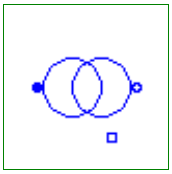
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy03



Transformer Dy3

Information

Transformer Dy3

Parameters

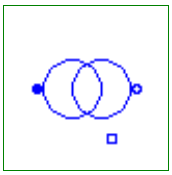
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]

IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy05



Transformer Dy5

Information

Transformer Dy5

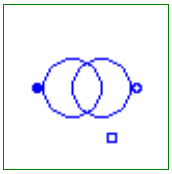
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy07



Transformer Dy7

Information

Transformer Dy7

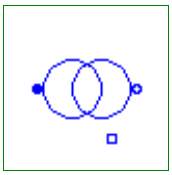
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy09



Transformer Dy9

Information

Transformer Dy9

Parameters

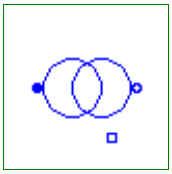
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dy.Dy11



Transformer Dy11

Information

Transformer Dy11

Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dd

Transformers: primary D / secondary d

Information

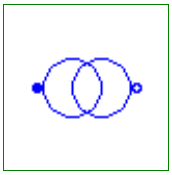
This package contains transformers primary D connected / secondary d connected in all possbile vector groups.

Package Content

Name	Description
Dd00	Transformer Dd0
Dd02	Transformer Dd2
Dd04	Transformer Dd4
Dd06	Transformer Dd6

Dd08	Transformer Dd8
Dd10	Transformer Dd10

Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd00



Transformer Dd0

Information

Transformer Dd0

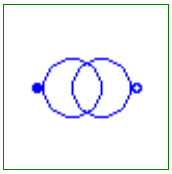
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd02



Transformer Dd2

Information

Transformer Dd2

Parameters

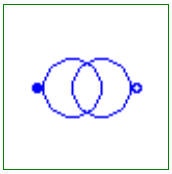
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]

Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd04



Transformer Dd4

Information

Transformer Dd4

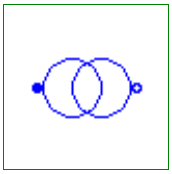
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd06



Transformer Dd6

Information

Transformer Dd6

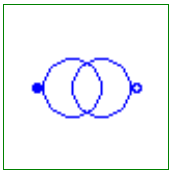
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd08



Transformer Dd8

Information

Transformer Dd8

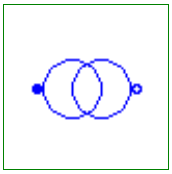
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dd.Dd10



Transformer Dd10

Information

Transformer Dd10

Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dz

Transformers: primary D / secondary ziag-zag

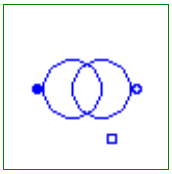
Information

This package contains transformers primary D connected / secondary d connected in all possbile vector groups.

Package Content

Name	Description
Dz00	Transformer Dz0
Dz02	Transformer Dz2
Dz04	Transformer Dz4
Dz06	Transformer Dz6
Dz08	Transformer Dz8
Dz10	Transformer Dz10

Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz00



Transformer Dz0

Information

Transformer Dz0

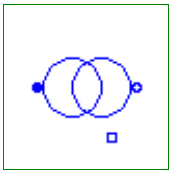
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz02



Transformer Dz2

Information

Transformer Dz2

Parameters

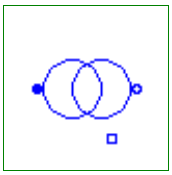
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]

IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz04



Transformer Dz4

Information

Transformer Dz4

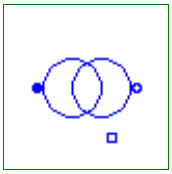
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz06



Transformer Dz6

Information

Transformer Dz6

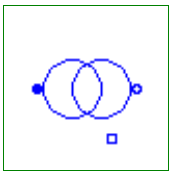
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz08



Transformer Dz8

Information

Transformer Dz8

Parameters

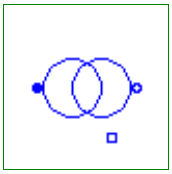
Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Transformers.Dz.Dz10



Transformer Dz10

Information

Transformer Dz10

Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	
NegativePin	starpoint2	

Modelica.Electrical.Machines.BasicMachines.Components

Machine components like AirGaps

Information

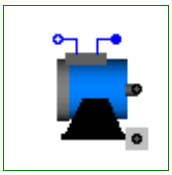
This package contains components for modeling electrical machines, specially threephase induction machines, based on space phasor theory. These models use package SpacePhasors.

Package Content

Name	Description
BasicAIM	Partial model for asynchronous induction machine
BasicSM	Partial model for synchronous induction machine
PartialAirGap	Partial airgap model
AirGapS	Airgap in stator-fixed coordinate system

AirGapR	Airgap in rotor-fixed coordinate system
SquirrelCage	Squirrel Cage
DamperCage	Squirrel Cage
ElectricalExcitation	Electrical excitation
PermanentMagnet	Permanent magnet excitation
BasicDCMachine	Partial model for DC machine
PartialAirGapDC	Partial airgap model of a DC machine
AirGapDC	Linear airgap model of a DC machine
BasicTransformer	Partial model of threephase transformer
PartialCore	Partial model of transformer core with 3 windings
IdealCore	Ideal transformer with 3 windings

Modelica.Electrical.Machines.BasicMachines.Components.BasicAIM



Partial model for asynchronous induction machine

Information

Partial model for induction machine models, containing:

- main parts of the icon
- stator plugs
- mechanical connectors

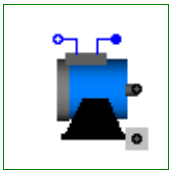
Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.29	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Integer	p	2	number of pole pairs (Integer)
Frequency	fNominal	50	nominal frequency [Hz]
AirGapS	airGapS	redeclare AirGapS airGapS(fi...	
Nominal resistances and inductances			
Resistance	Rs	0.03	warm stator resistance per phase [Ohm]
Inductance	Lssigma	3*(1 - sqrt(1 - 0.0667))/(2*...	stator stray inductance per phase [H]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePlug	plug_sp	
NegativePlug	plug_sn	

Modelica.Electrical.Machines.BasicMachines.Components.BasicSM



Partial model for synchronous induction machine

Information

Partial model for induction machine models, containing:

- main parts of the icon
- stator plugs
- mechanical connectors

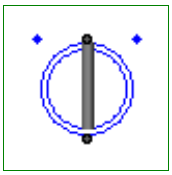
Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.29	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Integer	p	2	number of pole pairs (Integer)
AirGapR	airGapR	redeclare AirGapR airGapR(fi...	
Excitation			
Frequency	fNominal	50	nominal frequency [Hz]
Nominal resistances and inductances			
Resistance	Rs	0.03	warm stator resistance per phase [Ohm]
Inductance	Lssigma	3*(1 - sqrt(1 - 0.0667))/(2*...	stator stray inductance per phase [H]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePlug	plug_sp	
NegativePlug	plug_sn	

Modelica.Electrical.Machines.BasicMachines.Components.PartialAirGap



Partial airgap model

Information

Partial model of the airgap, using only equations.

Parameters

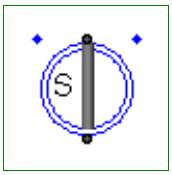
Type	Name	Default	Description
Integer	m	3	number of phases
Integer	p		number of pole pairs

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting

SpacePhasor	spacePhasor_s	
SpacePhasor	spacePhasor_r	

Modelica.Electrical.Machines.BasicMachines.Components.AirGapS



Airgap in stator-fixed coordinate system

Information

Model of the airgap in stator-fixed coordinate system, using only equations.

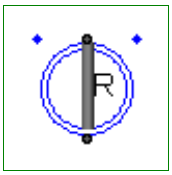
Parameters

Type	Name	Default	Description
Inductance	Lm		main field inductance [H]
Integer	m	3	number of phases
Integer	p		number of pole pairs

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
SpacePhasor	spacePhasor_s	
SpacePhasor	spacePhasor_r	

Modelica.Electrical.Machines.BasicMachines.Components.AirGapR



Airgap in rotor-fixed coordinate system

Information

Model of the airgap in rotor-fixed coordinate system, using only equations.

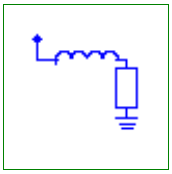
Parameters

Type	Name	Default	Description
Inductance	Lmd		main field inductance d-axis [H]
Inductance	Lmq		main field inductance q-axis [H]
Integer	m	3	number of phases
Integer	p		number of pole pairs

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
SpacePhasor	spacePhasor_s	
SpacePhasor	spacePhasor_r	

Modelica.Electrical.Machines.BasicMachines.Components.SquirrelCage



Squirrel Cage

Information

Model of a squirrel cage / damper cage in two axis.

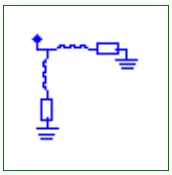
Parameters

Type	Name	Default	Description
Inductance	Lrsigma		rotor stray inductance per phase translated to stator [H]
Resistance	Rr		warm rotor resistance per phase translated to stator [Ohm]

Connectors

Type	Name	Description
SpacePhasor	spacePhasor_r	

Modelica.Electrical.Machines.BasicMachines.Components.DamperCage



Squirrel Cage

Information

Model of an unsymmetrical damper cage cage in two axis.

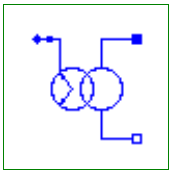
Parameters

Type	Name	Default	Description
Inductance	Lrsigma		stray inductance in d-axis per phase translated to stator [H]
Inductance	Lrsigmaq		stray inductance in q-axis per phase translated to stator [H]
Resistance	Rr		warm resistance in d-axis per phase translated to stator [Ohm]
Resistance	Rrq		warm resistance in q-axis per phase translated to stator [Ohm]

Connectors

Type	Name	Description
SpacePhasor	spacePhasor_r	

Modelica.Electrical.Machines.BasicMachines.Components.ElectricalExcitation



Electrical excitation

Information

Model of an electrical excitation, converting excitation to space phasor.

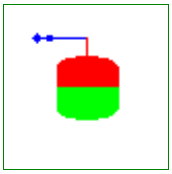
Parameters

Type	Name	Default	Description
Real	TurnsRatio	1	stator current / excitation current

Connectors

Type	Name	Description
SpacePhasor	spacePhasor_r	
PositivePin	pin_ep	
NegativePin	pin_en	

Modelica.Electrical.Machines.BasicMachines.Components.PermanentMagnet



Permanent magnet excitation

Information

Model of a permanent magnet excitation, characterized by an equivalent excitation current.

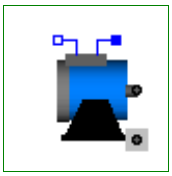
Parameters

Type	Name	Default	Description
Current	Ie		Equivalent excitation current [A]

Connectors

Type	Name	Description
SpacePhasor	spacePhasor_r	

Modelica.Electrical.Machines.BasicMachines.Components.BasicDCMachine



Partial model for DC machine

Information

Partial model for DC machine models, containing:

- main parts of the icon
- armature pins
- mechanical connectors

Parameters

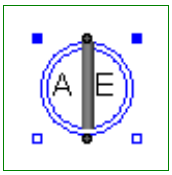
Type	Name	Default	Description
Inertia	J_Rotor	0.15	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Real	TurnsRatio		Ratio of armature turns over number of turns of the excitation winding
AirGapDC	airGapDC	redeclare AirGapDC airGapDC(...	
Nominal parameters			
Voltage	VaNominal	100	nominal armature voltage [V]
Current	IaNominal	100	nominal armature current [A]
AngularVelocity_rpm	rpmNominal	1425	nominal speed [rev/min]

Nominal resistances and inductances			
Resistance	Ra	0.05	warm armature resistance [Ohm]
Inductance	La	0.0015	armature inductance [H]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePin	pin_ap	
NegativePin	pin_an	

Modelica.Electrical.Machines.BasicMachines.Components.PartialAirGapDC



Partial airgap model of a DC machine

Information

Linear model of the airgap (without saturation effects) of a DC machine, using only equations.
Induced excitation voltage is calculated from der(flux), where flux is defined by excitation inductance times excitation current.
Induced armature voltage is calculated from flux times angular velocity.

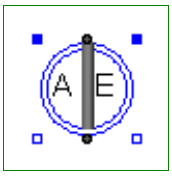
Parameters

Type	Name	Default	Description
Real	TurnsRatio		Ratio of armature turns over number of turns of the excitation winding

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePin	pin_ap	
PositivePin	pin_ep	
NegativePin	pin_an	
NegativePin	pin_en	

Modelica.Electrical.Machines.BasicMachines.Components.AirGapDC



Linear airgap model of a DC machine

Information

Linear model of the airgap (without saturation effects) of a DC machine, using only equations.
Induced excitation voltage is calculated from der(flux), where flux is defined by excitation inductance times excitation current.
Induced armature voltage is calculated from flux times angular velocity.

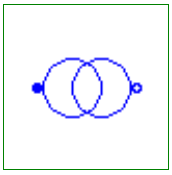
Parameters

Type	Name	Default	Description
Real	TurnsRatio		Ratio of armature turns over number of turns of the excitation winding
Inductance	Le		Excitation inductance [H]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePin	pin_ap	
PositivePin	pin_ep	
NegativePin	pin_an	
NegativePin	pin_en	

Modelica.Electrical.Machines.BasicMachines.Components.BasicTransformer



Partial model of threephase transformer

Information

Partialmodel of a threephase transformer, containing primary and secondary resistances and stray inductances, as well as the iron core. Circuit layout (vector group) of primary and secondary windings have to be defined.

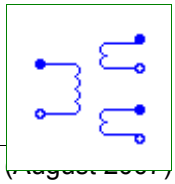
Parameters

Type	Name	Default	Description
Real	n	1	Primary voltage (line-to-line) / Secondary voltage (line-to-line)
Resistance	R1	5E-3/(if C1 == "D" then 1 el...	Warm primary resistance per phase [Ohm]
Inductance	L1sigma	78E-6/(if C1 == "D" then 1 e...	Primary stray inductance per phase [H]
Resistance	R2	5E-3/(if C2 == "d" then 1 el...	Warm secondary resistance per phase [Ohm]
Inductance	L2sigma	78E-6/(if C2 == "d" then 1 e...	Secondary stray inductance per phase [H]
IdealCore	core	redeclare IdealCore core(fin...	

Connectors

Type	Name	Description
PositivePlug	plug1	
NegativePlug	plug2	

Modelica.Electrical.Machines.BasicMachines.Components.PartialCore



Partial model of transformer core with 3 windings

Information

Partial model of transformer core with 3 windings; saturation function flux versus magentizing current has to be defined.

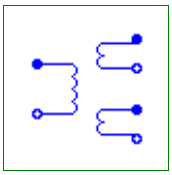
Parameters

Type	Name	Default	Description
Integer	m	3	Number of phases
Real	n12	1	Turns ratio 1:2
Real	n13	2	Turns ratio 1:3

Connectors

Type	Name	Description
PositivePlug	plug_p1	
NegativePlug	plug_n1	
PositivePlug	plug_p2	
NegativePlug	plug_n2	
PositivePlug	plug_p3	
NegativePlug	plug_n3	

Modelica.Electrical.Machines.BasicMachines.Components.IdealCore



Ideal transformer with 3 windings

Information

Ideal transformer with 3 windings: no magnetizing current.

Parameters

Type	Name	Default	Description
Integer	m	3	Number of phases
Real	n12	1	Turns ratio 1:2
Real	n13	2	Turns ratio 1:3

Connectors

Type	Name	Description
PositivePlug	plug_p1	
NegativePlug	plug_n1	
PositivePlug	plug_p2	
NegativePlug	plug_n2	
PositivePlug	plug_p3	
NegativePlug	plug_n3	

Modelica.Electrical.Machines.Sensors

Sensors for machine modelling

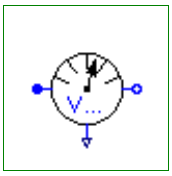
Information

This package contains sensors that are usefull when modelling machines.

Package Content

Name	Description
VoltageRMSsensor	Length of spcae phasor -> RMS voltage
CurrentRMSsensor	Length of spcae phasor -> RMS current
ElectricalPowerSensor	Instantaneous power from spcae phasors
MechanicalPowerSensor	Mechanical power = torque x speed
RotorAngle	Rotor lagging angle

Modelica.Electrical.Machines.Sensors.VoltageRMSsensor



Length of spcae phasor -> RMS voltage

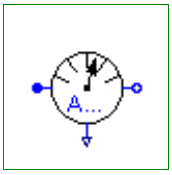
Information

Measured 3-phase instantaneous voltages are transformed to the corresponding space phasor; output is length of the space phasor divided by sqrt(2), thus giving in sinusoidal stationary state RMS voltage.

Connectors

Type	Name	Description
output RealOutput	V	
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.Machines.Sensors.CurrentRMSsensor



Length of spcae phasor -> RMS current

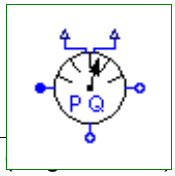
Information

Measured 3-phase instantaneous currents are transformed to the corresponding space phasor; output is length of the space phasor divided by sqrt(2), thus giving in sinusoidal stationary state RMS current.

Connectors

Type	Name	Description
output RealOutput	I	
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.Machines.Sensors.ElectricalPowerSensor



Instantaneous power from spcae phasors

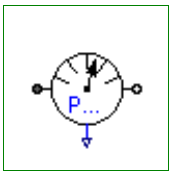
Information

3-phase instantaneous voltages (plug_p - plug_nv) and currents (plug_p - plug_ni) are transformed to the corresponding space phasors,
which are used to calculate power quantities:
P = instantaneous power, thus giving in stationary state active power.
Q = giving in stationary state reactive power.

Connectors

Type	Name	Description
output RealOutput	P	
output RealOutput	Q	
PositivePlug	plug_p	
NegativePlug	plug_ni	
NegativePlug	plug_nv	

Modelica.Electrical.Machines.Sensors.MechanicalPowerSensor



Mechanical power = torque x speed

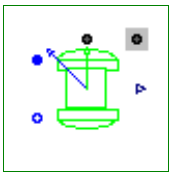
Information

Calculates (mechanical) power from torque times angular speed.

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
output RealOutput	P	

Modelica.Electrical.Machines.Sensors.RotorAngle



Rotor lagging angle

Information

Calculates rotor lagging angle by measuring the stator phase voltages, transforming them to the correspondig space phasor in stator-fixed coordinate system,
rotating the space phasor to the rotor-fixed coordinate system and calculating the angle of this space phasor.

The sensor's housing is implicitely fixed.
Since the machine's stator also implicitely fixed, the angle at the flange is equal to the angle of the machine's rotor against the stator.

Parameters

Type	Name	Default	Description
Integer	p		number of pole pairs

Connectors

Type	Name	Description
output RealOutput	rotorAngle	
PositivePlug	plug_p	
NegativePlug	plug_n	
Flange_a	flange	
Flange_a	support	support at which the reaction torque is acting

---

Modelica.Electrical.Machines.SpacePhasors

Library with space phasor-models

Information

This package contains components, blocks and functions to utilize space phasor theory.

Package Content

Name	Description
Components	Basic space phasor models
Blocks	Blocks for space phasor transformation
Functions	Functions for space phasor transformation

Modelica.Electrical.Machines.SpacePhasors.Components

Basic space phasor models

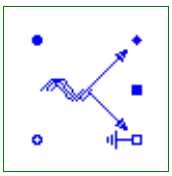
Information

This package contains basic space phasor models.
Real and imaginary part of voltage space phasor are the potentials v_[2] of the space phasor connector; (implicit grounded).
Real and imaginary part of current space phasor are the currents i_[2] at the space phasor connector; a ground has to be used where necessary for currents flowing back.

Package Content

Name	Description
SpacePhasor	Physical transformation: three phase <-> space phasors
Rotator	Rotates space phasor

Modelica.Electrical.Machines.SpacePhasors.Components.SpacePhasor



Physical transformation: three phase <-> space phasors

Information

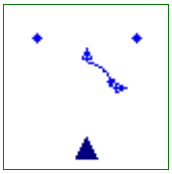
Physical transformation of voltages and currents: three phases <-> space phasors:

x[k] = X0 + {cos(-(k - 1)/m*2*pi),-sin(-(k - 1)/m*2*pi) * X[Re,Im]
and vice versa:
X0 = sum(x[k])/m
X[Re,Im] = sum(2/m*{cos((k - 1)/m*2*pi),sin((k - 1)/m*2*pi)}*x[k])
were x designates three phase values, X[Re,Im] designates the space phasor and X0 designates the zero sequence system.
Physcial transformation means that both voltages and currents are transformed in both directions. Zero-sequence voltage and current are present at pin zero. An additional zero-sequence impedance could be connected between pin zero and pin ground.

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
PositivePin	zero	
NegativePin	ground	
SpacePhasor	spacePhasor	

Modelica.Electrical.Machines.SpacePhasors.Components.Rotator



Rotates space phasor

Information

Rotates space phasors of left connector to right connector by the angle provided by the input signal "angle" from one coordinate system into another.

Connectors

Type	Name	Description
SpacePhasor	spacePhasor_a	
SpacePhasor	spacePhasor_b	
input RealInput	angle	

Modelica.Electrical.Machines.SpacePhasors.Blocks

Blocks for space phasor transformation

Information

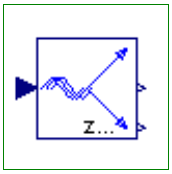
This package contains space phasor transformation blocks for use in controllers:

- ToSpacePhasor: transforms a set of threephase values to space phasor and zero sequence system
- FromSpacePhasor: transforms a space phasor and zero sequence system to a set of threephase values
- Rotator: rotates a space phasor (from one coordinate system into another)
- ToPolar: Converts a space phasor from rectangular coordinates to polar coordinates
- FromPolar: Converts a space phasor from polar coordinates to rectangular coordinates

Package Content

Name	Description
ToSpacePhasor	Conversion: three phase -> space phasor
FromSpacePhasor	Conversion: space phasor -> three phase
Rotator	Rotates space phasor
ToPolar	Converts a space phasor to polar coordinates
FromPolar	Converts a space phasor from polar coordinates

Modelica.Electrical.Machines.SpacePhasors.Blocks.ToSpacePhasor



Conversion: three phase -> space phasor

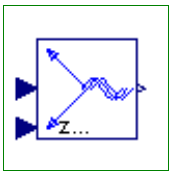
Information

Transformation of threephase values (voltages or currents) to space phasor and zero sequence value.

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y[nout]	Connector of Real output signals
output RealOutput	zero	

Modelica.Electrical.Machines.SpacePhasors.Blocks.FromSpacePhasor



Conversion: space phasor -> three phase

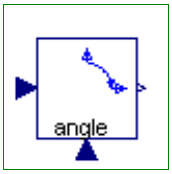
Information

Transformation of space phasor and zero sequence value to threephase values (voltages or currents).

Connectors

Type	Name	Description
input RealInput	u[nin]	Connector of Real input signals
output RealOutput	y[nout]	Connector of Real output signals
input RealInput	zero	

Modelica.Electrical.Machines.SpacePhasors.Blocks.Rotator



Rotates space phasor

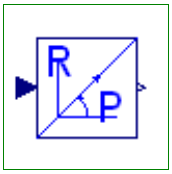
Information

Rotates a space phasor (voltage or current) by the angle provided by the input signal "angle" from one coordinate system into another.

Connectors

Type	Name	Description
input RealInput	u[n]	Connector of Real input signals
output RealOutput	y[n]	Connector of Real output signals
input RealInput	angle	

Modelica.Electrical.Machines.SpacePhasors.Blocks.ToPolar



Converts a space phasor to polar coordinates

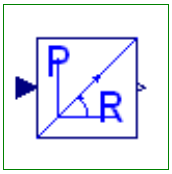
Information

Converts a space phasor from rectangular coordinates to polar coordinates.

Connectors

Type	Name	Description
input RealInput	u[n]	Connector of Real input signals
output RealOutput	y[n]	Connector of Real output signals

Modelica.Electrical.Machines.SpacePhasors.Blocks.FromPolar



Converts a space phasor from polar coordinates

Information

Converts a space phasor from polar coordinates to rectangular coordinates.

Connectors

Type	Name	Description
input RealInput	u[n]	Connector of Real input signals
output RealOutput	y[n]	Connector of Real output signals

Modelica.Electrical.Machines.SpacePhasors.Functions

Functions for space phasor transformation

Information

This package contains space phasor transformation functions for use in calculations:

- ToSpacePhasor: transforms a set of threephase values to space phasor and zero sequence system
- FromSpacePhasor: transforms a space phasor and zero sequence system to a set of threephase values
- Rotator: rotates a space phasor (from one coordinate system into another)
- ToPolar: Converts a space phasor from rectangular coordinates to polar coordinates
- FromPolar: Converts a space phasor from polar coordinates to rectangular coordinates

Package Content

Name	Description
ToSpacePhasor	Conversion: three phase -> space phasor
FromSpacePhasor	Conversion: space phasor -> three phase
Rotator	Rotates space phasor
ToPolar	Converts a space phasor to polar coordinates
FromPolar	Converts a space phasor from polar coordinates

Modelica.Electrical.Machines.SpacePhasors.Functions.ToSpacePhasor

Conversion: three phase -> space phasor

Information

Transformation of three phase values (voltages or currents) to space phasor and zero sequence value:
y[k] = X0 + {cos(-(k - 1)/m*2*pi),-sin(-(k - 1)/m*2*pi) * X[Re,Im]
were y designates three phase values, X[Re,Im] designates the space phasor and X0 designates the zero sequence system.

Inputs

Type	Name	Default	Description
Real	x[3]		

Outputs

Type	Name	Description
Real	y[2]	
Real	y0	

Modelica.Electrical.Machines.SpacePhasors.Functions.FromSpacePhasor

Conversion: space phasor -> three phase

Information

Transformation of space phasor and zero sequence value to three phase values (voltages or currents):
Y0 = sum(x[k])/m
Y[Re,Im] = sum(2/m*{cos((k - 1)/m*2*pi),sin((k - 1)/m*2*pi)}*x[k])
were x designates three phase values, Y[Re,Im] designates the space phasor and Y0 designates the zero sequence system.

Inputs

Type	Name	Default	Description
Real	x[2]		
Real	x0		

Outputs

Type	Name	Description
Real	y[3]	

Modelica.Electrical.Machines.SpacePhasors.Functions.Rotator

Rotates space phasor

Information

Rotates a space phasor (voltage or current) by the angle provided by input argument "angle" from one coordinate system into another:
y[Re,Im] := {{+cos(-angle),-sin(-angle)},{+sin(-angle),+cos(-angle)}}*x[Re,Im]
where y[Re,Im] designates the space phasor in the new coordinate system (twisted by angle against old coordinate system) and y[Re,Im] designates the space phasor in the old coordinate system.

Inputs

Type	Name	Default	Description
Real	x[2]		
Angle	angle		[rad]

Outputs

Type	Name	Description
Real	y[2]	

Modelica.Electrical.Machines.SpacePhasors.Functions.ToPolar

Converts a space phasor to polar coordinates

Information

Converts a space phasor from rectangular coordinates to polar coordinates, providing angle=0 for {0,0}.

Inputs

Type	Name	Default	Description
Real	x[2]		

Outputs

Type	Name	Description
Real	absolute	
Angle	angle	[rad]

Modelica.Electrical.Machines.SpacePhasors.Functions.FromPolar

Converts a space phasor from polar coordinates

Information

Converts a space phasor from polar coordinates to rectangular coordinates.

Inputs

Type	Name	Default	Description
Real	absolute		
Angle	angle		[rad]

Outputs

Type	Name	Description
Real	x[2]	

Modelica.Electrical.Machines.Interfaces

SpacePhasor connector and PartialMachines

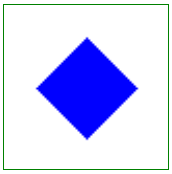
Information

This package contains the space phasor connector and partial models for machine models.

Package Content

Name	Description
SpacePhasor	Connector for Space Phasors
Adapter	From Modelica.Mechanics.Rotational.Interfaces.TwoFlangesAndBearingH
PartialBasicMachine	Partial model for all machines
PartialBasicInductionMachine	Partial model for induction machine
PartialBasicDCMachine	Partial model for DC machine

Modelica.Electrical.Machines.Interfaces.SpacePhasor



Connector for Space Phasors

Information

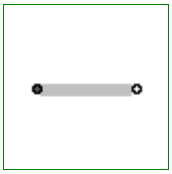
Connector for Space Phasors:

- Voltage v_[2] ... Real and Imaginary part of voltage space phasor
- Current i_[2] ... Real and Imaginary part of current space phasor

Contents

Type	Name	Description
Voltage	v_[2]	[V]
flow Current	i_[2]	[A]

Modelica.Electrical.Machines.Interfaces.Adapter



From Modelica.Mechanics.Rotational.Interfaces.TwoFlangesAndBearingH

Information

From Modelica.Mechanics.Rotational.Interfaces.TwoFlangesAndBearingH:
If bearingConnected = true torque is given from flange_a to flange_b
If bearingConnected = false flange_a is fixed

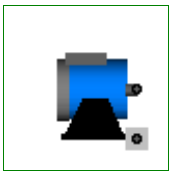
Parameters

Type	Name	Default	Description
Boolean	bearingConnected		Choose wether bearing is connected

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	

Modelica.Electrical.Machines.Interfaces.PartialBasicMachine



Partial model for all machines

Information

Base partial model DC machines:

- main parts of the icon
- mechanical flange
- mechanical support

Besides the mechanical connector *flange_a* (i.e. the shaft) the machines have a second mechanical connector *support*.
If nothing is connected to *support*, it is assumed that the stator is fixed.
Otherwise reaction torque (i.e. airGap torque, minus acceleration torque for stator's moment of inertia) can be measured at *support*.
One may also fix the the shaft and let rotate the stator; parameter Js is only of importance when the stator is rotating.
Take care: Even if You connect only a sensor (e.g. Modelica.Mechanics.Rotational.Sensors.RelAngleSensor or Machines.Sensors.RotorDisplacementAngle) to the *support*, You have to fix the *support*!

Parameters

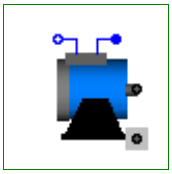
Type	Name	Default	Description
Inertia	J_Rotor		rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting

---

Modelica.Electrical.Machines.Interfaces.PartialBasicInductionMachine



Partial model for induction machine

Information

Partial model for induction machine models, containing:

- main parts of the icon
- stator plugs
- mechanical connectors

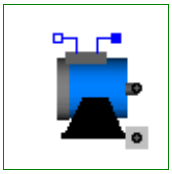
Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.29	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]
Integer	p	2	number of pole pairs (Integer)

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePlug	plug_sp	
NegativePlug	plug_sn	

Modelica.Electrical.Machines.Interfaces.PartialBasicDCMachine



Partial model for DC machine

Information

Partial model for DC machine models, containing:

- main parts of the icon
- armature pins
- mechanical connectors

Parameters

Type	Name	Default	Description
Inertia	J_Rotor	0.15	rotor's moment of inertia [kg.m2]
Inertia	J_Stator	J_Rotor	stator's moment of inertia [kg.m2]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_a	support	support at which the reaction torque is acting
PositivePin	pin_ap	
NegativePin	pin_an	

Modelica.Electrical.MultiPhase

Library for electrical components with 2, 3 or more phases

Information

This package contains packages for electrical multiphase components, based on Modelica.Electrical.Analog:

- Basic: basic components (resistor, capacitor, inductor, ...)
- Ideal: ideal elements (switches, diode, transformer, ...)
- Sensors: sensors to measure potentials, voltages, and currents
- Sources: time-dependend and controlled voltage and current sources

This package is intended to be used the same way as Modelica.Electrical.Analog but to make design of multiphase models easier.
The package is based on the plug: a composite connector containing m pins.
It is possible to connect plugs to plugs or single pins of a plug to single pins.
Potentials may be accessed as `plug.pin[].v`, currents may be accessed as `plug.pin[].i`.

Further development:

- temperature-dependent resistor
- lines (m-phase models)

Main Author:
Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern
Austria
email: a.haumer@haumer.at

Copyright © 1998-2007, Modelica Association and Anton Haumer.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Basic	Basic components for electrical multiphase models
Examples	Multiphase test examples
Ideal	Multiphase components with idealized behaviour
Interfaces	Interfaces for electrical multiphase models
Sensors	Multiphase potential, voltage and current Sensors
Sources	Multiphase voltage and current sources

Modelica.Electrical.MultiPhase.Basic

Basic components for electrical multiphase models

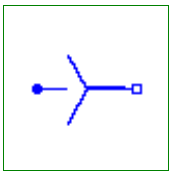
Information

This package contains basic analog electrical multiphase components.

Package Content

Name	Description
Star	Star-connection
Delta	Delta (polygon) connection
PlugToPin_p	Connect one (positive) Pin
PlugToPin_n	Connect one (negative) Pin
Resistor	Ideal linear electrical resistors
Conductor	Ideal linear electrical conductors
Capacitor	Ideal linear electrical capacitors
Inductor	Ideal linear electrical inductors
SaturatingInductor	Simple model of inductors with saturation
Transformer	Multiphase Transformer
VariableResistor	Ideal linear electrical resistors with variable resistance
VariableConductor	Ideal linear electrical conductors with variable conductance
VariableCapacitor	Ideal linear electrical capacitors with variable capacitance
VariableInductor	Ideal linear electrical inductors with variable inductance

Modelica.Electrical.MultiPhase.Basic.Star



Star-connection

Information

Connects all pins of plug_p to pin_n, thus establishing a so-called star-connection.

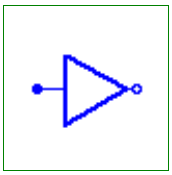
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePin	pin_n	

Modelica.Electrical.MultiPhase.Basic.Delta



Delta (polygon) connection

Information

Connects in a cyclic way plug_n.pin[j] to plug_p.pin[j+1], thus establishing a so-called delta (or polygon) connection when used in parallel to another component.

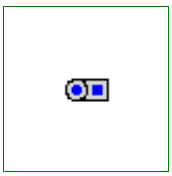
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Basic.PlugToPin_p



Connect one (positive) Pin

Information

Connects pin *k* of plug_p to pin_p, leaving the other pins of plug_p unconnected.

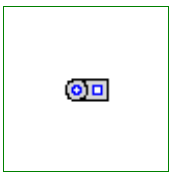
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Integer	k	1	phase index

Connectors

Type	Name	Description
PositivePlug	plug_p	
PositivePin	pin_p	

Modelica.Electrical.MultiPhase.Basic.PlugToPin_n



Connect one (negative) Pin

Information

Connects pin *k* of plug_n to pin_n, leaving the other pins of plug_n unconnected.

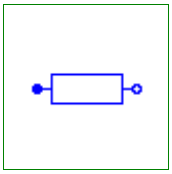
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Integer	k	1	phase index

Connectors

Type	Name	Description
NegativePlug	plug_n	
NegativePin	pin_n	

Modelica.Electrical.MultiPhase.Basic.Resistor



Ideal linear electrical resistors

Information

Contains m resistors (Modelica.Electrical.Analog.Basic.Resistor)

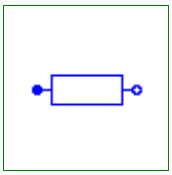
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Resistance	R[m]	fill(1, m)	Resistance [Ohm]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Basic.Conductor



Ideal linear electrical conductors

Information

Contains m conductors (Modelica.Electrical.Analog.Basic.Conductor)

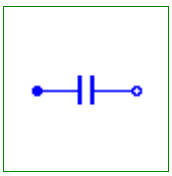
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Conductance	G[m]	fill(1, m)	Conductance [S]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Basic.Capacitor



Ideal linear electrical capacitors

Information

Contains m capacitors (Modelica.Electrical.Analog.Basic.Capacitor)

Parameters

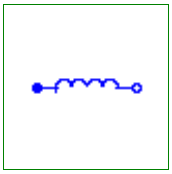
Type	Name	Default	Description
Integer	m	3	number of phases

Capacitance	C[m]	fill(1, m)	Capacitance [F]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Basic.Inductor



Ideal linear electrical inductors

Information

Contains m inductors (Modelica.Electrical.Analog.Basic.Inductor)

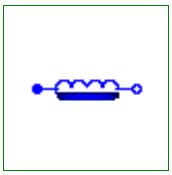
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Inductance	L[m]	fill(1, m)	Inductance [H]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Basic.SaturatingInductor



Simple model of inductors with saturation

Information

Contains m saturating inductors (Modelica.Electrical.Analog.Basic.SaturatingInductor)

Attention!!!
Each element of the array of saturatingInductors is only dependent on the current flowing through this element.

Parameters

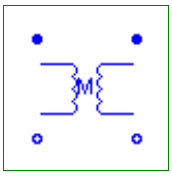
Type	Name	Default	Description
Integer	m	3	number of phases
Current	Inom[m]	fill(1, m)	Nominal current [A]
Inductance	Lnom[m]	fill(1, m)	Nominal inductance at Nominal current [H]
Inductance	Lzer[m]	{2*Lnom[j] for j in 1:m}	Inductance near current=0 [H]
Inductance	Linf[m]	{Lnom[j]/2 for j in 1:m}	Inductance at large currents [H]

Connectors

Type	Name	Description

PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Basic.Transformer



Multiphase Transformer

Information

Contains m transformers (Modelica.Electrical.Analog.Basic.Transformer)

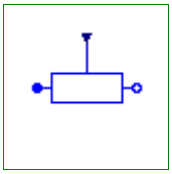
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Inductance	L1[m]	fill(1, m)	Primary inductance [H]
Inductance	L2[m]	fill(1, m)	Secondary inductance [H]
Inductance	M[m]	fill(1, m)	Coupling inductance [H]

Connectors

Type	Name	Description
PositivePlug	plug_p1	
PositivePlug	plug_p2	
NegativePlug	plug_n1	
NegativePlug	plug_n2	

Modelica.Electrical.MultiPhase.Basic.VariableResistor



Ideal linear electrical resistors with variable resistance

Information

Contains m variable resistors (Modelica.Electrical.Analog.Basic.VariableResistor)

Attention!!!
It is recomended that none of the R_Port signals should not cross the zero value. Otherwise depending on the surrounding circuit the probability of singularities is high.

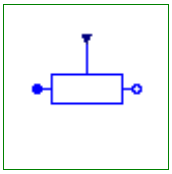
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input RealInput	R[m]	

Modelica.Electrical.MultiPhase.Basic.VariableConductor



Ideal linear electrical conductors with variable conductance

Information

Contains m variable conductors (Modelica.Electrical.Analog.Basic.VariableConductor)

Attention!!!
It is recomended that none of the G_Port signals should not cross the zero value. Otherwise depending on the surrounding circuit the probability of singularities is high.

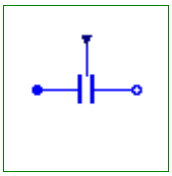
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input RealInput	G[m]	

Modelica.Electrical.MultiPhase.Basic.VariableCapacitor



Ideal linear electrical capacitors with variable capacitance

Information

Contains m variable capacitors (Modelica.Electrical.Analog.Basic.VariableCapacitor)

It is required that each C_Port.signal ≥ 0, otherwise an assertion is raised. To avoid a variable index system, C = Cmin, if 0 ≤ C_Port.signal < Cmin, where Cmin is a parameter with default value Modelica.Constants.eps.

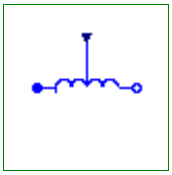
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Capacitance	Cmin[m]	fill(Modelica.Constants.eps,...	minimum Capacitance [F]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input RealInput	C[m]	

Modelica.Electrical.MultiPhase.Basic.VariableInductor



Ideal linear electrical inductors with variable inductance

Information

Contains m variable inductors (Modelica.Electrical.Analog.Basic.VariableInductor)

It is required that each L_Port.signal ≥ 0, otherwise an assertion is raised. To avoid a variable index system, L = Lmin, if 0 ≤ L_Port.signal < Lmin, where Lmin is a parameter with default value Modelica.Constants.eps.

Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Inductance	Lmin[m]	fill(Modelica.Constants.eps,...	minimum Inductance [H]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input RealInput	L[m]	

Modelica.Electrical.MultiPhase.Examples

Multiphase test examples

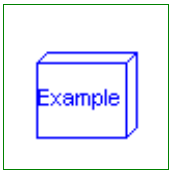
Information

This package contains test examples of analog electrical multiphase circuits.

Package Content

Name	Description
TransformerYY	Test example with multiphase components
TransformerYD	Test example with multiphase components
Rectifier	Test example with multiphase components

Modelica.Electrical.MultiPhase.Examples.TransformerYY



Test example with multiphase components

Information

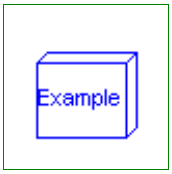
Test example with multiphase components:
Star-connected voltage source feeds via a Y-Y-transformer with internal impedance (RT, LT) a load resistor RT.
Using f=5 Hz LT=3mH defines nominal voltage drop of approximately 10 %.
Simulate for 1 second (2 periods) and compare voltages and currents of source, transformer and load.

Parameters

Type	Name	Default	Description
Integer	m	3	Number of phases

Voltage	V	1	Amplitude of Star-Voltage [V]
Frequency	f	5	Frequency [Hz]
Inductance	LT	0.003	Transformer stray inductance [H]
Resistance	RT	0.05	Transformer resistance [Ohm]
Resistance	RL	1	Load Resistance [Ohm]

Modelica.Electrical.MultiPhase.Examples.TransformerYD



Test example with multiphase components

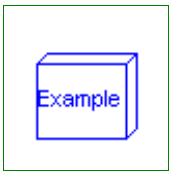
Information

Test example with multiphase components:
Star-connected voltage source feeds via a Y-D-transformer with internal impedance (RT, LT) a load resistor RT.
Using f=5 Hz LT=3mH defines nominal voltage drop of approximately 10 %.
Simulate for 1 second (2 periods) and compare voltages and currents of source, transformer and load.

Parameters

Type	Name	Default	Description
Integer	m	3	Number of phases
Voltage	V	1	Amplitude of Star-Voltage [V]
Frequency	f	5	Frequency [Hz]
Inductance	LT	0.003	Transformer stray inductance [H]
Resistance	RT	0.05	Transformer resistance [Ohm]
Resistance	RL	1	Load Resistance [Ohm]
Real	nT	1/sqrt((1 - Modelica.Math.co...	Transformer ratio

Modelica.Electrical.MultiPhase.Examples.Rectifier



Test example with multiphase components

Information

Test example with multiphase components:
Star-connected voltage source feeds via a line reactor a diode bridge rectifier with a DC burden.
Using f=5 Hz, simulate for 1 second (2 periods) and compare voltages and currents of source and DC burden, neglecting initial transient.

Parameters

Type	Name	Default	Description
Integer	m	3	Number of phases
Voltage	V	1	Amplitude of Star-Voltage [V]
Frequency	f	5	Frequency [Hz]
Inductance	L	0.001	Line Inductance [H]
Resistance	RL	2	Load Resistance [Ohm]
Capacitance	C	0.05	Total DC-Capacitance [F]

Resistance	RE	1E6	Earthing Resistance [Ohm]

Modelica.Electrical.MultiPhase.Ideal

Multiphase components with idealized behaviour

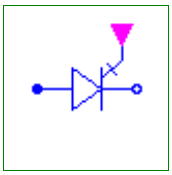
Information

This package contains analog electrical multiphase components with idealized behaviour, like thyristor, diode, switch, transformer.

Package Content

Name	Description
IdealThyristor	Multiphase ideal thyristor
IdealGTOThyristor	Multiphase ideal GTO thyristor
IdealCommutingSwitch	Multiphase ideal commuting switch
IdealIntermediateSwitch	Multiphase ideal intermediate switch
IdealDiode	Multiphase ideal diode
IdealTransformer	Multiphase ideal transformer
Idle	Multiphase idle branch
Short	Multiphase short cut branch
IdealOpeningSwitch	Multiphase ideal opener
IdealClosingSwitch	Multiphase ideal closer

Modelica.Electrical.MultiPhase.Ideal.IdealThyristor



Multiphase ideal thyristor

Information

Contains m ideal thyristors (Modelica.Electrical.Analog.Ideal.IdealThyristor).

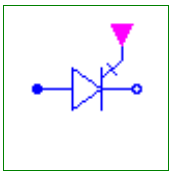
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Resistance	Ron[m]	fill(1.E-5, m)	Closed thyristor resistance [Ohm]
Conductance	Goff[m]	fill(1.E-5, m)	Opened thyristor conductance [S]
Voltage	Vknee[m]	zeros(m)	Treshold voltage [V]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input BooleanInput	fire[m]	

Modelica.Electrical.MultiPhase.Ideal.IdealGTOThyristor



Multiphase ideal GTO thyristor

Information

Contains m ideal GTO thyristors (Modelica.Electrical.Analog.Ideal.IdealGTOThyristor).

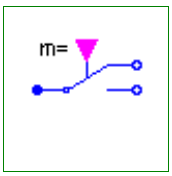
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Resistance	Ron[m]	fill(1.E-5, m)	Closed thyristor resistance [Ohm]
Conductance	Goff[m]	fill(1.E-5, m)	Opened thyristor conductance [S]
Voltage	Vknee[m]	zeros(m)	Treshold voltage [V]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input BooleanInput	fire[m]	

Modelica.Electrical.MultiPhase.Ideal.IdealCommutingSwitch



Multiphase ideal commuting switch

Information

Contains m ideal commuting switches (Modelica.Electrical.Analog.Ideal.IdealCommutingSwitch).

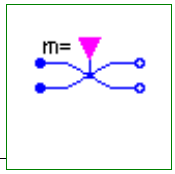
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Resistance	Ron[m]	fill(1.E-5, m)	Closed switch resistance [Ohm]
Conductance	Goff[m]	fill(1.E-5, m)	Opened switch conductance [S]

Connectors

Type	Name	Description
input BooleanInput	control[m]	true => p--n2 connected, false => p--n1 connected
PositivePlug	plug_p	
NegativePlug	plug_n2	
NegativePlug	plug_n1	

Modelica.Electrical.MultiPhase.Ideal.IdealIntermediateSwitch



Multiphase ideal intermediate switch

Information

Contains m ideal intermediate switches (Modelica.Electrical.Analog.Ideal.IdealIntermediateSwitch).

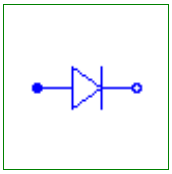
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Resistance	Ron[m]	fill(1.E-5, m)	Closed switch resistance [Ohm]
Conductance	Goff[m]	fill(1.E-5, m)	Opened switch conductance [S]

Connectors

Type	Name	Description
input BooleanInput	control[m]	true => p1--n2, p2--n1 connected, otherwise p1--n1, p2--n2 connected
PositivePlug	plug_p1	
PositivePlug	plug_p2	
NegativePlug	plug_n2	
NegativePlug	plug_n1	

Modelica.Electrical.MultiPhase.Ideal.IdealDiode



Multiphase ideal diode

Information

Contains m ideal diodes (Modelica.Electrical.Analog.Ideal.IdealDiode).

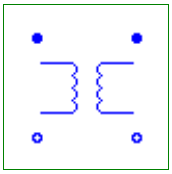
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Resistance	Ron[m]	fill(1.E-5, m)	Closed diode resistance [Ohm]
Conductance	Goff[m]	fill(1.E-5, m)	Opened diode conductance [S]
Voltage	Vknee[m]	zeros(m)	Treshold voltage [V]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Ideal.IdealTransformer



Multiphase ideal transformer

Information

Contains m ideal transformers (Modelica.Electrical.Analog.Ideal.IdealTransformer).

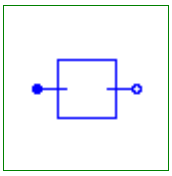
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Real	n[m]	fill(1, m)	Turns ratio

Connectors

Type	Name	Description
PositivePlug	plug_p1	
PositivePlug	plug_p2	
NegativePlug	plug_n1	
NegativePlug	plug_n2	

Modelica.Electrical.MultiPhase.Ideal.Idle



Multiphase idle branch

Information

Contains m idles (Modelica.Electrical.Analog.Ideal.Idle)

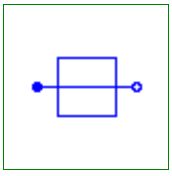
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Ideal.Short



Multiphase short cut branch

Information

Contains m short cuts (Modelica.Electrical.Analog.Ideal.Short)

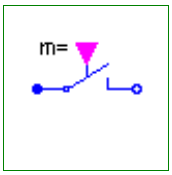
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Ideal.IdealOpeningSwitch



Multiphase ideal opener

Information

Contains m ideal opening switches (Modelica.Electrical.Analog.Ideal.IdealOpeningSwitch).

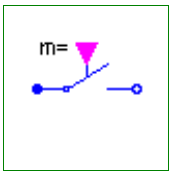
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Resistance	Ron[m]	fill(1.E-5, m)	Closed switch resistance [Ohm]
Conductance	Goff[m]	fill(1.E-5, m)	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input BooleanInput	control[m]	true => switch open, false => p--n connected

Modelica.Electrical.MultiPhase.Ideal.IdealClosingSwitch



Multiphase ideal closer

Information

Contains m ideal closing switches (Modelica.Electrical.Analog.Ideal.IdealClosingSwitch).

< /HTML>*Error:Found no end-tag in HTML-documentation*

Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Resistance	Ron[m]	fill(1.E-5, m)	Closed switch resistance [Ohm]
Conductance	Goff[m]	fill(1.E-5, m)	Opened switch conductance [S]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input BooleanInput	control[m]	true => p--n connected, false => switch open

Modelica.Electrical.MultiPhase.Interfaces

Interfaces for electrical multiphase models

Information

This package contains connectors and interfaces (partial models) for electrical multiphase components, based on Modelica.Electrical.Analog.

Package Content

Name	Description
Plug	Plug with m pins for an electric component
PositivePlug	Positive plug with m pins
NegativePlug	Negative plug with m pins
TwoPlug	Component with one m-phase electric port
OnePort	Component with two electrical plugs and currents from plug_p to plug_n
FourPlug	Component with two m-phase electric ports
TwoPort	Component with two m-phase electric ports, including currents

---

Modelica.Electrical.MultiPhase.Interfaces.Plug

Plug with m pins for an electric component

Information

Connectors PositivePlug and NegativePlug are nearly identical. The only difference is that the icons are different in order to identify more easily the plugs of a component. Usually, connector PositivePlug is used for the positive and connector NegativePlug for the negative plug of an electrical component.
Connector Plug is a composite connector containing m Pins (Modelica.Electrical.Analog.Interfaces.Pin).

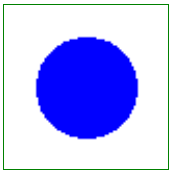
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Contents

Type	Name	Description
Integer	m	number of phases
Pin	pin[m]	

Modelica.Electrical.MultiPhase.Interfaces.PositivePlug



Positive plug with m pins

Information

Connectors PositivePlug and NegativePlug are nearly identical. The only difference is that the icons are different in order to identify more easily the plugs of a component. Usually, connector PositivePlug is used for the positive and connector NegativePlug for the negative plug of an electrical component.
Connector Plug is a composite connector containing m Pins (Modelica.Electrical.Analog.Interfaces.Pin).

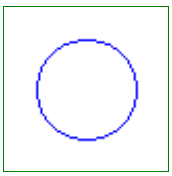
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Contents

Type	Name	Description
Integer	m	number of phases
Pin	pin[m]	

Modelica.Electrical.MultiPhase.Interfaces.NegativePlug



Negative plug with m pins

Information

Connectors PositivePlug and NegativePlug are nearly identical. The only difference is that the icons are different in order to identify more easily the plugs of a component. Usually, connector PositivePlug is used for the positive and connector NegativePlug for the negative plug of an electrical component.
Connector Plug is a composite connector containing m Pins (Modelica.Electrical.Analog.Interfaces.Pin).

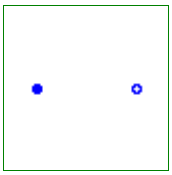
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Contents

Type	Name	Description
Integer	m	number of phases
Pin	pin[m]	

Modelica.Electrical.MultiPhase.Interfaces.TwoPlug



Component with one m-phase electric port

Information

Superclass of elements which have **two** electrical plugs: the positive plug connector *plug_p*, and the negative plug connector *plug_n*. The currents flowing into plug_p are provided explicitly as currents i[m].

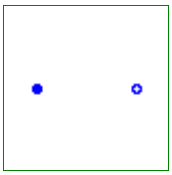
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Interfaces.OnePort



Component with two electrical plugs and currents from plug_p to plug_n

Information

Superclass of elements which have **two** electrical plugs: the positive plug connector *plug_p*, and the negative plug connector *plug_n*. The currents flowing into plug_p are provided explicitly as currents i[m]. It is assumed that the currents flowing into plug_p are identical to the currents flowing out of plug_n.

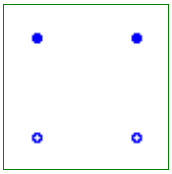
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Interfaces.FourPlug



Component with two m-phase electric ports

Information

Superclass of elements which have **four** electrical plugs.

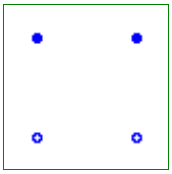
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p1	
PositivePlug	plug_p2	
NegativePlug	plug_n1	
NegativePlug	plug_n2	

Modelica.Electrical.MultiPhase.Interfaces.TwoPort



Component with two m-phase electric ports, including currents

Information

Superclass of elements which have **four** electrical plugs. It is assumed that the currents flowing into plug_p1 are identical to the currents flowing out of plug_n1, and that the currents flowing into plug_p2 are identical to the currents flowing out of plug_n2.

Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p1	
PositivePlug	plug_p2	
NegativePlug	plug_n1	
NegativePlug	plug_n2	

Modelica.Electrical.MultiPhase.Sensors

Multiphase potential, voltage and current Sensors

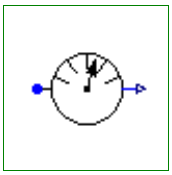
Information

This package contains multiphase potential, voltage, and current sensors.

Package Content

Name	Description
PotentialSensor	Multiphase potential sensor
VoltageSensor	Multiphase voltage sensor
CurrentSensor	Multiphase current sensor
PowerSensor	Multiphase instantaneous power sensor

Modelica.Electrical.MultiPhase.Sensors.PotentialSensor



Multiphase potential sensor

Information

Contains m potential sensors (Modelica.Electrical.Analog.Sensors.PotentialSensor), thus measuring the m potentials *phi[m]* of the m pins of plug_p.

Parameters

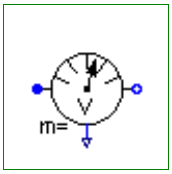
Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
output RealOutput	phi[m]	Absolute voltage potential as output signal

---

Modelica.Electrical.MultiPhase.Sensors.VoltageSensor



Multiphase voltage sensor

Information

Contains m voltage sensors (Modelica.Electrical.Analog.Sensors.VoltageSensor), thus measuring the m potential differences *v[m]* between the m pins of plug_p and plug_n.

Parameters

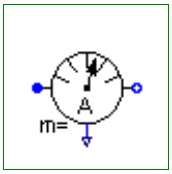
Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
output RealOutput	v[m]	Voltage between pin p and n (= p.v - n.v) as output signal

---

Modelica.Electrical.MultiPhase.Sensors.CurrentSensor



Multiphase current sensor

Information

Contains m current sensors (Modelica.Electrical.Analog.Sensors.CurrentSensor), thus measuring the m currents *i[m]* flowing from the m pins of plug_p to the m pins of plug_n.

Parameters

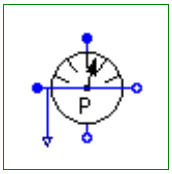
Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
output RealOutput	i[m]	current in the branch from p to n as output signal

---

Modelica.Electrical.MultiPhase.Sensors.PowerSensor



Multiphase instantaneous power sensor

Information

This power sensor measures instantaneous electrical power of a multiphase system and has a separated voltage and current path. The plugs of the voltage path are `pv` and `nv`, the plugs of the current path are `pc` and `nc`. The internal resistance of each current path is zero, the internal resistance of each voltage path is infinite.

Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	pc	Positive plug, current path
NegativePlug	nc	Negative plug, current path
PositivePlug	pv	Positive plug, voltage path
NegativePlug	nv	Negative plug, voltage path
output RealOutput	power	

Modelica.Electrical.MultiPhase.Sources

Multiphase voltage and current sources

Information

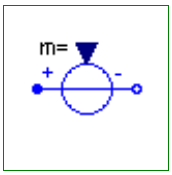
This package contains time-dependend and controlled multiphase voltage and current sources:

- SignalVoltage: fed by Modelica.Blocks.Sources arbitrary waveforms of voltages are possible
- SineVoltage : phase shift between consecutive voltages by default = `pi/m`
- SignalCurrent: fed by Modelica.Blocks.Sources arbitrary waveforms of currents are possible
- SineCurrent : phase shift between consecutive currents by default = `pi/m`

Package Content

Name	Description
SignalVoltage	Multiphase signal voltage source
ConstantVoltage	Multiphase constant voltage source
SineVoltage	Multiphase sine voltage source
SignalCurrent	Multiphase sine current source
ConstantCurrent	Multiphase constant current source
SineCurrent	Multiphase sine current source

Modelica.Electrical.MultiPhase.Sources.SignalVoltage



Multiphase signal voltage source

Information

Contains m signal controlled voltage sources (Modelica.Electrical.Analog.Sources.SignalVoltage)

Parameters

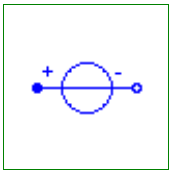
Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input RealInput	v[m]	Voltage between pin p and n (= p.v - n.v) as input signal

---

Modelica.Electrical.MultiPhase.Sources.ConstantVoltage



Multiphase constant voltage source

Information

Contains m constant voltage sources (Modelica.Electrical.Analog.Sources.ConstantVoltage)

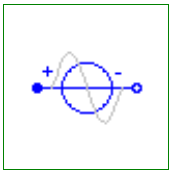
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Voltage	V[m]	fill(1, m)	Value of constant voltage [V]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Sources.SineVoltage



Multiphase sine voltage source

Information

Contains m sine voltage sources (Modelica.Electrical.Analog.Sources.SineVoltage) with a default phase shift of -(j-1)/m * 2*pi for j in 1:m.

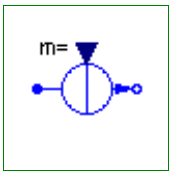
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Voltage	V[m]	fill(1, m)	Amplitudes of sine waves [V]
Angle	phase[m]	-{(j - 1)/m*2*Modelica.Const...	Phases of sine waves [rad]
Frequency	freqHz[m]	fill(1, m)	Frequencies of sine waves [Hz]
Voltage	offset[m]	zeros(m)	Voltage offsets [V]
Time	startTime[m]	zeros(m)	Time offsets [s]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Sources.SignalCurrent



Multiphase sine current source

Information

Contains m signal controlled current sources (Modelica.Electrical.Analog.Sources.SignalCurrent)

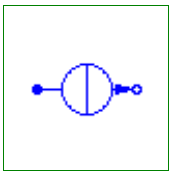
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	
input RealInput	i[m]	Current flowing from pin p to pin n as input signal

Modelica.Electrical.MultiPhase.Sources.ConstantCurrent



Multiphase constant current source

Information

Contains m constant current sources (Modelica.Electrical.Analog.Sources.ConstantCurrent)

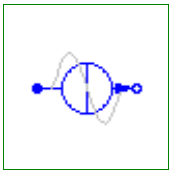
Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Current	I[m]	fill(1, m)	Value of constant current [A]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Electrical.MultiPhase.Sources.SineCurrent



Multiphase sine current source

Information

Contains m sine current sources (Modelica.Electrical.Analog.Sources.SineCurrent) with a default phase shift of -(j-1)/m * 2*pi for j in 1:m.

Parameters

Type	Name	Default	Description
Integer	m	3	number of phases
Current	I[m]	fill(1, m)	Amplitudes of sine waves [A]
Angle	phase[m]	-{(j - 1)/m*2*Modelica.Const...	Phases of sine waves [rad]
Frequency	freqHz[m]	fill(1, m)	Frequencies of sine waves [Hz]
Current	offset[m]	zeros(m)	Current offsets [A]
Time	startTime[m]	zeros(m)	Time offsets [s]

Connectors

Type	Name	Description
PositivePlug	plug_p	
NegativePlug	plug_n	

Modelica.Icons

Library of icons

Information

This package contains definitions for the graphical layout of components which may be used in different libraries. The icons can be utilized by inheriting them in the desired class using "extends" or by directly copying the "icon" layer.

Main Author:

Martin Otter
Deutsches Zentrum fuer Luft und Raumfahrt e.V. (DLR)
Oberpfaffenhofen
Postfach 1116
D-82230 Wessling
email: Martin.Otter@dlr.de

Copyright © 1998-2007, Modelica Association and DLR.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Info	Icon for an information class
Library	Icon for library
Library2	Icon for library where additional icon elements shall be added
Example	Icon for an example model
Function	Icon for a function
Record	Icon for a record
Enumeration	Icon for an enumeration (emulated by a package)

TypeReal	Icon for a Real type
TypeInteger	Icon for an Integer type
TypeBoolean	Icon for a Boolean type
TypeString	Icon for a String type
TranslationalSensor	Icon representing translational measurement device
RotationalSensor	Icon representing rotational measurement device
GearIcon	Icon for gearbox
MotorIcon	Icon for electrical motor
SignalBus	Icon for signal bus
SignalSubBus	Icon for signal sub-bus

Types and constants

```
type TypeReal "Icon for a Real type"
    extends Real;
end TypeReal;

type TypeInteger "Icon for an Integer type"
    extends Integer;
end TypeInteger;

type TypeBoolean "Icon for a Boolean type"
    extends Boolean;
end TypeBoolean;

type TypeString "Icon for a String type"
    extends String;
end TypeString;
```

Modelica.Icons.Info

Icon for an information class

Information

This icon is designed for an **information** class.

Modelica.Icons.Library

Icon for library

Information

This icon is designed for a **library**.

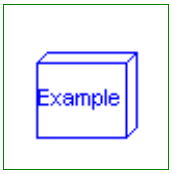
Modelica.Icons.Library2

Icon for library where additional icon elements shall be added

Information

This icon is designed for a **package** where a package specific graphic is additionally included in the icon.

Modelica.Icons.Example



Icon for an example model

Information

This icon is designed for an **Example package**, i.e. a package containing executable demo models.

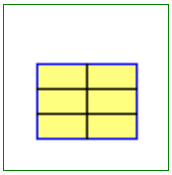
Modelica.Icons.Function

Icon for a function

Information

This icon is designed for a **function**

Modelica.Icons.Record



Icon for a record

Information

This icon is designed for a **record**

Modelica definition

```
partial record Record "Icon for a record"

end Record;
```

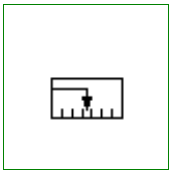
Modelica.Icons.Enumeration

Icon for an enumeration (emulated by a package)

Information

This icon is designed for an **enumeration** (that is emulated by a package).

Modelica.Icons.TranslationalSensor

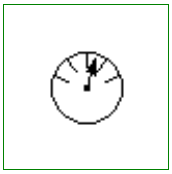


Icon representing translational measurement device

Information

This icon is designed for a **translational sensor** model.

Modelica.Icons.RotationalSensor

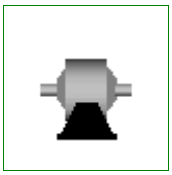


Icon representing rotational measurement device

Information

This icon is designed for a **rotational sensor** model.

Modelica.Icons.GearIcon

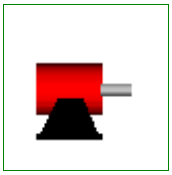


Icon for gearbox

Information

This icon is designed for a **gearbox** model.

Modelica.Icons.MotorIcon

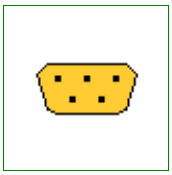


Icon for electrical motor

Information

This icon is designed for an **electrical motor** model.

Modelica.Icons.SignalBus

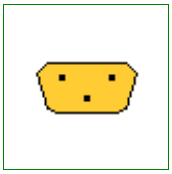


Icon for signal bus

Information

This icon is designed for a **signal bus** connector.

Modelica.Icons.SignalSubBus



Icon for signal sub-bus

Information

This icon is designed for a **sub-bus** in a signal connector.

Modelica.Math

Library of mathematical functions (e.g., sin, cos) and of functions operating on vectors and matrices

Information

This package contains **basic mathematical functions** (such as sin(..)), as well as functions operating on **vectors** and **matrices**.

Main Author:

Martin Otter
Deutsches Zentrum für Luft und Raumfahrt e.V. (DLR)
Institut für Robotik und Mechatronik
Postfach 1116
D-82230 Wessling
Germany
email: Martin.Otter@dlr.de

Copyright © 1998-2007, Modelica Association and DLR.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Vectors	Library of functions operating on vectors
Matrices	Library of functions operating on matrices
sin	Sine
cos	Cosine
tan	Tangent (u shall not be -pi/2, pi/2, 3*pi/2, ...)
asin	Inverse sine (-1 <= u <= 1)
acos	Inverse cosine (-1 <= u <= 1)
atan	Inverse tangent
atan2	Four quadrant inverse tangent
atan3	Four quadrant inverse tangens (select solution that is closest to given angle y0)
sinh	Hyperbolic sine
cosh	Hyperbolic cosine
tanh	Hyperbolic tangent
asinh	Inverse of sinh (area hyperbolic sine)
acosh	Inverse of cosh (area hyperbolic cosine)
exp	Exponential, base e
log	Natural (base e) logarithm (u shall be > 0)
log10	Base 10 logarithm (u shall be > 0)
baseIcon1	Basic icon for mathematical function with y-axis on left side

baseIcon2	Basic icon for mathematical function with y-axis in middle
tempInterpol1	Temporary function for linear interpolation (will be removed)
tempInterpol2	Temporary function for vectorized linear interpolation (will be removed)

Modelica.Math.Vectors

Library of functions operating on vectors

Information

Library content

This library provides functions operating on vectors:

Function	*Description*
isEqual(v1, v2)	Determines whether two vectors have the same size and elements
norm(v,p)	p-norm of vector v
length(v)	Length of vector v (= norm(v,2), but inlined and therefore usable in symbolic manipulations)
normalize(v)	Return normalized vector such that length = 1 and prevent zero-division for zero vector
reverse(v)	Reverse vector elements
sort(v)	Sort elements of vector in ascending or descending order

See also

Matrices

Package Content

Name	Description
isEqual	Determine if two Real vectors are numerically identical
norm	Return the p-norm of a vector
length	Return length of a vector (better as norm(), if further symbolic processing is performed)
normalize	Return normalized vector such that length = 1 and prevent zero-division for zero vector
reverse	Reverse vector elements (e.g. v[1] becomes last element)
sort	Sort elements of vector in ascending or descending order

Modelica.Math.Vectors.isEqual

Determine if two Real vectors are numerically identical

Information

Syntax

```
Vectors.isEqual(v1, v2);
Vectors.isEqual(v1, v2, eps=0);
```

Description

The function call "`Vectors.isEqual(v1, v2)`" returns **true**, if the two Real vectors v1 and v2 have the same dimensions and the same elements. Otherwise the function returns **false**. Two elements e1 and e2 of the two vectors are checked on equality by the test "abs(e1-e2) ≤ eps", where "eps" can be provided as third argument of the function. Default is "eps = 0".

Modelica.Utilities.Strings.isEqual

Example

```
  Real v1[3] = {1, 2, 3};
  Real v2[3] = {1, 2, 3, 4};
  Real v3[3] = {1, 2, 3.0001};
  Boolean result;
algorithm
  result := Vectors.isEqual(v1,v2);      // = false
  result := Vectors.isEqual(v1,v3);      // = false
  result := Vectors.isEqual(v1,v1);      // = true
  result := Vectors.isEqual(v1,v3,0.1); // = true
```

See also

Matrices.isEqual, Strings.isEqual

Inputs

Type	Name	Default	Description
Real	v1[:]		First vector
Real	v2[:]		Second vector (may have different length as v1
Real	eps	0	Two elements e1 and e2 of the two vectors are identical if abs(e1-e2) <= eps

Outputs

Type	Name	Description
Boolean	result	= true, if vectors have the same length and the same elements

---

Modelica.Math.Vectors.norm

Return the p-norm of a vector

Information

Syntax

```
Vectors.norm(v);
Vectors.norm(v,p=2);    // 1 ≤ p ≤ ∞
```

Description

The function call "`Vectors.norm(v)`" returns the **Euclidean norm** "`sqrt(v*v)`" of vector v. With the optional second argument "p", any other p-norm can be computed:

$$\|v\|_p = \left(\sum_{i=1}^{n} |v[i]|^p\right)^{1/p}, \quad 1 \le p \le \infty$$

Besides the Euclidean norm (p=2), also the 1-norm and the infinity-norm are sometimes used:

1-norm	= sum(abs(v))	**norm**(v,1)
2-norm	= sqrt(v*v)	**norm**(v) or **norm**(v,2)
infinity-norm	= max(abs(v))	**norm**(v,Modelica.Constants.**inf**)

Note, for any vector norm the following inequality holds:

```
norm(v1+v2,p) ≤ norm(v1,p) + norm(v2,p)
```

Example

```
v = {2, -4, -2, -1};
norm(v,1);     // = 9
norm(v,2);     // = 5
norm(v);       // = 5
norm(v,10.5); // = 4.00052597412635
norm(v,Modelica.Constants.inf);  // = 4
```

See also

Matrices.norm

Inputs

Type	Name	Default	Description
Real	v[:]		Vector
Real	p	2	Type of p-norm (often used: 1, 2, or Modelica.Constants.inf)

Outputs

Type	Name	Description
Real	result	p-norm of vector v

Modelica.Math.Vectors.length

Return length of a vector (better as norm(), if further symbolic processing is performed)

Information

Syntax

```
Vectors.length(v);
```

Description

The function call "`Vectors.length(v)`" returns the **Euclidean length** "`sqrt(v*v)`" of vector v. The

function call is equivalent to Vectors.norm(v). The advantage of length(v) over norm(v)"is that function length(..) is implemented in one statement and therefore the function is usually automatically inlined. Further symbolic processing is therefore possible, which is not the case with function norm(..).

Example

```
v = {2, -4, -2, -1};
length(v);  // = 5
```

See also

Vectors.norm

Inputs

Type	Name	Default	Description
Real	v[:]		Vector

Outputs

Type	Name	Description
Real	result	Length of vector v

Modelica.Math.Vectors.normalize

Return normalized vector such that length = 1 and prevent zero-division for zero vector

Information

Syntax

```
Vectors.normalize(v);
Vectors.normalize(v,eps=100*Modelica.Constants.eps);
```

Description

The function call "`Vectors.normalize(v)`" returns the **unit vector** "`v/length(v)`" of vector v. If length(v) is close to zero (more precisely, if length(v) < eps), v is returned in order to avoid a division by zero. For many applications this is useful, because often the unit vector **e** = **v**/length(**v**) is used to compute a vector x***e**, where the scalar x is in the order of length(**v**), i.e., x***e** is small, when length(**v**) is small and then it is fine to replace **e** by **v** to avoid a division by zero.

Since the function is implemented in one statement, it is usually inlined and therefore symbolic processing is possible.

Example

```
normalize({1,2,3});  // = {0.267, 0.534, 0.802}
normalize({0,0,0});  // = {0,0,0}
```

See also

Vectors.length

Inputs

Type	Name	Default	Description
Real	v[:]		Vector
Real	eps	100*Modelica.Constants.eps	if |v| < eps then result = v

Outputs

Type	Name	Description
Real	result[size(v, 1)]	Input vector v normalized to length=1

Modelica.Math.Vectors.reverse

Reverse vector elements (e.g. v[1] becomes last element)

Information

Syntax

```
Vectors.reverse(v);
```

Description

The function call "`Vectors.reverse(v)`" returns the vector elements in reverse order.

Example

```
reverse({1,2,3,4});  // = {4,3,2,1}
```

Inputs

Type	Name	Default	Description
Real	v[:]		Vector

Outputs

Type	Name	Description
Real	result[size(v, 1)]	Elements of vector v in reversed order

---

Modelica.Math.Vectors.sort

Sort elements of vector in ascending or descending order

Information

Syntax

```
            sorted_v = Vectors.sort(v);
(sorted_v, indices) = Vectors.sort(v, ascending=true);
```

Description

Function **sort**(..) sorts a Real vector v in ascending order and returns the result in sorted_v. If the optional argument "ascending" is **false**, the vector is sorted in descending order. In the optional second output argument the indices of the sorted vector with respect to the original vector are given, such that sorted_v = v[indices].

Example

```
(v2, i2) := Vectors.sort({-1, 8, 3, 6, 2});
     -> v2 = {-1, 2, 3, 6, 8}
        i2 = {1, 5, 3, 4, 2}
```

Inputs

Type	Name	Default	Description
Real	v[:]		Vector to be sorted
Boolean	ascending	true	= true if ascending order, otherwise descending order

Outputs

Type	Name	Description
Real	sorted_v[size(v, 1)]	Sorted vector
Integer	indices[size(v, 1)]	sorted_v = v[indices]

Modelica.Math.Matrices

Library of functions operating on matrices

Information

Library content

This library provides functions operating on matrices:

Function	*Description*
isEqual(M1, M2)	Determines whether two matrices have the same size and elements
norm(A)	1-, 2- and infinity-norm of matrix A
sort(M)	Sort rows or columns of matrix in ascending or descending order
solve(A,b)	Solve real system of linear equations A*x=b with a b vector
solve2(A,B)	Solve real system of linear equations A*X=B with a B matrix
leastSquares(A,b)	Solve overdetermined or underdetermined real system of linear equations A*x=b in a least squares sense
equalityLeastSquares(A,a,B,b)	Solve a linear equality constrained least squares problem: min|A*x-a|^2 subject to B*x=b
(LU,p,info) = LU(A)	LU decomposition of square or rectangular matrix
LU_solve(LU,p,b)	Solve real system of linear equations P*L*U*x=b with a b vector and an LU decomposition from "LU(..)"
LU_solve2(LU,p,B)	Solve real system of linear equations P*L*U*X=B with a B matrix and an LU decomposition from "LU(..)"
(Q,R,p) = QR(A)	QR decomposition with column pivoting of rectangular matrix (Q*R = A[:,p])

eval = eigenValues(A) (eval,evec) = eigenValues(A)	Compute eigenvalues and optionally eigenvectors for a real, nonsymmetric matrix
eigenValueMatrix(eigen)	Return real valued block diagonal matrix J of eigenvalues of matrix A (A=V*J*Vinv)
sigma = singularValues(A) (sigma,U,VT) = singularValues(A)	Compute singular values and optionally left and right singular vectors
det(A)	Determinant of a matrix (do **not** use; use rank(..))
inv(A)	Inverse of a matrix
rank(A)	Rank of a matrix
balance(A)	Balance a square matrix to improve the condition
exp(A)	Compute the exponential of a matrix by adaptive Taylor series expansion with scaling and balancing
(P, G) = integralExp(A,B)	Compute the exponential of a matrix and its integral
(P, G, GT) = integralExpT(A,B)	Compute the exponential of a matrix and two integrals

Most functions are solely an interface to the external LAPACK library (http://www.netlib.org/lapack). The details of this library are described in:

Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., and Sorensen D.:
Lapack Users' Guide. Third Edition, SIAM, 1999.

See also

Vectors

Package Content

Name	Description
isEqual	Compare whether two Real matrices are identical
norm	Returns the norm of a matrix
sort	Sort rows or columns of matrix in ascending or descending order
solve	Solve real system of linear equations A*x=b with a b vector (Gaussian elemination with partial pivoting)
solve2	Solve real system of linear equations A*X=B with a B matrix (Gaussian elemination with partial pivoting)
leastSquares	Solve overdetermined or underdetermined real system of linear equations A*x=b in a least squares sense (A may be rank deficient)
equalityLeastSquares	Solve a linear equality constrained least squares problem
LU	LU decomposition of square or rectangular matrix
LU_solve	Solve real system of linear equations P*L*U*x=b with a b vector and an LU decomposition (from LU(..))
LU_solve2	Solve real system of linear equations P*L*U*X=B with a B matrix and an LU decomposition (from LU(..))
QR	QR decomposition of a square matrix with column pivoting (A(:,p) = Q*R)
eigenValues	Compute eigenvalues and eigenvectors for a real, nonsymmetric matrix
eigenValueMatrix	Return real valued block diagonal matrix J of eigenvalues of matrix A (A=V*J*Vinv)

singularValues	Compute singular values and left and right singular vectors
det	Determinant of a matrix (computed by LU decomposition)
inv	Inverse of a matrix (try to avoid, use function solve(..) instead)
rank	Rank of a matrix (computed with singular values)
balance	Balancing of matrix A to improve the condition of A
exp	Compute the exponential of a matrix by adaptive Taylor series expansion with scaling and balancing
integralExp	Computation of the transition-matrix phi and its integral gamma
integralExpT	Computation of the transition-matrix phi and the integral gamma and gamma1

Modelica.Math.Matrices.isEqual

Compare whether two Real matrices are identical

Information

Syntax

```
Matrices.isEqual(M1, M2);
Matrices.isEqual(M1, M2, eps=0);
```

Description

The function call "`Matrices.isEqual(M1, M2)`" returns **true**, if the two Real matrices M1 and M2 have the same dimensions and the same elements. Otherwise the function returns **false**. Two elements e1 and e2 of the two matrices are checked on equality by the test "abs(e1-e2) ≤ eps", where "eps" can be provided as third argument of the function. Default is "eps = 0".

Example

```
  Real A1[2,2] = [1,2; 3,4];
  Real A2[3,2] = [1,2; 3,4; 5,6];
  Real A3[2,2] = [1,2, 3,4.0001];
  Boolean result;
algorithm
  result := Matrices.isEqual(M1,M2);      // = false
  result := Matrices.isEqual(M1,M3);      // = false
  result := Matrices.isEqual(M1,M1);      // = true
  result := Matrices.isEqual(M1,M3,0.1); // = true
```

See also

Vectors.isEqual, Strings.isEqual

Inputs

Type	Name	Default	Description
Real	M1[:, :]		First matrix
Real	M2[:, :]		Second matrix (may have different size as M1

Real	eps	0	Two elements e1 and e2 of the two matrices are identical if abs(e1-e2) <= eps

Outputs

Type	Name	Description
Boolean	result	= true, if matrices have the same size and the same elements

---

Modelica.Math.Matrices.norm

Returns the norm of a matrix

Information

Syntax

```
Matrices.norm(A);
Matrices.norm(A, p=2);
```

Description

The function call "`Matrices.norm(A)`" returns the 2-norm of matrix A, i.e., the largest singular value of A. The function call "`Matrices.norm(A, p)`" returns the p-norm of matrix A. The only allowed values for p are

- "p=1": the largest column sum of A
- "p=2": the largest singular value of A
- "p=Modelica.Constants.inf": the largest row sum of A

Note, for any matrices A1, A2 the following inequality holds:

```
Matrices.norm(A1+A2,p) ≤ Matrices.norm(A1,p) + Matrices.norm(A2,p)
```

Note, for any matrix A and vector v the following inequality holds:

```
Vectors.norm(A*v,p) ≤ Matrices.norm(A,p)*Vectors.norm(A,p)
```

Inputs

Type	Name	Default	Description
Real	A[:, :]		Input matrix
Real	p	2	Type of p-norm (only allowed: 1, 2 or Modelica.Constants.inf)

Outputs

Type	Name	Description
Real	result	p-norm of matrix A

Modelica.Math.Matrices.sort

Sort rows or columns of matrix in ascending or descending order

Information

Syntax

```
              sorted_M = Matrices.sort(M);
(sorted_M, indices) = Matrices.sort(M, sortRows=true, ascending=true);
```

Description

Function **sort**(..) sorts the rows of a Real matrix M in ascending order and returns the result in sorted_M. If the optional argument "sortRows" is **false**, the columns of the matrix are sorted. If the optional argument "ascending" is **false**, the rows or columns are sorted in descending order. In the optional second output argument, the indices of the sorted rows or columns with respect to the original matrix are given, such that

```
sorted_M = if sortedRow then M[indices,:] else M[:,indices];
```

Example

```
(M2, i2) := Matrices.sort([2, 1,  0;
                           2, 0, -1]);
     -> M2 = [2, 0, -1;
              2, 1, 0 ];
        i2 = {2,1};
```

Inputs

Type	Name	Default	Description
Real	M[:, :]		Matrix to be sorted
Boolean	sortRows	true	= true if rows are sorted, otherwise columns
Boolean	ascending	true	= true if ascending order, otherwise descending order

Outputs

Type	Name	Description
Real	sorted_M[size(M, 1), size(M, 2)]	Sorted matrix
Integer	indices[if sortRows then size(M, 1) else size(M, 2)]	sorted_M = if sortRows then M[indices,:] else M[:,indices]

Modelica.Math.Matrices.solve

Solve real system of linear equations A*x=b with a b vector (Gaussian elemination with partial pivoting)

Information

Syntax

```
Matrices.solve(A,b);
```

Description

This function call returns the solution **x** of the linear system of equations

A*x = b

If a unique solution **x** does not exist (since **A** is singular), an exception is raised.

Note, the solution is computed with the LAPACK function "dgesv", i.e., by Gaussian elemination with partial pivoting.

Example

```
  Real A[3,3] = [1,2,3;
                 3,4,5;
                 2,1,4];
  Real b[3] = {10,22,12};
  Real x[3];
algorithm
  x := Matrices.solve(A,b);  // x = {3,2,1}
```

See also

Matrices.LU, Matrices.LU_solve

Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		Matrix A of A*x = b
Real	b[size(A, 1)]		Vector b of A*x = b

Outputs

Type	Name	Description
Real	x[size(b, 1)]	Vector x such that A*x = b

Modelica.Math.Matrices.solve2

Solve real system of linear equations A*X=B with a B matrix (Gaussian elemination with partial pivoting)

Information

Syntax

```
Matrices.solve2(A,b);
```

Description

This function call returns the solution **X** of the linear system of equations

A*X = B

If a unique solution **X** does not exist (since **A** is singular), an exception is raised.

Note, the solution is computed with the LAPACK function "dgesv", i.e., by Gaussian elemination with partial pivoting.

Example

```
  Real A[3,3] = [1,2,3;
                 3,4,5;
                 2,1,4];
  Real B[3,2] = [10, 20;
                 22, 44;
                 12, 24];
  Real X[3,2];
algorithm
  (LU, pivots) := Matrices.LU(A);
  X := Matrices.solve2(A, B1);  /* X = [3, 6;
                                        2, 4;
                                        1, 2] */
```

See also

Matrices.LU, Matrices.LU_solve2

Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		Matrix A of A*X = B
Real	B[size(A, 1), :]		Matrix B of A*X = B

Outputs

Type	Name	Description
Real	X[size(B, 1), size(B, 2)]	Matrix X such that A*X = B

Modelica.Math.Matrices.leastSquares

Solve overdetermined or underdetermined real system of linear equations A*x=b in a least squares sense (A may be rank deficient)

Information

Syntax

```
x = Matrices.leastSquares(A,b);
```

Description

A linear system of equations A*x = b has no solutions or infinitely many solutions if A is not square. Function "leastSquares" returns a solution in a least squarse sense:

```
size(A,1) > size(A,2):  returns x such that |A*x - b|^2 is a minimum
size(A,1) = size(A,2):  returns x such that A*x = b
size(A,1) < size(A,2):  returns x such that |x|^2 is a minimum for all
                        vectors x that fulfill A*x = b
```

Note, the solution is computed with the LAPACK function "dgelsx", i.e., QR or LQ factorization of A with column pivoting. If A does not have full rank, the solution is not unique and from the infinitely many solutions

the one is selected that minimizes both |x|^2 and |A*x - b|^2.

Inputs

Type	Name	Default	Description
Real	A[:, :]		Matrix A
Real	b[size(A, 1)]		Vector b

Outputs

Type	Name	Description
Real	x[size(A, 2)]	Vector x such that min|A*x-b|^2 if size(A,1) >= size(A,2) or min|x|^2 and A*x=b, if size(A,1) < size(A,2)

Modelica.Math.Matrices.equalityLeastSquares

Solve a linear equality constrained least squares problem

Information

Syntax

```
x = Matrices.equalityLeastSquares(A,a,B,b);
```

Description

This function returns the solution **x** of the linear equality-constrained least squares problem:

min|**A*****x** - **a**|^2 over **x**, subject to **B*****x** = **b**

It is required that the dimensions of A and B fulfill the following relationship:

size(B,1) ≤ size(A,2) ≤ size(A,1) + size(B,1)

Note, the solution is computed with the LAPACK function "dgglse" using the generalized RQ factorization under the assumptions that B has full row rank (= size(B,1)) and the matrix [A;B] has full column rank (= size(A,2)). In this case, the problem has a unique solution.

Inputs

Type	Name	Default	Description
Real	A[:, :]		Minimize |A*x - a|^2
Real	a[size(A, 1)]		
Real	B[:, size(A, 2)]		subject to B*x=b
Real	b[size(B, 1)]		

Outputs

Type	Name	Description
Real	x[size(A, 2)]	solution vector

Modelica.Math.Matrices.LU

LU decomposition of square or rectangular matrix

Information

Syntax

```
(LU, pivots)       = Matrices.LU(A);
(LU, pivots, info) = Matrices.LU(A);
```

Description

This function call returns the LU decomposition of a "Real[m,n]" matrix A, i.e.,

P*L*U = A

where **P** is a permutation matrix (implicitely defined by vector `pivots`), **L** is a lower triangular matrix with unit diagonal elements (lower trapezoidal if m > n), and **U** is an upper triangular matrix (upper trapezoidal if m < n). Matrices **L** and **U** are stored in the returned matrix `LU` (the diagonal of **L** is not stored). With the companion function Matrices.LU_solve, this decomposition can be used to solve linear systems (**P*L*U**)***x** = **b** with different right hand side vectors **b**. If a linear system of equations with just one right hand side vector **b** shall be solved, it is more convenient to just use the function Matrices.solve.

The optional third (Integer) output argument has the following meaning:

info = 0:successful exit

info > 0: if info = i, U[i,i] is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

The LU factorization is computed with the LAPACK function "dgetrf", i.e., by Gaussian elemination using partial pivoting with row interchanges. Vector "pivots" are the pivot indices, i.e., for 1 ≤ i ≤ min(m,n), row i of matrix A was interchanged with row pivots[i].

Example

```
  Real A[3,3] = [1,2,3;
                 3,4,5;
                 2,1,4];
  Real b1[3] = {10,22,12};
  Real b2[3] = { 7,13,10};
  Real    LU[3,3];
  Integer pivots[3];
  Real    x1[3];
  Real    x2[3];
algorithm
  (LU, pivots) := Matrices.LU(A);
  x1 := Matrices.LU_solve(LU, pivots, b1);  // x1 = {3,2,1}
  x2 := Matrices.LU_solve(LU, pivots, b2);  // x2 = {1,0,2}
```

See also

Matrices.LU_solve, Matrices.solve,

Inputs

Type	Name	Default	Description

Real	A[:, :]		Square or rectangular matrix

Outputs

Type	Name	Description
Real	LU[size(A, 1), size(A, 2)]	L,U factors (used with LU_solve(..))
Integer	pivots[min(size(A, 1), size(A, 2))]	pivot indices (used with LU_solve(..))
Integer	info	Information

Modelica.Math.Matrices.LU_solve

Solve real system of linear equations P*L*U*x=b with a b vector and an LU decomposition (from LU(..))

Information

Syntax

```
Matrices.LU_solve(LU, pivots, b);
```

Description

This function call returns the solution **x** of the linear systems of equations

P*L*U*x = b;

where **P** is a permutation matrix (implicitely defined by vector `pivots`), **L** is a lower triangular matrix with unit diagonal elements (lower trapezoidal if m > n), and **U** is an upper triangular matrix (upper trapezoidal if m < n). The matrices of this decomposition are computed with function Matrices.LU that returns arguments `LU` and `pivots` used as input arguments of `Matrices.LU_solve`. With `Matrices.LU` and `Matrices.LU_solve` it is possible to efficiently solve linear systems with different right hand side vectors. If a linear system of equations with just one right hand side vector shall be solved, it is more convenient to just use the function Matrices.solve.

If a unique solution **x** does not exist (since the LU decomposition is singular), an exception is raised.

The LU factorization is computed with the LAPACK function "dgetrf", i.e., by Gaussian elemination using partial pivoting with row interchanges. Vector "pivots" are the pivot indices, i.e., for 1 ≤ i ≤ min(m,n), row i of matrix A was interchanged with row pivots[i].

Example

```
  Real A[3,3] = [1,2,3;
                 3,4,5;
                 2,1,4];
  Real b1[3] = {10,22,12};
  Real b2[3] = { 7,13,10};
  Real    LU[3,3];
  Integer pivots[3];
  Real    x1[3];
  Real    x2[3];
algorithm
  (LU, pivots) := Matrices.LU(A);
  x1 := Matrices.LU_solve(LU, pivots, b1);  // x1 = {3,2,1}
  x2 := Matrices.LU_solve(LU, pivots, b2);  // x2 = {1,0,2}
```

See also

Matrices.LU, Matrices.solve,

Inputs

Type	Name	Default	Description
Real	LU[:, size(LU, 1)]		L,U factors of Matrices.LU(..) for a square matrix
Integer	pivots[size(LU, 1)]		Pivots indices of Matrices.LU(..)
Real	b[size(LU, 1)]		Right hand side vector of P*L*U*x=b

Outputs

Type	Name	Description
Real	x[size(b, 1)]	Solution vector such that P*L*U*x = b

Modelica.Math.Matrices.LU_solve2

Solve real system of linear equations P*L*U*X=B with a B matrix and an LU decomposition (from LU(..))

Information

Syntax

```
Matrices.LU_solve(LU, pivots, B);
```

Description

This function call returns the solution **X** of the linear systems of equations

P*L*U*X = B;

where **P** is a permutation matrix (implicitely defined by vector `pivots`), **L** is a lower triangular matrix with unit diagonal elements (lower trapezoidal if m > n), and **U** is an upper triangular matrix (upper trapezoidal if m < n). The matrices of this decomposition are computed with function Matrices.LU that returns arguments `LU` and `pivots` used as input arguments of `Matrices.LU_solve2`. With `Matrices.LU` and `Matrices.LU_solve2` it is possible to efficiently solve linear systems with different right hand side **matrices**. If a linear system of equations with just one right hand side matrix shall be solved, it is more convenient to just use the function Matrices.solve2.

If a unique solution **X** does not exist (since the LU decomposition is singular), an exception is raised.

The LU factorization is computed with the LAPACK function "dgetrf", i.e., by Gaussian elemination using partial pivoting with row interchanges. Vector "pivots" are the pivot indices, i.e., for 1 ≤ i ≤ min(m,n), row i of matrix A was interchanged with row pivots[i].

Example

```
Real A[3,3] = [1,2,3;
               3,4,5;
               2,1,4];
Real B1[3] = [10, 20;
              22, 44;
              12, 24];
```

```
  Real B2[3] = [ 7, 14;
                13, 26;
                10, 20];
  Real    LU[3,3];
  Integer pivots[3];
  Real    X1[3,2];
  Real    X2[3,2];
algorithm
  (LU, pivots) := Matrices.LU(A);
  X1 := Matrices.LU_solve2(LU, pivots, B1);  /* X1 = [3, 6;
                                                   2, 4;
                                                   1, 2] */
  X2 := Matrices.LU_solve2(LU, pivots, B2);  /* X2 = [1, 2;
                                                   0, 0;
                                                   2, 4] */
```

See also

Matrices.LU, Matrices.solve2,

Inputs

Type	Name	Default	Description
Real	LU[:, size(LU, 1)]		L,U factors of Matrices.LU(..) for a square matrix
Integer	pivots[size(LU, 1)]		Pivots indices of Matrices.LU(..)
Real	B[size(LU, 1), :]		Right hand side matrix of P*L*U*X=B

Outputs

Type	Name	Description
Real	X[size(B, 1), size(B, 2)]	Solution matrix such that P*L*U*X = B

Modelica.Math.Matrices.QR

QR decomposition of a square matrix with column pivoting (A(:,p) = Q*R)

Information

Syntax

```
(Q,R,p) = Matrices.QR(A);
```

Description

This function returns the QR decomposition of a rectangular matrix **A** (the number of columns of **A** must be less than or equal to the number of rows):

Q***R** = **A**[:,**p**]

where **Q** is a rectangular matrix that has orthonormal columns and has the same size as A ($\mathbf{Q}^T\mathbf{Q}=\mathbf{I}$), **R** is a square, upper triangular matrix and **p** is a permutation vector. Matrix **R** has the following important properties:

- The absolute value of a diagonal element of **R** is the largest value in this row, i.e., abs(R[i,i]) ≥ abs(R[i,j]).
- The diagonal elements of **R** are sorted according to size, such that the largest absolute value is abs(R[1,1]) and abs(R[i,i]) ≥ abs(R[j,j]) with i < j.

This means that if abs(R[i,i]) ≤ ε then abs(R[j,k]) ≤ ε for j ≥ i, i.e., the i-th row up to the last row of **R** have small elements and can be treated as being zero. This allows to, e.g., estimate the row-rank of **R** (which is the same row-rank as **A**). Furthermore, **R** can be partitioned in two parts

$$\mathbf{A}[:,p] = \mathbf{Q} * \begin{bmatrix} \mathbf{R}_1, & \mathbf{R}_2; \\ \mathbf{0}, & \mathbf{0} \end{bmatrix}$$

where $\mathbf{R}_1$ is a regular, upper triangular matrix.

Note, the solution is computed with the LAPACK functions "dgeqpf" and "dorgqr", i.e., by Housholder transformations with column pivoting. If **Q** is not needed, the function may be called as: `(,R,p) = QR(A)`.

Example

```
  Real A[3,3] = [1,2,3;
                 3,4,5;
                 2,1,4];
  Real R[3,3];
algorithm
  (,R) := Matrices.QR(A);  // R = [-7.07.., -4.24.., -3.67..;
                                    0     , -1.73.., -0.23..;
                                    0     ,  0     ,  0.65..];
```

Inputs

Type	Name	Default	Description
Real	A[:, :]		Rectangular matrix with size(A,1) >= size(A,2)

Outputs

Type	Name	Description
Real	Q[size(A, 1), size(A, 2)]	Rectangular matrix with orthonormal columns such that Q*R=A[:,p]
Real	R[size(A, 2), size(A, 2)]	Square upper triangular matrix
Integer	p[size(A, 2)]	Column permutation vector

Modelica.Math.Matrices.eigenValues

Compute eigenvalues and eigenvectors for a real, nonsymmetric matrix

Information

Syntax

```
                    eigenvalues = Matrices.eigenValues(A);
(eigenvalues, eigenvectors) = Matrices.eigenValues(A);
```

Description

This function call returns the eigenvalues and optionally the (right) eigenvectors of a square matrix **A**. The

first column of "eigenvalues" contains the real and the second column contains the imaginary part of the eigenvalues. If the i-th eigenvalue has no imaginary part, then eigenvectors[:,i] is the corresponding real eigenvector. If the i-th eigenvalue has an imaginary part, then eigenvalues[i+1,:] is the conjugate complex eigenvalue and eigenvectors[:,i] is the real and eigenvectors[:,i+1] is the imaginary part of the eigenvector of the i-th eigenvalue. With function Matrices.eigenValueMatrix, a real block diagonal matrix is constructed from the eigenvalues such that

```
A = eigenvectors * eigenValueMatrix(eigenvalues) * inv(eigenvectors)
```

provided the eigenvector matrix "eigenvectors" can be inverted (an inversion is possible, if all eigenvalues are different and no eigenvalue is zero).

Example

```
  Real A[3,3] = [1,2,3;
                 3,4,5;
                 2,1,4];
  Real eval;
algorithm
  eval := Matrices.eigenValues(A);  // eval = [-0.618, 0;
                                    //           8.0  , 0;
                                    //           1.618, 0];
```

i.e., matrix A has the 3 real eigenvalues -0.618, 8, 1.618.

See also

Matrices.eigenValueMatrix, Matrices.singularValues

Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		Matrix

Outputs

Type	Name	Description
Real	eigenvalues[size(A, 1), 2]	Eigenvalues of matrix A (Re: first column, Im: second column)
Real	eigenvectors[size(A, 1), size(A, 2)]	Real-valued eigenvector matrix

---

Modelica.Math.Matrices.eigenValueMatrix

Return real valued block diagonal matrix J of eigenvalues of matrix A (A=V*J*Vinv)

Information

Syntax

```
Matrices.eigenValueMatrix(eigenvalues);
```

Description

The function call returns a block diagonal matrix **J** from the the two-column matrix `eigenvalues` (computed by function Matrices.eigenValues). Matrix `eigenvalues` must have the real part of the eigenvalues in the

first column and the imaginary part in the second column. If an eigenvalue i has a vanishing imaginary part, then **J**[i,i] = eigenvalues[i,1], i.e., the diagonal element of **J** is the real eigenvalue. Otherwise, eigenvalue i and conjugate complex eigenvalue i+1 are used to construct a 2 by 2 diagonal block of **J**:

```
J[i  , i]   := eigenvalues[i,1];
J[i  , i+1] := eigenvalues[i,2];
J[i+1, i]   := eigenvalues[i+1,2];
J[i+1, i+1] := eigenvalues[i+1,1];
```

See also

Matrices.eigenValues

Inputs

Type	Name	Default	Description
Real	eigenValues[:, 2]		Eigen values from function eigenValues(..) (Re: first column, Im: second column)

Outputs

Type	Name	Description
Real	J[size(eigenValues, 1), size(eigenValues, 1)]	Real valued block diagonal matrix with eigen values (Re: 1x1 block, Im: 2x2 block)

Modelica.Math.Matrices.singularValues

Compute singular values and left and right singular vectors

Information

Syntax

```
          sigma = Matrices.singularValues(A);
(sigma, U, VT) = Matrices.singularValues(A);
```

Description

This function computes the singular values and optionally the singular vectors of matrix A. Basically the singular value decomposition of A is computed, i.e.,

$$\mathbf{A} = \mathbf{U}\ \mathbf{\Sigma}\ \mathbf{V}^T$$
```
  = U*Sigma*VT
```

where **U** and **V** are orthogonal matrices ($\mathbf{UU}^T=\mathbf{I}$, $\mathbf{VV}^T=\mathbf{I}$). $\mathbf{\Sigma}$ = diag(σ_i) has the same size as matrix A with nonnegative diagonal elements in decreasing order and with all other elements zero (σ_1 is the largest element). The function returns the singular values σ_i in vector `sigma` and the orthogonal matrices in matrices `U` and `V`.

Example

```
A = [1, 2,  3,  4;
     3, 4,  5, -2;
```

```
    -1, 2, -3,  5];
(sigma, U, VT) = singularValues(A);
results in:
   sigma = {8.33, 6.94, 2.31};
i.e.
   Sigma = [8.33,    0,    0, 0;
               0, 6.94,    0, 0;
               0,    0, 2.31, 0]
```

See also

Matrices.eigenValues

Inputs

Type	Name	Default	Description
Real	A[:, :]		Matrix

Outputs

Type	Name	Description
Real	sigma[min(size(A, 1), size(A, 2))]	Singular values
Real	U[size(A, 1), size(A, 1)]	Left orthogonal matrix
Real	VT[size(A, 2), size(A, 2)]	Transposed right orthogonal matrix

---

Modelica.Math.Matrices.det

Determinant of a matrix (computed by LU decomposition)

Information

Syntax

```
Matrices.det(A);
```

Description

This function call returns the determinant of matrix A computed by a LU decomposition. Usally, this function should never be used, because there are nearly always better numerical algorithms as by computing the determinant. E.g., use function Matrices.rank to compute the rank of a matrix.

See also

Matrices.rank, Matrices.solve

Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		

Outputs

Type	Name	Description
Real	result	Determinant of matrix A

Modelica.Math.Matrices.inv

Inverse of a matrix (try to avoid, use function solve(..) instead)

Information

Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		

Outputs

Type	Name	Description
Real	invA[size(A, 1), size(A, 2)]	Inverse of matrix A

Modelica.Math.Matrices.rank

Rank of a matrix (computed with singular values)

Information

Inputs

Type	Name	Default	Description
Real	A[:, :]		Matrix
Real	eps	0	If eps > 0, the singular values are checked against eps; otherwise eps=max(size(A))*norm(A)*Modelica.Constants.eps is used

Outputs

Type	Name	Description
Integer	result	Rank of matrix A

Modelica.Math.Matrices.balance

Balancing of matrix A to improve the condition of A

Information

The function transformates the matrix A, so that the norm of the i-th column is nearby the i-th row. (D,B)=Matrices.balance(A) returns a vector D, such that B=inv(diagonal(D))*A*diagonal(D) has better condition. The elements of D are multiples of 2. Balancing attempts to make the norm of each row equal to the norm of the belonging column.
Balancing is used to minimize roundoff errors inducted through large matrix calculations like Taylor-series approximation or computation of eigenvalues.

Example:

```
- A = [1, 10,  1000; .01,  0,  10; .005,  .01,  10]
- Matrices.norm(A, 1);
  = 1020.0
- (T,B)=Matrices.balance(A)
- T
  = {256, 16, 0.5}
- B
  =  [1,     0.625,   1.953125;
      0.16,  0,       0.3125;
      2.56,  0.32,   10.0]
- Matrices.norm(B, 1);
  = 12.265625
```

The Algorithm is taken from

H. D. Joos, G. Grbel:
RASP'91 Regulator Analysis and Synthesis Programs
DLR - Control Systems Group 1991

which based on the balanc function from EISPACK.

Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		

Outputs

Type	Name	Description
Real	D[size(A, 1)]	diagonal(D)=T is transformation matrix, such that T*A*inv(T) has smaller condition as A
Real	B[size(A, 1), size(A, 1)]	Balanced matrix (= diagonal(D)*A*inv(diagonal(D)))

Modelica.Math.Matrices.exp

Compute the exponential of a matrix by adaptive Taylor series expansion with scaling and balancing

Information

This function computes

$$\Phi = e^{(\mathbf{A}T)} = \mathbf{I} + \mathbf{A}T + \frac{(\mathbf{A}T)^2}{2!} + \frac{(\mathbf{A}T)^3}{3!} + \ldots$$

where e=2.71828..., **A** is an n x n matrix with real elements and T is a real number, e.g., the sampling time. **A** may be singular. With the exponential of a matrix it is, e.g., possible to compute the solution of a linear system of differential equations

$$\mathrm{der}(\mathbf{x}) = \mathbf{A}*\mathbf{x} \quad \rightarrow \quad \mathbf{x}(t0 + T) = e^{(\mathbf{A}T)}*x(t0)$$

The function is called as

```
Phi = Matrices.exp(A,T);
```

or

```
M = Matrices.exp(A);
```

what calculates M as the exponential of matrix A.

Algorithmic details:

The algorithm is taken from

H. D. Joos, G. Gruebel:
RASP'91 Regulator Analysis and Synthesis Programs
DLR - Control Systems Group 1991

The following steps are performed to calculate the exponential of A:

1. Matrix **A** is balanced
   (= is transformed with a diagonal matrix **D**, such that inv(**D**)*__A__*__D__ has a smaller condition as **A**).
2. The scalar T is divided by a multiple of 2 such that norm(inv(**D**)*__A__*__D__*T/2^k) < 0.5. Note, that (1) and (2) are implemented such that no round-off errors are introduced.
3. The matrix from (2) is approximated by explicitly performing the Taylor series expansion with a variable number of terms. Truncation occurs if a new term does no longer contribute to the value of **Φ** from the previous iteration.
4. The resulting matrix is transformed back, by reverting the steps of (2) and (1).

In several sources it is not recommended to use Taylor series expansion to calculate the exponential of a matrix, such as in 'C.B. Moler and C.F. Van Loan: Nineteen dubious ways to compute the exponential of a matrix. SIAM Review 20, pp. 801-836, 1979' or in the documentation of m-file expm2 in Matlab version 6 (http://www.MathWorks.com) where it is stated that 'As a practical numerical method, this is often slow and inaccurate'. These statements are valid for a direct implementation of the Taylor series expansion, but *not* for the implementation variant used in this function.

Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		
Real	T	1	

Outputs

Type	Name	Description
Real	phi[size(A, 1), size(A, 1)]	= exp(A*T)

Modelica.Math.Matrices.integralExp

Computation of the transition-matrix phi and its integral gamma

Information

The function uses a Taylor series expansion with Balancing and scaling/squaring to approximate the integral **Ψ** of the matrix exponential **Φ**=e^(AT):

$$\Psi = \int e^{As}\,ds = IT + \frac{AT^2}{2!} + \frac{A^2 T^3}{3!} + \ldots + \frac{A^k T^{(k+1)}}{(k+1)!}$$

Φ is calculated through **Φ** = I + A***Ψ**, so A may be singular. **Γ** is simple **Ψ***B.

The algorithm runs in the following steps:

1. Balancing
2. Scaling
3. Taylor series expansion
4. Re-scaling
5. Re-Balancing

Balancing put the bad condition of a square matrix *A* into a diagonal transformation matrix *D*. This reduce the effort of following calculations. Afterwards the result have to be re-balanced by transformation D*Atransf *inv(D).
Scaling halfen T k-times, until the norm of A*T is less than 0.5. This garantees minumum rounding errors in the following series expansion. The re-scaling based on the equation exp(A*2T) = exp(AT)^2. The needed re-scaling formula for psi thus becomes:

```
        Φ = Φ'*Φ'
I + A*Ψ = I + 2A*Ψ' + A^2*Ψ'^2
        Ψ = A*Ψ'^2 + 2*Ψ'
```

where psi' is the scaled result from the series expansion while psi is the re-scaled matrix.

The function is normally used to discretize a state-space system as the zero-order-hold equivalent:

```
x(k+1) = Φ*x(k) + Γ*u(k)
  y(k) = C*x(k) + D*u(k)
```

The zero-order-hold sampling, also known as step-invariant method, gives exact values of the state variables, under the assumption that the control signal u is constant between the sampling instants. Zero-order-hold sampling is discribed in

K. J. Astroem, B. Wittenmark:
Computer Controlled Systems - Theory and Design
Third Edition, p. 32

Syntax:

```
(phi,gamma) = Matrices.expIntegral(A,B,T)
                  A,phi: [n,n] square matrices
                B,gamma: [n,m] input matrix
                      T: scalar, e.g. sampling time
```

The Algorithm to calculate psi is taken from

H. D. Joos, G. Gruebel:
RASP'91 Regulator Analysis and Synthesis Programs
DLR - Control Systems Group 1991

Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		
Real	B[size(A, 1), :]		
Real	T	1	

Outputs

Type	Name	Description
Real	phi[size(A, 1), size(A, 1)]	= exp(A*T)

Real	gamma[size(A, 1), size(B, 2)]	= integral(phi)*B

Modelica.Math.Matrices.integralExpT

Computation of the transition-matrix phi and the integral gamma and gamma1

Information

The function calculates the matrices phi,gamma,gamma1 through the equation:

```
                                    [ A B 0 ]
[phi gamma gamma1] = [I 0 0]*exp([ 0 0 I ]*T)
                                    [ 0 0 0 ]
```

Syntax:

```
(phi,gamma,gamma1) = Matrices.ExpIntegral2(A,B,T)
                A,phi: [n,n] square matrices
      B,gamma,gamma1: [n,m] matrices
                   T: scalar, e.g. sampling time
```

The matrices define the discretized first-order-hold equivalent of a state-space system:

```
x(k+1) = phi*x(k) + gamma*u(k) + gamma1/T*(u(k+1) - u(k))
```

The first-order-hold sampling, also known as ramp-invariant method, gives more smooth control signals as the ZOH equivalent. First-order-hold sampling is discribed in

K. J. Astroem, B. Wittenmark:
Computer Controlled Systems - Theory and Design
Third Edition, p. 256

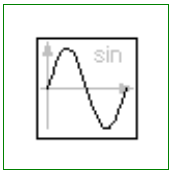
Inputs

Type	Name	Default	Description
Real	A[:, size(A, 1)]		
Real	B[size(A, 1), :]		
Real	T	1	

Outputs

Type	Name	Description
Real	phi[size(A, 1), size(A, 1)]	= exp(A*T)
Real	gamma[size(A, 1), size(B, 2)]	= integral(phi)*B
Real	gamma1[size(A, 1), size(B, 2)]	= integral((T-t)*exp(A*t))*B

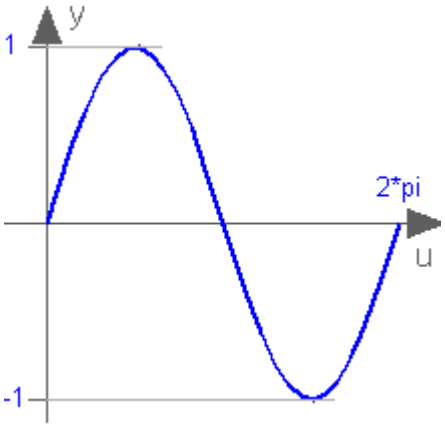
Modelica.Math.sin



Sine

Information

This function returns y = sin(u), with -∞ < u < ∞:



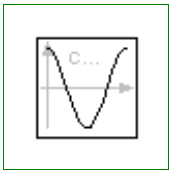
Inputs

Type	Name	Default	Description
Angle	u		[rad]

Outputs

Type	Name	Description
Real	y	

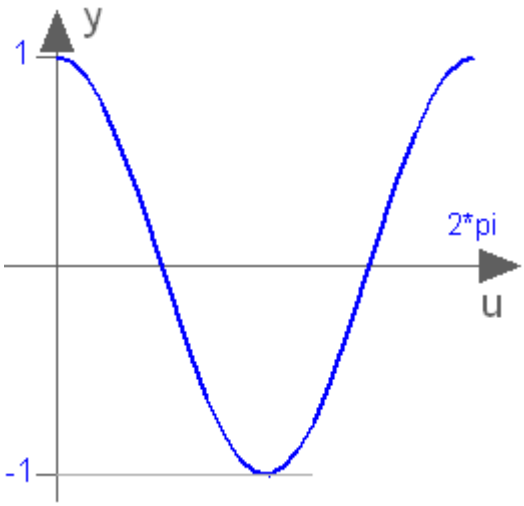
Modelica.Math.cos



Cosine

Information

This function returns y = cos(u), with -∞ < u < ∞:



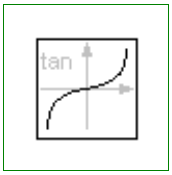
Inputs

Type	Name	Default	Description
Angle	u		[rad]

Outputs

Type	Name	Description
Real	y	

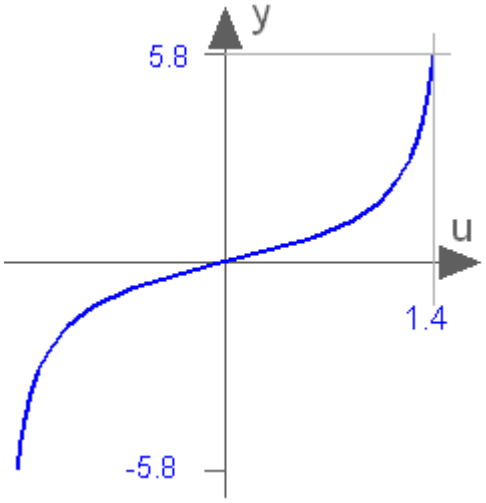
Modelica.Math.tan



Tangent (u shall not be -pi/2, pi/2, 3*pi/2, ...)

Information

This function returns y = tan(u), with -∞ < u < ∞ (if u is a multiple of (2n-1)*pi/2, y = tan(u) is +/- infinity).



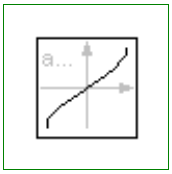
Inputs

Type	Name	Default	Description
Angle	u		[rad]

Outputs

Type	Name	Description
Real	y	

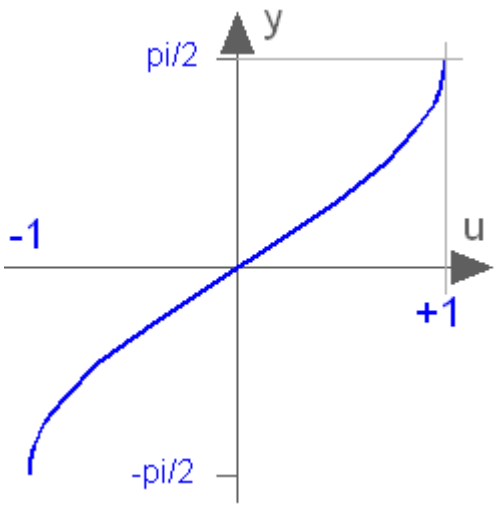
Modelica.Math.asin



Inverse sine (-1 <= u <= 1)

Information

This function returns y = asin(u), with -1 ≤ u ≤ +1:



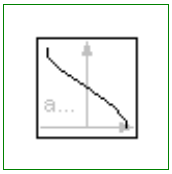
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Angle	y	[rad]

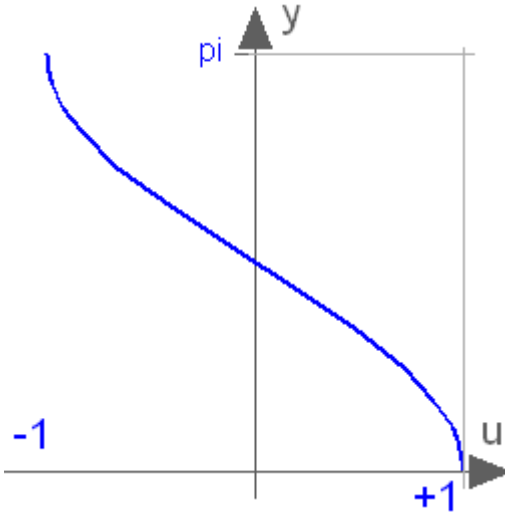
Modelica.Math.acos



Inverse cosine (-1 <= u <= 1)

Information

This function returns y = acos(u), with -1 ≤ u ≤ +1:



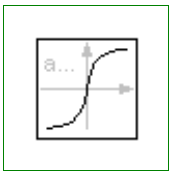
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Angle	y	[rad]

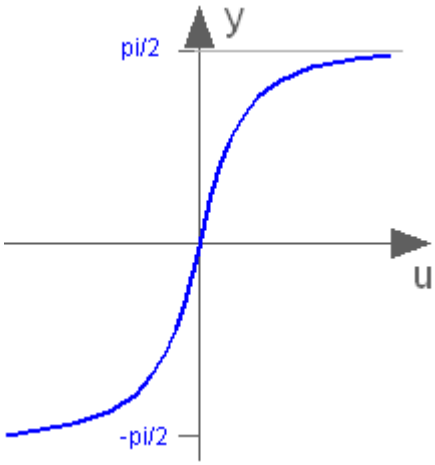
Modelica.Math.atan



Inverse tangent

Information

This function returns y = atan(u), with -∞ < u < ∞:



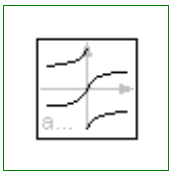
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Angle	y	[rad]

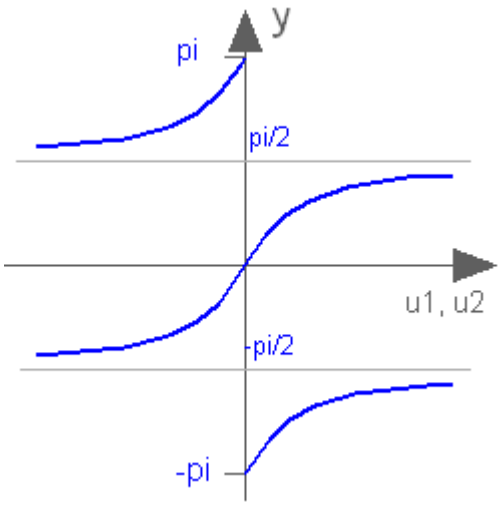
Modelica.Math.atan2



Four quadrant inverse tangent

Information

This function returns y = atan2(u1,u2) such that tan(y) = u1/u2 and y is in the range -pi < y ≤ pi. u2 may be zero, provided u1 is not zero. Usually u1, u2 is provided in such a form that u1 = sin(y) and u2 = cos(y):



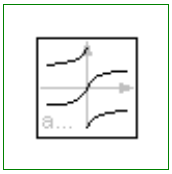
Inputs

Type	Name	Default	Description
Real	u1		
Real	u2		

Outputs

Type	Name	Description
Angle	y	[rad]

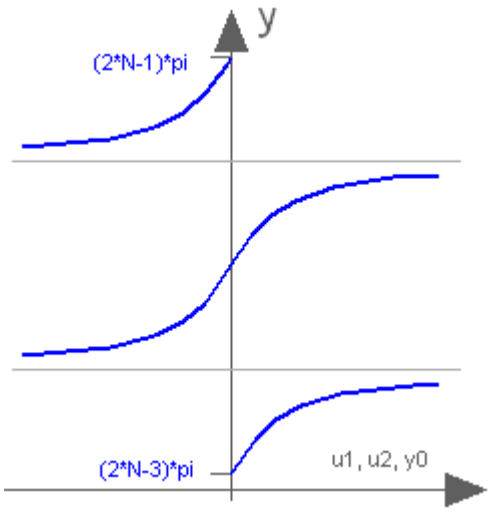
Modelica.Math.atan3



Four quadrant inverse tangens (select solution that is closest to given angle y0)

Information

This function returns y = **atan3**(u1,u2,y0) such that **tan**(y) = u1/u2 and y is in the range: -pi < y-y0 < pi. u2 may be zero, provided u1 is not zero. The difference to Modelica.Math.atan2(..) is the optional third argument y0 that allows to specify which of the infinite many solutions shall be returned:



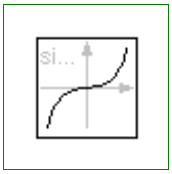
Inputs

Type	Name	Default	Description
Real	u1		
Real	u2		
Angle	y0	0	y shall be in the range: -pi < y-y0 < pi [rad]

Outputs

Type	Name	Description
Angle	y	[rad]

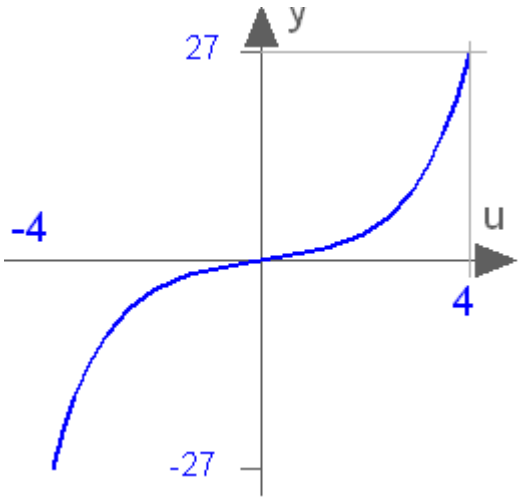
Modelica.Math.sinh



Hyperbolic sine

Information

This function returns y = sinh(u), with -∞ < u < ∞:



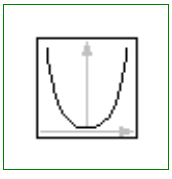
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Real	y	

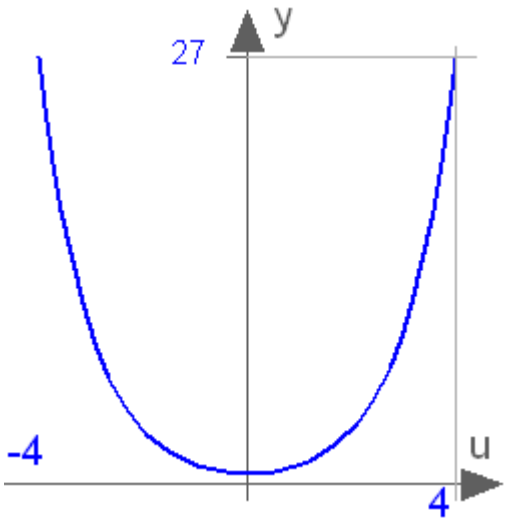
Modelica.Math.cosh



Hyperbolic cosine

Information

This function returns y = cosh(u), with -∞ < u < ∞:



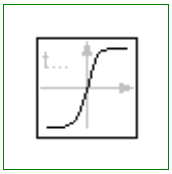
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Real	y	

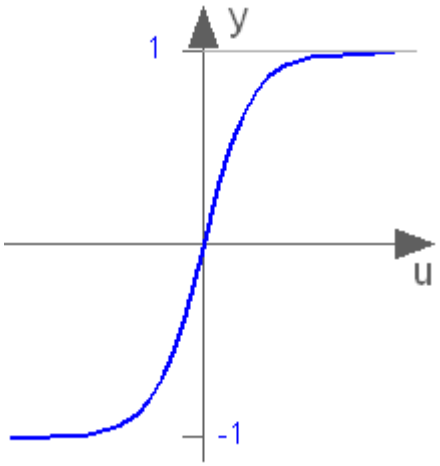
Modelica.Math.tanh



Hyperbolic tangent

Information

This function returns y = tanh(u), with -∞ < u < ∞:



Inputs

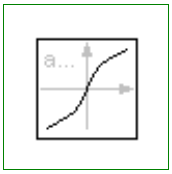
Type	Name	Default	Description

Real	u		

Outputs

Type	Name	Description
Real	y	

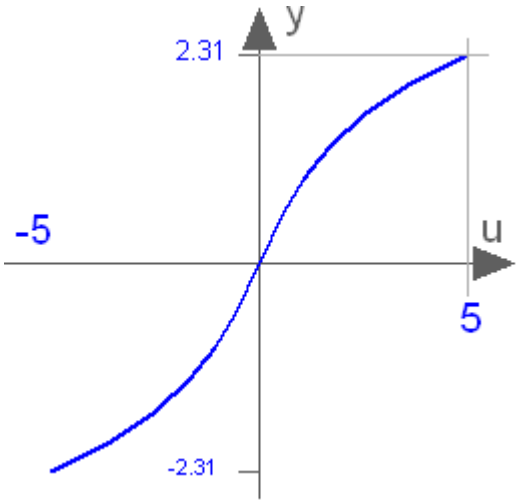
Modelica.Math.asinh



Inverse of sinh (area hyperbolic sine)

Information

The function returns the area hyperbolic sine of its input argument u. This inverse of sinh(..) is unique and there is no restriction on the input argument u of asinh(u) (-∞ < u < ∞):



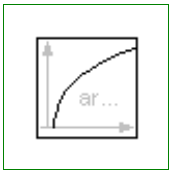
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Real	y	

Modelica.Math.acosh



Inverse of cosh (area hyperbolic cosine)

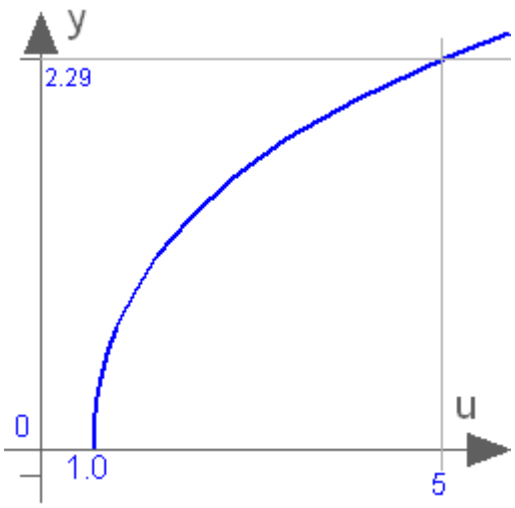
Information

This function returns the area hyperbolic cosine of its input argument u. The valid range of u is

```
+1 ≤ u < +∞
```

If the function is called with u < 1, an error occurs. The function cosh(u) has two inverse functions (the curve

looks similar to a sqrt(..) function). acosh(..) returns the inverse that is positive. At u=1, the derivative dy/du is infinite. Therefore, this function should not be used in a model, if u can become close to 1:



Inputs

Type	Name	Default	Description
Real	u		

Outputs

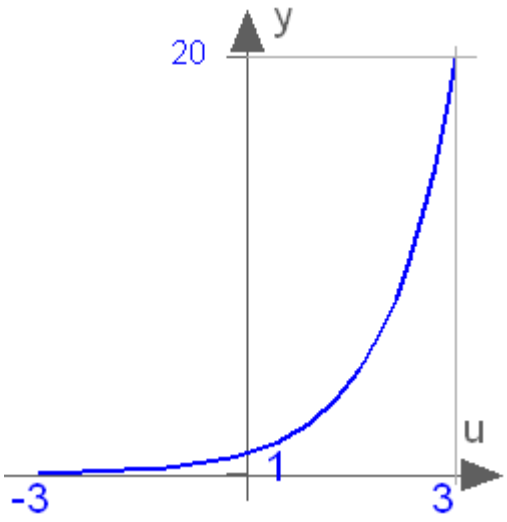
Type	Name	Description
Real	y	

Modelica.Math.exp

Exponential, base e

Information

This function returns y = exp(u), with -∞ < u < ∞:



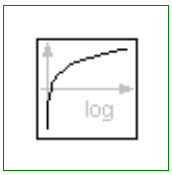
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Real	y	

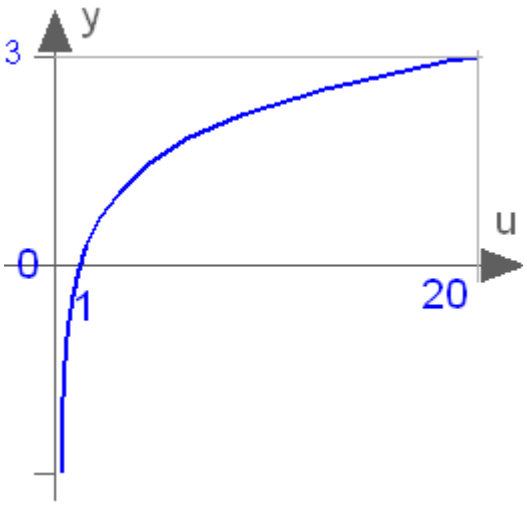
Modelica.Math.log



Natural (base e) logarithm (u shall be > 0)

Information

This function returns y = log(10) (the natural logarithm of u), with u > 0:



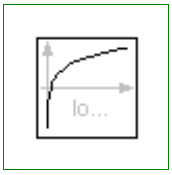
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Real	y	

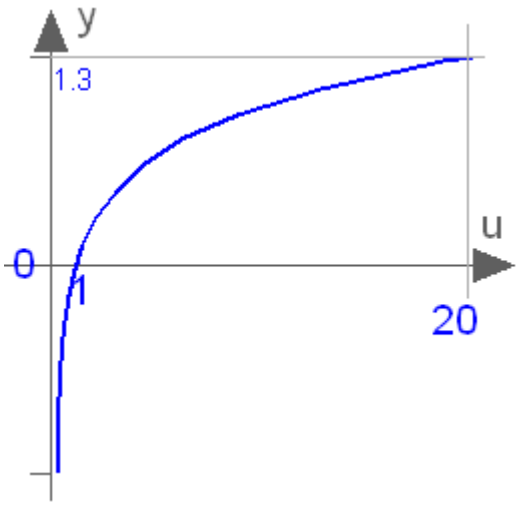
Modelica.Math.log10



Base 10 logarithm (u shall be > 0)

Information

This function returns y = log10(u), with u > 0:



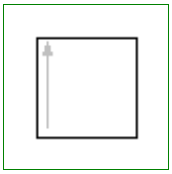
Inputs

Type	Name	Default	Description
Real	u		

Outputs

Type	Name	Description
Real	y	

Modelica.Math.baseIcon1

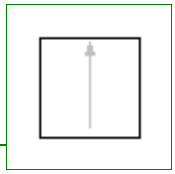


Basic icon for mathematical function with y-axis on left side

Information

Icon for a mathematical function, consisting of an y-axis on the left side. It is expected, that an x-axis is added and a plot of the function.

Modelica.Math.baseIcon2



Basic icon for mathematical function with y-axis in middle

Modelica.Math.tempInterpol1

Temporary function for linear interpolation (will be removed)

Information

Inputs

Type	Name	Default	Description
Real	u		input value (first column of table)

Real	table[:, :]		table to be interpolated
Integer	icol		column of table to be interpolated

Outputs

Type	Name	Description
Real	y	interpolated input value (icol column of table)

Modelica.Math.tempInterpol2

Temporary function for vectorized linear interpolation (will be removed)

Information

Inputs

Type	Name	Default	Description
Real	u		input value (first column of table)
Real	table[:, :]		table to be interpolated
Integer	icol[:]		column(s) of table to be interpolated

Outputs

Type	Name	Description
Real	y[1, size(icol, 1)]	interpolated input value(s) (column(s) icol of table)

Modelica.Mechanics

Library of 1-dim. and 3-dim. mechanical components (multi-body, rotational, translational)

Information

This package contains components to model the movement of 1-dim. rotational, 1-dim. translational, and 3-dim. **mechanical systems**.

Package Content

Name	Description
MultiBody	Library to model 3-dimensional mechanical systems
Rotational	Library to model 1-dimensional, rotational mechanical systems
Translational	Library to model 1-dimensional, translational mechanical systems

---

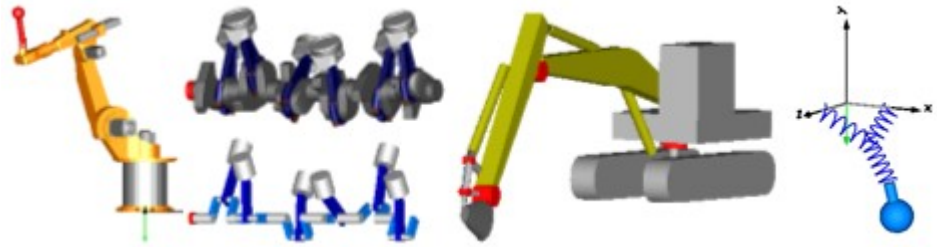
Modelica.Mechanics.MultiBody

Library to model 3-dimensional mechanical systems

Information

Library **MultiBody** is a **free** Modelica package providing 3-dimensional mechanical components to model in

a convenient way **mechanical systems**, such as robots, mechanisms, vehicles. Typical animations generated with this library are shown in the next figure:



For an introduction, have especially a look at:

- MultiBody.UsersGuide discusses the most important aspects how to use this library.
- MultiBody.Examples contains examples that demonstrate the usage of this library.

Note, that the MultiBody library replaces the long used ModelicaAdditions.MultiBody library. In MultiBody.UsersGuide.Upgrade it is described how to upgrade.

Copyright © 1998-2007, Modelica Association and DLR.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
UsersGuide	User's Guide of MultiBody Library
World	World coordinate system + gravity field + default animation definition
Examples	Examples that demonstrate the usage of the MultiBody library
Forces	Components that exert forces and/or torques between frames
Frames	Functions to transform rotational frame quantities
Interfaces	Connectors and partial models for 3-dim. mechanical components
Joints	Components that constrain the motion between two frames
Parts	Rigid components such as bodies with mass and inertia and massless rods
Sensors	Sensors to measure variables
Types	Constants and types with choices, especially to build menus
Visualizers	3-dimensional visual objects used for animation

Modelica.Mechanics.MultiBody.UsersGuide

User's Guide of MultiBody Library

Library **MultiBody** is a **free** Modelica package providing 3-dimensional mechanical components to model in a convenient way **mechanical systems**, such as robots, mechanisms, vehicles. This package contains the User's Guide for the MultiBody library.

1. Tutorial gives an introduction into the most important aspects of the library.
2. Upgrade describes how to upgrade from former versions, especially from the "old" ModelicaAdditions.MultiBody library.
3. Release Notes summarizes the differences between different versions of this library.

4. Literature provides references that have been used to design and implement this library.
5. Contact provides information about the author of the library as well as acknowledgments.

Package Content

Name	Description
Tutorial	Tutorial
Upgrade	Upgrade from Former Versions
ReleaseNotes	Release notes
Literature	Literature
Contact	Contact

Modelica.Mechanics.MultiBody.UsersGuide.Tutorial

Tutorial

This tutorial provides an introduction into the MultiBody library.

1. Overview of MultiBody library summarizes the most important aspects.
2. A first example describes in detail all the steps to build a simple pendulum model.
3. Loop structures explains how to model kinematic loops, especially by analytically solving non-linear equations.

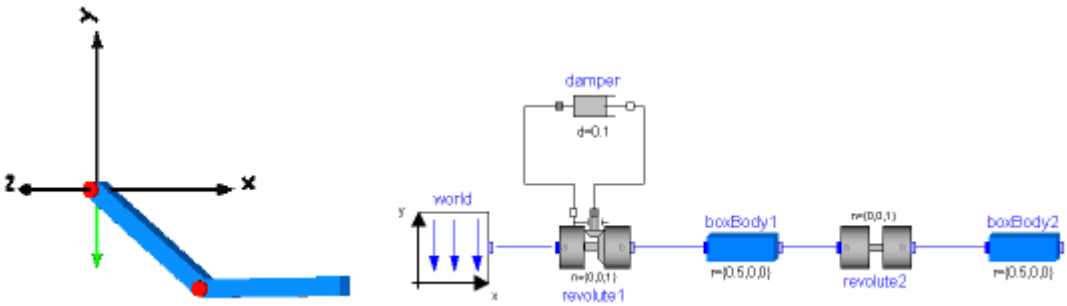
Package Content

Name	Description
OverView	Overview of MultiBody library
FirstExample	A first example
LoopStructures	Loop structures

Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.OverView

Overview of MultiBody library

Library **MultiBody** is a **free** Modelica package providing 3-dimensional mechanical components to model in a convenient way **mechanical systems**, such as robots, mechanisms, vehicles. A basic feature is that all components have **animation** information with appropriate default sizes and colors. A typical screenshot of the animation of a double pendulum is shown in the figure below, together with its schematic.



Note, that all components - the coordinate system of the world frame, the gravity acceleration vector, the

revolute joints and the bodies - are visualized in the animation.
This library replaces the long available ModelicaAdditions.MultiBody library, since it is much more easier to use and more powerful. The main features of the library are:

- About **60 main components**, i.e., joint, force, part, body, sensor and visualizer components that are ready to use and have useful default animation properties. One-dimensional force laws can be defined with components of the Modelica.Mechanics.Rotational and of the Modelica.Mechanics.Translational library and can be connected via available flange connectors to MultiBody components.
- About **75 functions** to operate in a convenient way on orientation objects, e.g., to transform vector quantities between frames, or compute the orientation object of a planar rotation. The basic idea is to hide the actual definition of an **orientation** by providing essentially an **Orientation** type together with **functions** operating on instances of this type. Orientation objects based on a 3x3 transformation matrix and on quaternions are provided. As a side effect, the equations in all other components are simpler and easier to understand.
- **A World model** has to be present in every model on top level. Here the gravity field is defined (currently: no gravity, uniform gravity, point gravity), the visualization of the world coordinate system and default settings for animation. If a world model is not present, it is automatically provided together with a warning message.
- **Built-in animation properties** of all components, such as joints, forces, bodies, sensors. This allows an easy visual check of the constructed model. Animation of every component can be switched off via a parameter. The animation of a complete system can be switched off via one parameter in the **world** model. If animation is switched off, all equations related to animation are removed from the generated code. This is especially important for real-time simulation.
- **Automatic handling of kinematic loops**. Components can be connected together in a nearly arbitrary fashion. It does not matter whether components are flipped. This does not influence the efficiency. If kinematic loop structures occur, this is automatically handeled in an efficient way by a new technique to transform a certain class of overdetermined sets of differential algebraic equations symbolically to a system where the number of equations and unknowns are the same (the user need **not** cut loops with special cut-joints to construct a tree-structure).
- **Automatic state selection from joints and bodies**. Most joints and all bodies have potential states. A Modelica translator, such as Dymola, will use the generalized coordinates of joints as states if possible. If this is not possible, states are selected from body coordinates. As a consequence, strange joints with 6 degrees of freedom are not necessary to define a body moving freely in space. An advanced user may select states manually from the **Advanced** menu of the corresponding components or use a Modelica parameter modification to set the "stateSelect" attribute directly.
- **Analytic solution of kinematic loops**. The non-linear equations occuring in kinematic loops are solved **analytically** for a large class of mechanisms, such as a 4 bar mechanism, a slider-crank mechanism or a MacPherson suspension. This is performed by constructing such loops with assembly joints JointXXX, available in the Modelica.Mechanics.MultiBody.Joints package. Assembly joints consist of 3 joints that have together 6 degrees of freedom, i.e., no constraints.They do not have potential states. When the motion of the two frame connectors are provided, a non-linear system of equation is solved analytically to compute the motion of the 3 joints. Analytic loop handling is especially important for real-time simulation.
- **Line force components may have mass**. Masses of line force components are located on the line on which the force is acting. They approximate the mass properties of a real physical device by one or two point masses. For example, a spring has often significant mass that has to be taken into account. If masses are set to zero, the additional code to handle these point masses is removed. If the masses are taken into account, the calculation overhead is small (the reason is that the occuring kinematic loops are analytically solved).
  Note, in this Beta-release, not all provided line force components have already an optional mass. This will be fixed in the next release.
- **Force components may be connected directly together**, e.g., 3-dimensional springs in series connection. Usually, multi-body programs have the restriction that force components can only be connected between two bodies. Such restrictions are not present in the Modelica multi-body library, since it is a fully object-oriented, equation based library. Usually, if force components are connected directly together, non-linear systems of equations occur. The advantage is often, that this may avoid stiff systems that would occur if a small mass has to be put in between the two force elements.

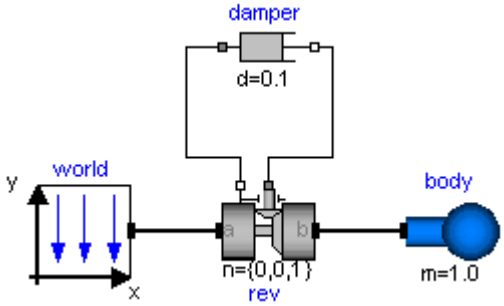
- **Initialization definition is available via menus**. Initialization of states in joints and bodies can be performed in the parameter menu, **without** typing Modelica statements. For non-standard initialization, the usual Modelica commands can be used.
- **Multi-body specific error messages**. Annotations and assert statements have been introduced that provide in many cases warning or error messages that are related to the library components (and not to specific equations as it is usual in Modelica libraries). This requires appropriate tool support, as it is, e.g., available in Dymola.
- **Inverse models** of mechanical systems can be easily defined by using motion generators, e.g., Modelica.Mechanics.Rotational.Position. Also, non-standard inverse models can be generated, e.g., when elasticity is present it might be necessary to differentiate equations several times.

Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.FirstExample

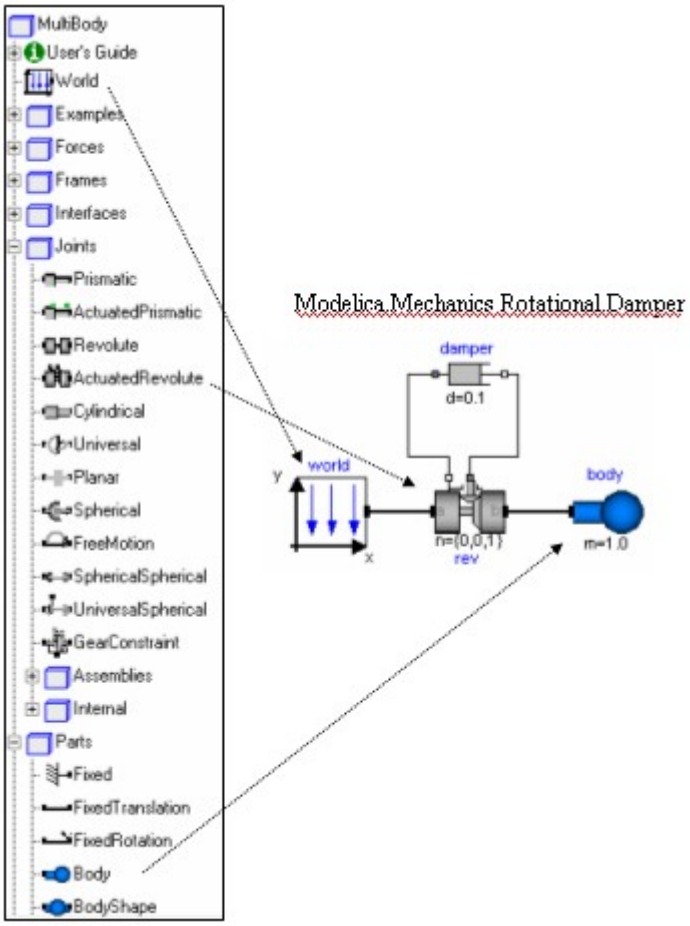
A first example

As a first example it shall be demonstrated how to build up, simulate and animate a **simple pendulum**.

A simple pendulum consisting of a **body** and a **revolute** joint with **linear damping** in the joint, is first build-up as Modelica composition diagram, resulting in:



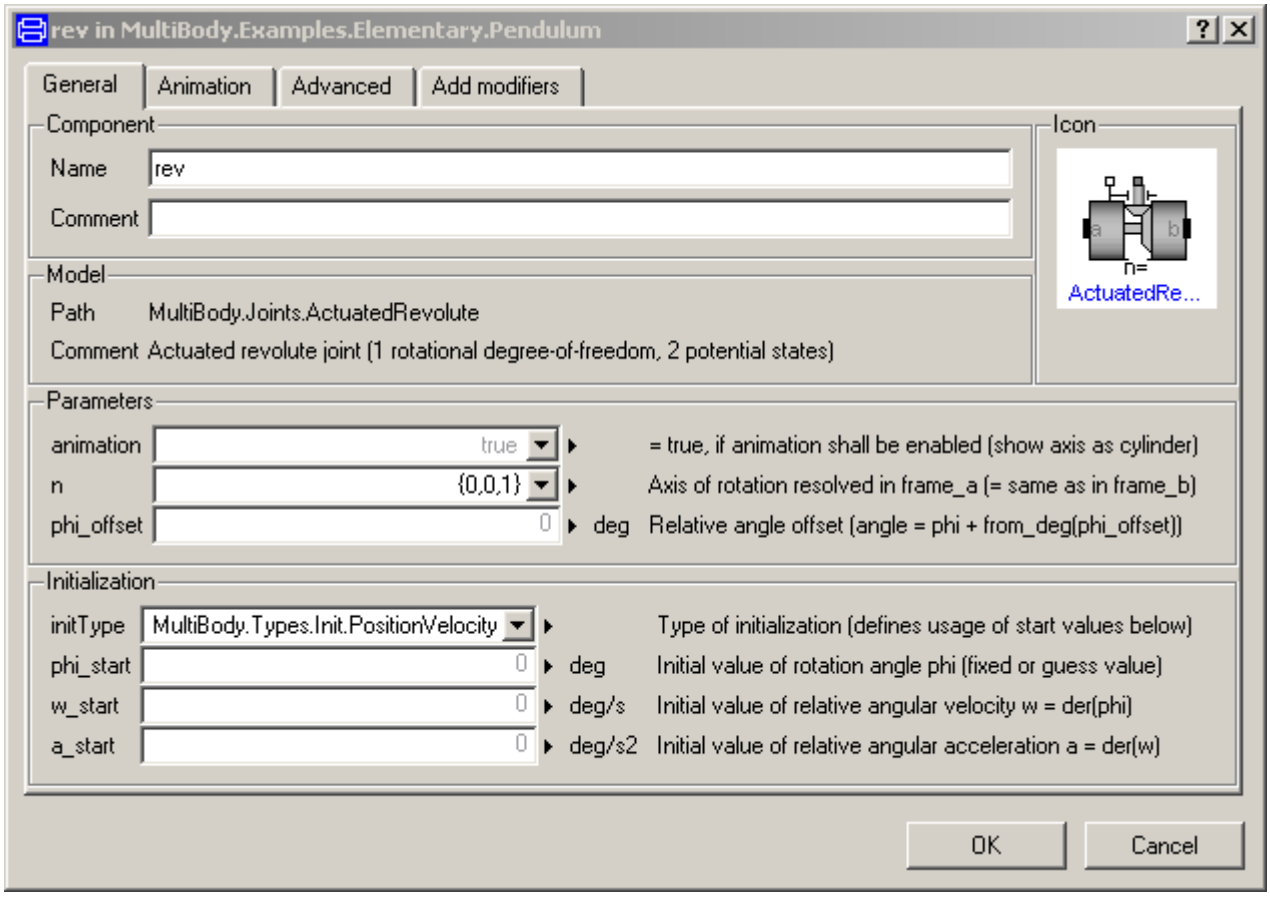
In the following figure the location of the used model components is shown. Drag these components in the diagram layer and connect them according to the figure:

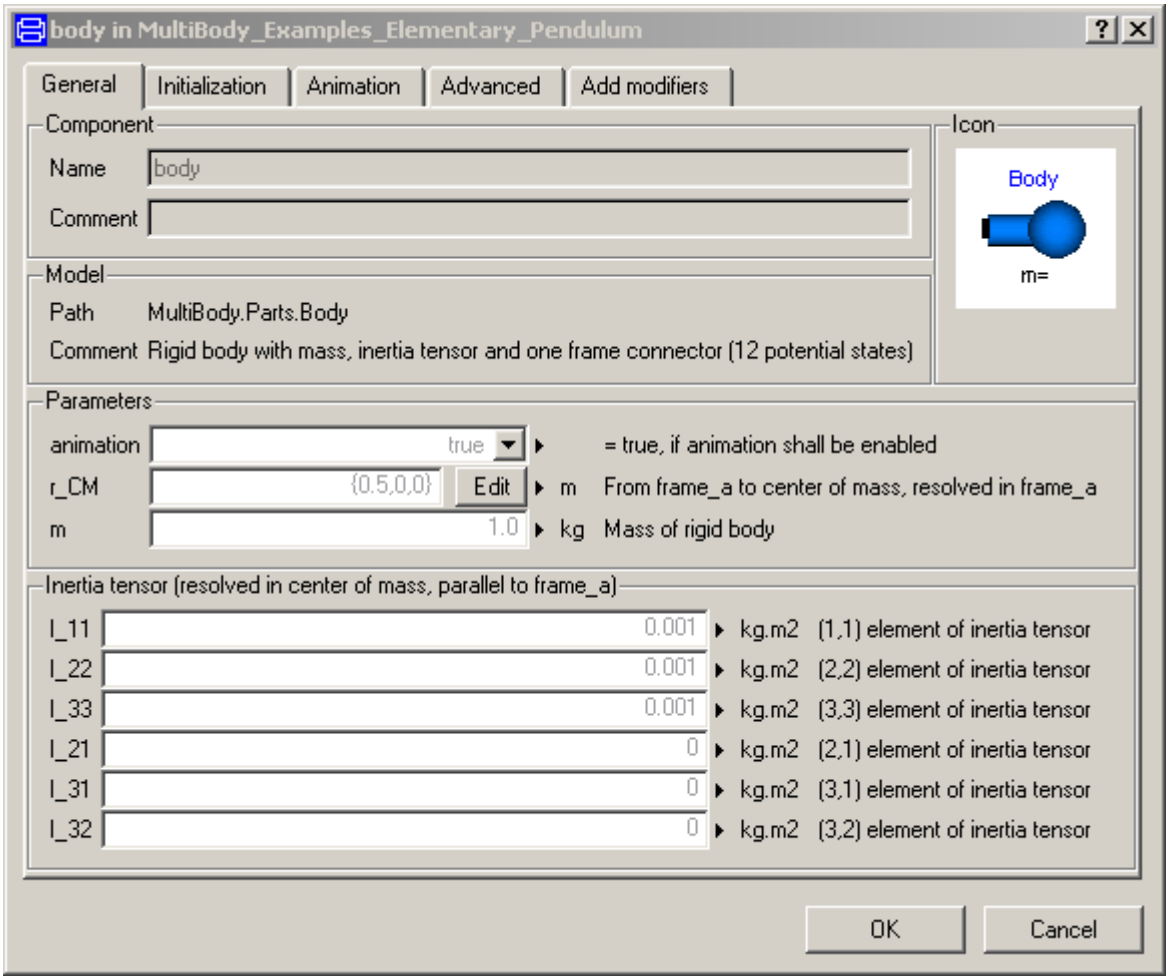


Every model that uses model components from the MultiBody library must have an instance of the Modelica.Mechanics.MultiBody.World model on highest level. The reason is that in the world object the gravity field is defined (uniform gravity or point gravity), as well as the default sizes of animation shapes and this information is reported to all used components. If the World object is missing, a warning message is printed and an instance of the World object with default settings is automatically utilized (this feature is defined with annotations and is, e.g., supported by Dymola).

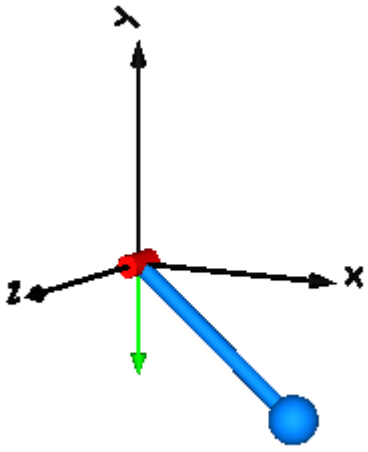
In a second step the parameters of the dragged components need to be defined. Some parameters are vectors that have to be defined with respect to a local coordinate system of the corresponding component. The easiest way to perform this is to define a **reference configuration** of your multi-body model: In this configuration, the relative coordinates of all joints are zero. This means that all coordinate systems on all components are parallel to each other. Therefore, this just means that all vectors are resolved in the world frame in this configuration.

The reference configuration for the simple pendulum shall be defined in the following way: The y-axis of the world frame is directed upwards, i.e., the opposite direction of the gravity acceleration. The x-axis of the world frame is orthogonal to it. The revolute joint is placed in the origin of the world frame. The rotation axis of the revolute joint is directed along the z-axis of the world frame. The body is placed on the x-axis of the world frame (i.e., the rotation angle of the revolute joint is zero, when the body is on the x-axis). In the following figures the definition of this reference configuration is shown in the parameter menus of the revolute joint and the body:


rev in MultiBody.Examples.Elementary.Pendulum
General
Animation
Advanced
Add modifiers
Component
Name
rev
Comment
Icon
a
b
n=
ActuatedRe...
Model
Path
MultiBody.Joints.ActuatedRevolute
Comment
Actuated revolute joint (1 rotational degree-of-freedom, 2 potential states)
Parameters
animation
true
= true, if animation shall be enabled (show axis as cylinder)
n
{0,0,1}
Axis of rotation resolved in frame_a (= same as in frame_b)
phi_offset
0
deg
Relative angle offset (angle = phi + from_deg(phi_offset))
Initialization
initType
MultiBody.Types.Init.PositionVelocity
Type of initialization (defines usage of start values below)
phi_start
0
deg
Initial value of rotation angle phi (fixed or guess value)
w_start
0
deg/s
Initial value of relative angular velocity w = der(phi)
a_start
0
deg/s2
Initial value of relative angular acceleration a = der(w)
OK
Cancel



Translate and simulate the model, e.g., with Dymola. Automatically, all defined components are visualized in an animation using default absolute or relative sizes of the components. For example, a body is visualized as a sphere and as a cylinder. The default size of the sphere is defined as parameter in the world object. You may change this size in the "Animation" parameter menu of the body (see parameter menu above). The default size of the cylinder is defined relatively to the size of the sphere (half of the sphere size). With default settings, the following animation is defined:



The world coordinate system is visualized as coordinate system with axes labels. The direction of the gravity acceleration vector is shown as green arrow. The red cylinder represents the rotation axis of the revolute

joint and the light blue shapes represent the body. The center of mass of the body is in the middle of the light blue sphere.

Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.LoopStructures

Loop structures

The MultiBody library has the feature that all components can be connected together in a nearly arbitrary fashion. Therefore, kinematic loop structures pose in principal no problems. In this section several examples are given, the special treatment of planar loops is discussed and it is explained how a kinematic loop structure can be modeled such that the occuring non-linear algebraic equation systems are solved analytically. There are the following sub-chapters:

1. Introduction
2. Planar loops.
3. Analytic loop handling.

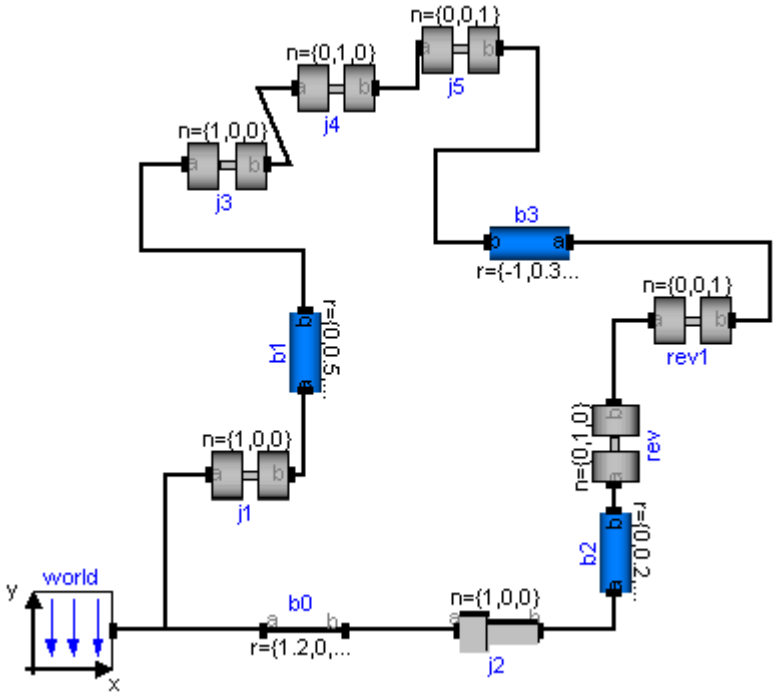
Package Content

Name	Description
Introduction	Introduction
PlanarLoops	Planar loops
AnalyticLoopHandling	Analytic loop handling

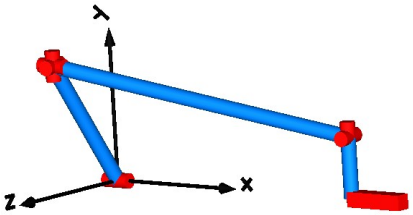
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.LoopStructures.Introduction

Introduction

In principal, now special action is needed, if loop structures occur (contrary to the ModelicaAdditions.MultiBody library). An example is presented in the figure below. It is available as MultiBody.Examples.Loops.Fourbar1



This mechanism consists of 6 revolute joints, 1 prismatic joint and forms a kinematical loop. It has has one degree of freedom. In the next figure the default animation is shown. Note, that the axes of the revolute joints are represented by the red cylinders and that the axis of the prismatic joint is represented by the red box on the lower right side.



Whenever loop structures occur, non-linear algebraic equations are present on "position level". It is then usually not possible by structural analysis to select states during translation (which is possible for non-loop structures). In the example above, Dymola detects a non-linear algebraic loop of 57 equations and reduces this to a system of 7 coupled algebraic equations. Note, that this is performed without using any "cut-joints" as it is usually done in multi-body programs, but by just appropriate symbolic equation manipulation. Via the dynamic dummy derivative method the generalized coordinates on position and velocity level from one of the 7 joints are dynamically selected as states during simulation. Whenever, these two states are no longer appropriate, states from one of the other joints are selected during simulation.

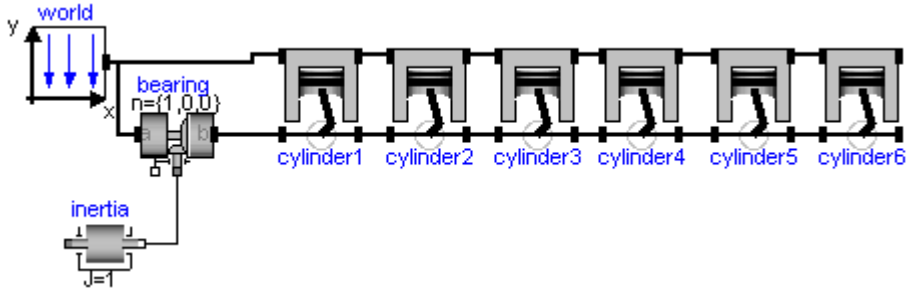
The efficiency of loop structures can usually be enhanced, if states are statically fixed at translation time. For this mechanism, the generalized coordinates of joint j1 (i.e., the rotation angle of the revolute joint and its derivative) can always be used as states. This can be stated by setting parameter "enforceStates = **true**" in the "Advanced" menu of the desired joint. This flag sets the attribute stateSelect of the generalized coordinates of the coresponding joint to "StateSelect.always". When setting this flag to **true** for joint j1 in the four bar mechanism, Dymola detects a non-linear algebraic loop of 40 equations and reduces this to a system of 5 coupled non-linear algebraic equations.

In many mechanisms it is possible to solve the non-linear algebraic equations analytically. For a certain class of systems this can be performed also with the MultiBody library. This technique is described in section "Analytic loop handling".

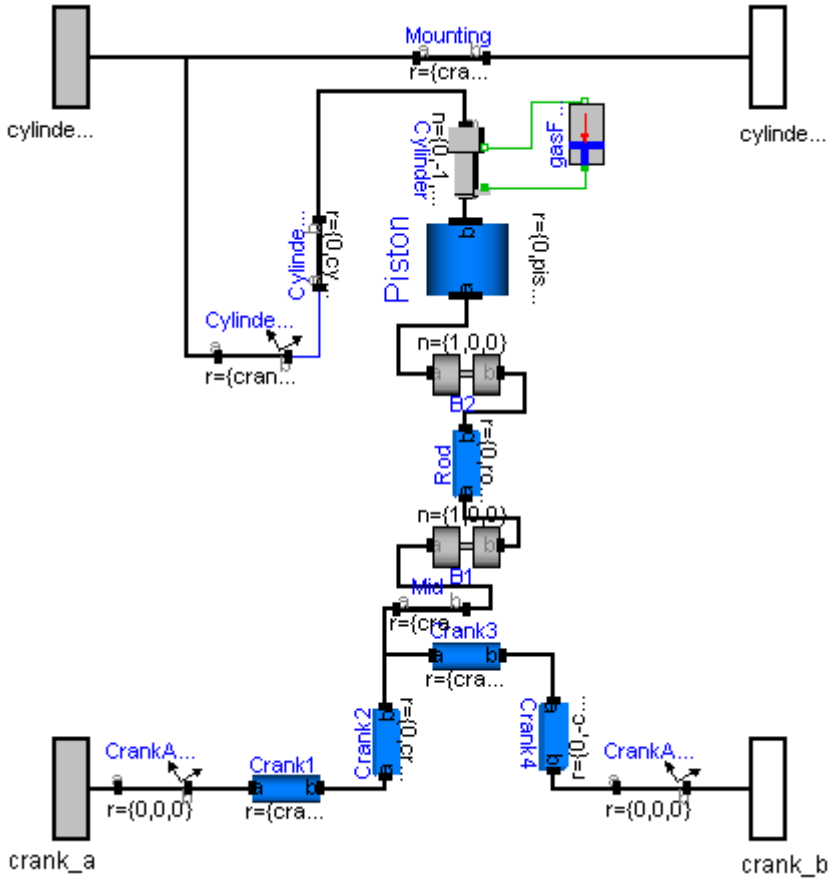
Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.LoopStructures.PlanarLoops

Planar loops

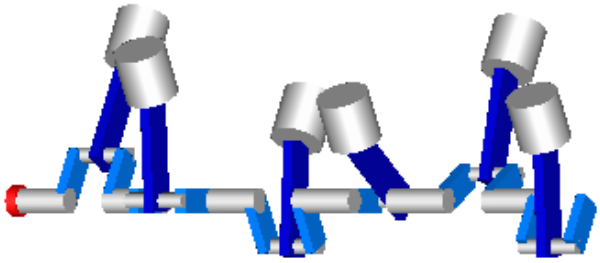
In the figure below, the model of a V6 engine is shown that has a simple combustion model. It is available as MultiBody.Examples.Loops.EngineV6.



The Modelica schematic of one cylinder is given in the figure below. Connecting 6 instances of this cylinder appropriately together results in the engine schematic displayed above.



In the next figure the animation of the engine is shown. Every cylinder consists essentially of 1 prismatic and 2 revolute joints that form a planar loop, since the axes of the two revolute joints are parallel to each other and the axis of the prismatic joint is orthogonal to the revolute joint axes. All 6 cylinders together form a coupled set of 6 loops that have together 1 degree of freedom.



All planar loops, and especially the engine, result in a DAE (= Differential-Algebraic Equation system) that does not have a unique solution. The reason is that, e.g., the cut forces in direction of the axes of the revolute joints cannot be uniquely computed. Any value fulfills the DAE equations. This is a structural property that is determined by the symbolic algorithms. Since they detect that the DAE is structurally singular, a further processing is not possible. Without additional information it is also impossible that the symbolic algorithms could be enhanced because if the axes of rotations of the revolute joints are only slightly changed such that they are no longer parallel to each other, the planar loop can no longer move and has 0 degrees of freedom. Algorithms based on pure structural information cannot distinguish these two cases.

The usual remedy is to remove superfluous constraints, e.g., along the axis of rotation of **one** revolute joint. Since this is not easy for an inexperienced modeler, the flag "**planarCutJoint**" is provided in the "**Advanced**" menu of a revolute joint that removes these constraints. This flag must be set to **true** for one revolute joint in every planar loop. In the engine example, this flag is set in the revolute joint B2 in the cylinder model.

If a modeler is not aware of the problems with planar loops and models them without special consideration, a

Modelica translator, such as Dymola, displays an error message and points out that a planar loop may be the reason and suggests to use the "planarCutJoint" flag. This error message is due to an annotation in the Frame connector.

```
connector Frame
   ...
   flow SI.Force f[3] annotation(unassignedMessage="..");
end Frame;
```

If no assignment can be found for some forces in a connector, the "unassignedMessage" is displayed. In most cases the reason for this is a planar loop or two joints that constrain the same motion. Both cases are discussed in the error message.

Note, that the non-linear algebraic equations occurring in planar loops can be solved analytically in most cases and therefore it is highly recommended to use the techniques discussed in section "Analytic loop handling" for such systems.

Modelica.Mechanics.MultiBody.UsersGuide.Tutorial.LoopStructures.AnalyticLoopHandling

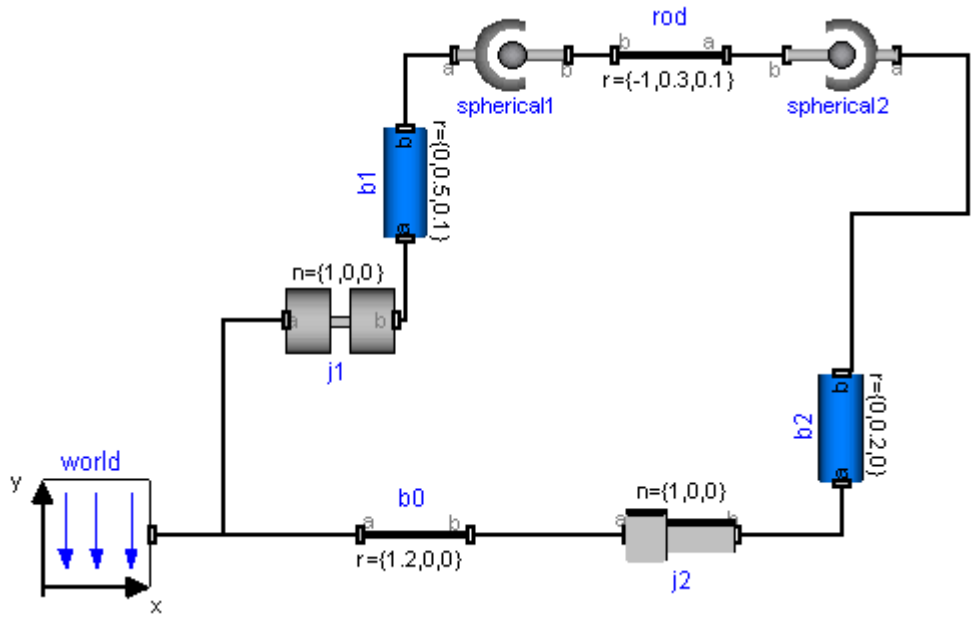
Analytic loop handling

It is well known that the non-linear algebraic equations of most mechanical loops in technical devices can be solved analytically. It is, however, difficult to perform this fully automatically and therefore none of the commercial, general purpose multi-body programs, such as MSC ADAMS, LMS DADS, SIMPACK, have this feature. These programs solve loop structures with pure numerical methods. Multi-body programs that are designed for real-time simulation of the dynamics of specific vehicles, such as ve-DYNA, usually contain manual implementations of a particular multi-body system (the vehicle) where the occurring loops are either analytically solved, if this is possible, or are treated by table look-up where the tables are constructed in a pre-processing phase. Without these features the required real-time capability would be difficult to achieve.

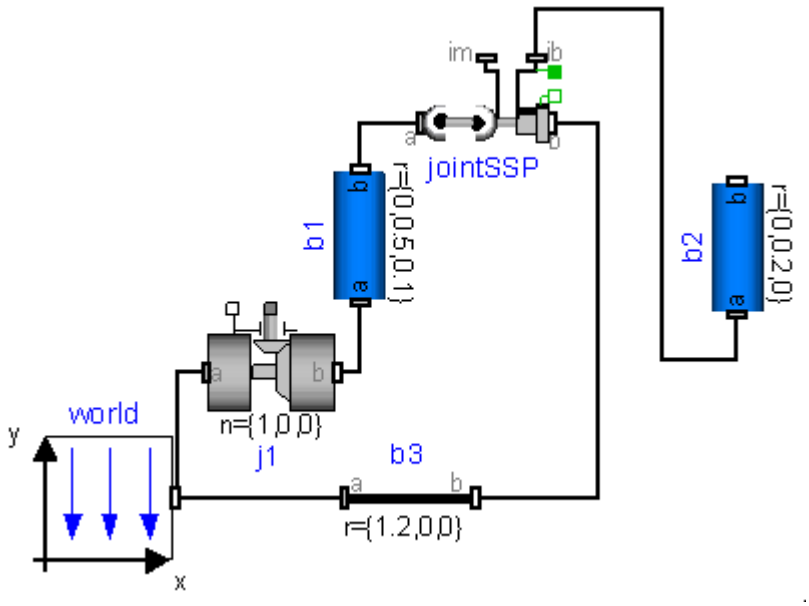
In a series of papers and dissertations Prof. Hiller and his group in Duisburg, Germany, have developed systematic methods to handle mechanical loops analytically. The "characteristic pair of joints" method basically cuts a loop at two joints and uses geometric invariants to reduce the number of algebraic equations, often down to one equation that can be solved analytically. Also several multi-body codes have been developed that are based on this method, e.g., MOBILE. Besides the very desired feature to solve non-linear algebraic equations analytically, i.e., efficiently and in a robust way, there are several drawbacks: It is difficult to apply this method automatically. Even if this would be possible in a good way, there is always the problem that it cannot be guaranteed that the statically selected states lead to no singularity during simulation. Therefore, the "characteristic pair of joints" method is usually manually applied which requires know-how and experience.

In the MultiBody library the "characteristic pair of joints" method is supported in a restricted form such that it can be applied also by non-specialists. The idea is to provide aggregations of joints in package MultiBody.Joints.Assemblies. as one object that either have **6** degrees of freedom or **3** degrees of freedom (for usage in planar loops).

As an example, a variant of the four bar mechanism is given in the figure below.



Here, the mechanism is modeled with one revolute joint, two spherical joints and one prismatic joint. In the figure below, the two spherical joints and the prismatic joint are collected together in an assembly object called "jointSSP" from MultiBody.Joints.Assemblies.JointSSP.



The JointSSP joint aggregation has a frame at the left side of the left spherical joint (frame_a) and a frame at the right side of the prismatic joint (frame_b). JointSSP, as all other objects from the Joints.Assemblies package, has the property, that the **generalized coordinates, and all other frames defined in the assembly, can be calculated given the movement of frame_a and of frame_b**. This is performed by **analytically** solving non-linear systems of equations (details are given in section xxx). From a structural point of view, the equations in an assembly object are written in the form

$$\mathbf{q} = \mathbf{f}_1(\mathbf{r}^a, \mathbf{R}^a, \mathbf{r}^b, \mathbf{R}^b)$$

where $\mathbf{r}^a$, $\mathbf{R}^a$, $\mathbf{r}^b$, $\mathbf{R}^b$ are the variables defining the position and orientation of the frame_a and frame_b connector, **q** are the generalized positional coordinates inside the assembly, e.g., the angle of a revolute joint. Given angle φ of revolute joint j1 from the four bar mechanism, frame_a and frame_b of the assembly object can be computed by a forward recursion

$$(\mathbf{r}^a, \mathbf{R}^a, \mathbf{r}^b, \mathbf{R}^b) = \mathbf{f}(\varphi)$$

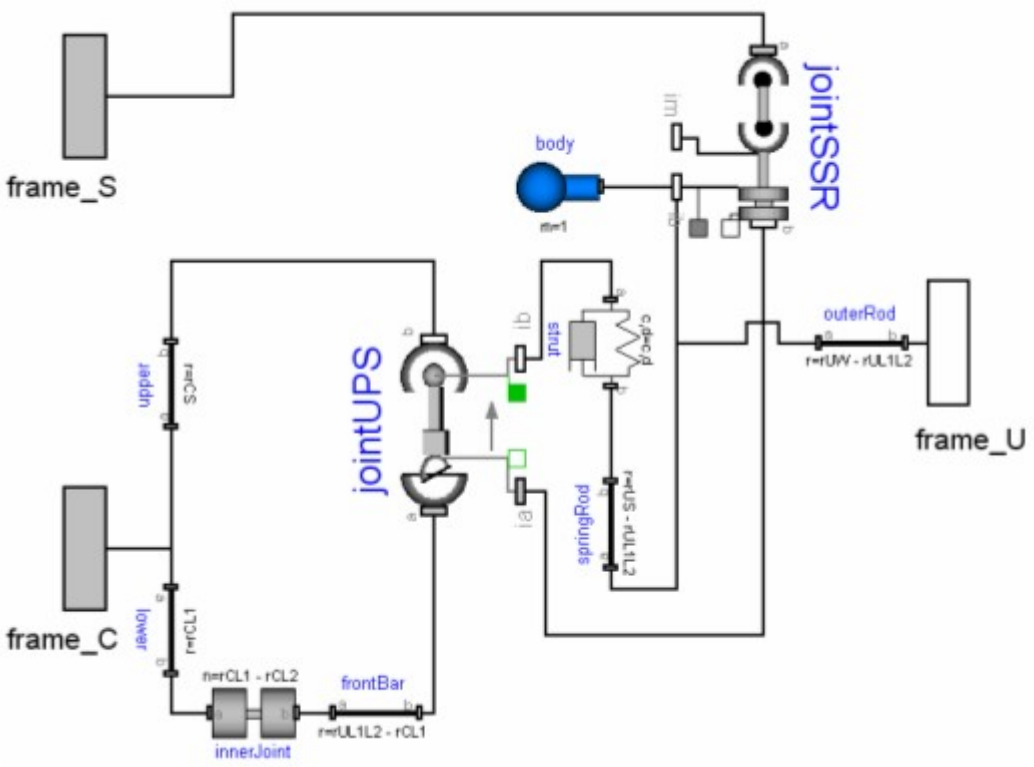
Since this is a structural property, the symbolic algorithms can automatically select φ and its derivative as states and then all positional variables can be computed in a forwards sequence. It is now understandable that a Modelica translator, such as Dymola, can transform the equations of the four bar mechanism to a recursive sequence of statements that has no non-linear algebraic loops anymore(remember, the previous "straightforward" solution with 6 revolute joints and 1 prismatic joint has a nonlinear system of equations of order 5).

The aggregated joint objects consist of a combination of either a revolute or prismatic joint and of a rod that has either two spherical joints at its two ends or a spherical and a universal joint, respectively. For all combinations, analytic solutions can be determined. For planar loops, combinations of 1, 2 or 3 revolute joints with parallel axes and of 2 or 1 prismatic joint with axes that are orthogonal to the revolute joints can be treated analytically. The currently supported combinations are listed in the table below. The missing combinations (such as JointSUP or Joint RPP) will be added in one of the next releases.

3-dimensional Loops:	
JointSSR	Spherical - Spherical - Revolute
JointSSP	Spherical - Spherical - Prismatic
JointUSR	Universal - Spherical - Revolute
JointUSP	Universal - Spherical - Prismatic
JointUPS	Universal - Prismatic - Spherical
Planar Loops:	
JointRRR	Revolute - Revolute - Revolute
JointRRP	Revolute - Revolute - Prismatic

On first view this seems to be quite restrictive. However, mechanical devices are usually built up with rods connected by spherical joints on each end, and additionally with revolute and prismatic joints. Therefore, the combinations of the above table occur frequently. The universal joint is usually not present in actual devices but is used (a) if two JointXXX components can be connected such that a revolute and a universal joint together form a spherical joint and (b) if the orientation of the connecting rod between two spherical joints is needed, e.g., since a body shall be attached. In this case one of the spherical joints might be replaced by a universal joint. This approximation is fine as long as the mass and inertia of the rod is not significant.

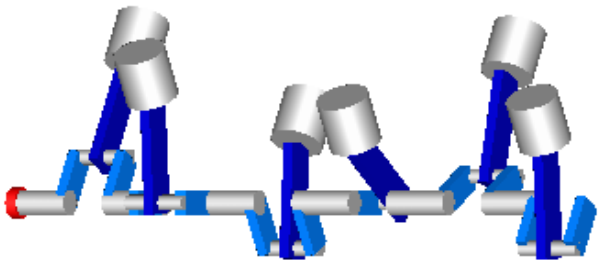
Let us discuss item (a) in more detail: The MacPherson suspension in the next figure is from the Modelica VehicleDynamics library.

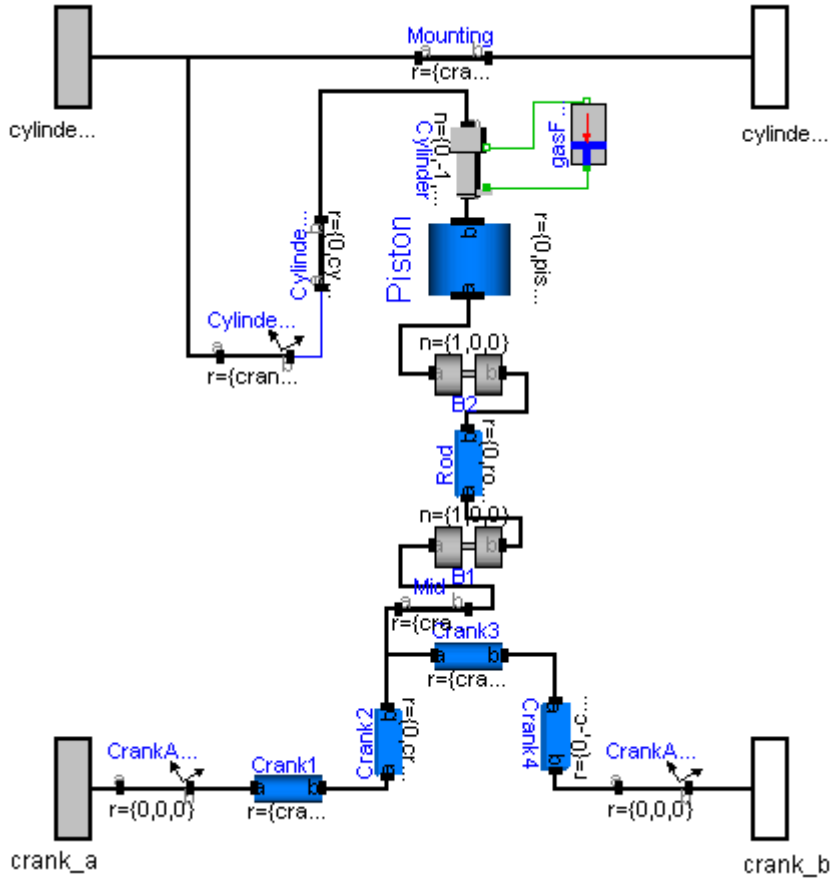


It has three frame connectors. The lower left one (frame_C) is fixed in the vehicle chassis. The upper left one (frame_S) is driven by the steering mechanism, i.e., the movement of both frames are given. The frame connector on the right (frame_U) drives the wheel. The three frames are connected by a mechanism consisting essentially of two rods with spherical joints on both ends. These are built up by a jointUPS and a jointSSR assembly. As can be seen, the universal joint from the jointUPS assembly is connected to the revolute joint of the jointSSR assembly. Therefore, we have 3 revolute joints connected together at one point and if the axes of rotations are chosen appropriately, this describes a spherical joint. In other words, the two connected assemblies define the desired two rods with spherical joints on each ends.

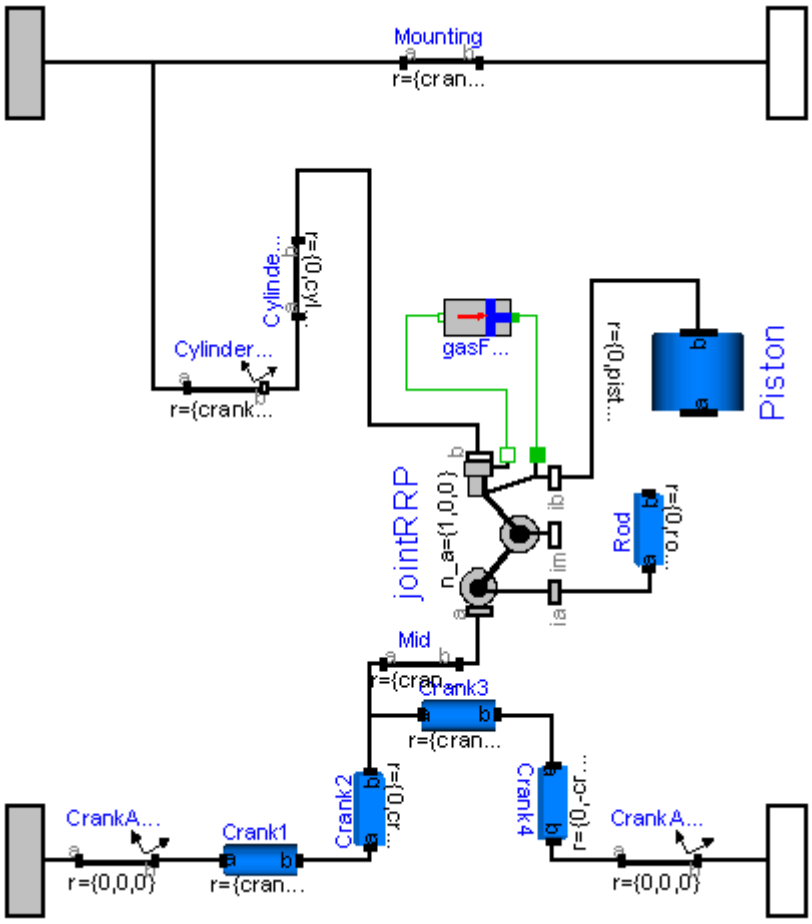
The movement of the chassis, frame_C, is computed somewhere else. When the generalized coordinates of revolute joint "innerJoint" (lower left part in figure) are used as states, then frame_a and frame_b of the jointUPS joint can be calculated. After the non-linear loop with jointUPS is (analytically) solved, all frames on this assembly are known, especially, the one connected to frame_b of the jointSSR assembly. Since frame_b of jointSSR is connected to frame_S which is computed from the steering mechanism, again the two required frame movements of the jointSSR assembly are calculated, meaning in turn that also all other frames on the jointSSR assembly can be computed, especially, the one connected to frame_U that drives the wheel. From this analysis it is clear that a tool is able to solve these coupled loops analytically.

Another example is the model of the V6 engine, see next figure for an animation view and the original definition of one cylinder with elementary joints.



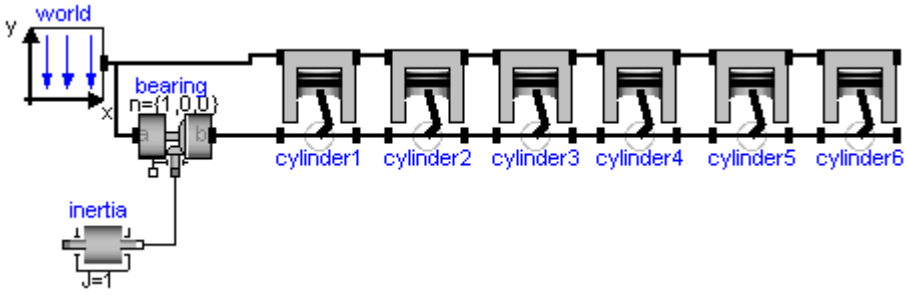


It is sufficient to rewrite the basic cylinder model by replacing the joints with a JointRRP object that has two revolute and one prismatic joint, see next figure.



Since 6 cylinders are connected together, 6 coupled loops with 6 JointRRP objects are present. This model is available as MultiBody.Examples.Loops.EngineV6_analytic.

The composition diagram of the connected 6 cylinders is shown in the next figure



It can be seen that the revolute joint of the crank shaft (joint "bearing" in left part of figure) might be selected as degree of freedom. Then the 4 connector frames of all cylinders can be computed. As a result the computations of the cylinders are decoupled from each other. Within one cylinder the position of frame_a and frame_b of the jointRRP assembly can be computed and therefore the generalized coordinates of the two revolute and the prismatic joint in the jointRRP object can be determined. From this analysis it is not surprising that a Modelica translator, such as Dymola, is able to transform the DAE equations into a sequential evaluation without any non-linear loop. Compare this nice result with the model using only elementary joints that leads to a DAE with 6 algebraic loops and 5 non-linear equations per loop. Additionally, a linear system of equations of order 43 is present. The simulation time is about 5 times faster with the analytic loop handling.

Modelica.Mechanics.MultiBody.UsersGuide.Upgrade

Upgrade from Former Versions

If different versions of the MultiBody library are not compatible to each other, corresponding conversion scripts are provided. As a result, models build with an older version of the MultiBody library are automatically converted to the new version when the model is loaded. The user is prompted whether automatic conversion shall take place or not. Problems are not to be expected. Still one should first make a copy of such a model as backup before the conversion is performed.

Upgrade from ModelicaAdditions.MultiBody

There is now also a conversion script from the "old" **ModelicaAdditions.MultiBody** library to the "new" Modelica.Mechanics.MultiBody library. This script is also automatically invoked. Since the differences between the "old" and the "new" MultiBody library are so large, not everything is converted and it might be that some pieces have to be adapted manually. Still, this script is useful, since many class names, parameters and modifiers are automatically converted.

Components from the following sublibraries are automatically converted to the Modelica.Mechanics.MultiBody library:

- ModelicaAdditions.MultiBody.Parts
- ModelicaAdditions.MultiBody.Joints
- ModelicaAdditions.MultiBody.Forces
- Part of ModelicaAdditions.MultiBody.Interfaces

Models using the ModelicaAdditions.MultiBody library that are programmed with **equations** are only partly converted: The Frame connectors will be converted to the "new" Frame connectors of the MultiBody library, but the equations that reference variables of the Frame connectors will **not** be converted. For a manual conversion, the following table might be helpful showing how the **variables** of the "old" and the "new" **Frame connectors** are related to each other (resolve2 and angularVelocity2 are functions from library Modelica.Mechanics.MultiBody.Frames):

ModelicaAdditions.MultiBody. Interfaces.Frame_a	**MultiBody.Interfaces.Frame_a**
frame_a.**r0**	= frame_a.r_0 (is converted)
frame_a.**S**	= transpose(frame_a.R)
frame_a.**v**	= resolve2(frame_a.R, **der**(frame_a.r_0))
frame_a.**w**	= angularVelocity2(frame_a.R)
frame_a.**a**	= resolve2(frame_a.R, **der**(v_0)); v_0 = der(r_0)
frame_a.**z**	= **der**(w); w = angulaVelocity2(frame_a.R)
frame_a.**f**	= frame_a.f (no conversion needed)
frame_a.**t**	= frame_a.t (no conversion needed)

Upgrade from MultiBody 0.99 (and earlier) to 1.0 (and later)

The conversion from MultiBody 0.99 to 1.0 does not work in some rare cases, where own components are implemented using functions of the MultiBody.Frames package. In this case, the conversion has to be performed manually. The changes in 1.0 with regards to 0.99 are:

The definition of the Modelica.Mechanics.MultiBody.Frames.Orientation object has changed. In 0.99 this was just an alias type for a transformation matrix (now Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.Orientation). In 1.0 the orientation object is a record holding the transformation matrix from frame 1 to frame 2 and the angular velocity of the transformation matrix resolved in frame 2. The reason is that this allows to compute the angular velocity in many cases by standard recursive formulas and not by differentiation of the transformation matrix. This is usually much more efficient. As a consequence, the following calls in 0.99 should be changed:

```
Frames.angularVelocity1(T,der(T)) -> Frames.angularVelocity1(T)
Frames.angularVelocity2(T,der(T)) -> Frames.angularVelocity2(T)
Frames.from_T(T)                  -> Frames.from_T2(T,der(T))
```

Modelica.Mechanics.MultiBody.UsersGuide.ReleaseNotes

Release notes

Version 1.1.3, 2005-04-21

This version has some minor improvements:

- Component MultiBody.**Sensors.Distance** has been modified: If the distance is less than s_small (a parameter in the "advanced" menu), it is approximated such that the derivative is finite for zero distance. Previously, an assert was triggered.

Version 1.1.2, 2005-04-06

This version has some minor improvements:

- Component MultiBody.**World** has a new parameter **driveTrainMechanics3D**. If set to **true**, 3-dim. mechanical effects of MultiBody.Parts.Mounting1D/Rotor1D/BevelGear1D are taken into account. If set to **false** (= default), 3-dim. mechanical effects in these elements are not taken into account and the frame connectors to connect to 3-dim. parts are disabled (all connections to such a disabled connector are also disabled, due to the new feature of conditional declarations in Modelica language 2.2).
- All references to "MultiBody.xxx" have been changed to "Modelica.Mechanics.MultiBody.xxx" in order that after copying of a component outside of the Modelica library, the references still remain valid.

Version 1.1.1, 2004-11-03

This version has some minor improvements:

- New function MultiBody.**Frames.from_T2** to compute an orientation object from a transformation matrix and its derivative.
- Added a paragraph in the "Upgrade from Former Versions" section of the User's Guide to show how a conversion from 0.99 to 1.0 that may fail in rare cases should be fixed manually.

Version 1.1, 2004-07-07

Included the MultiBody library as Modelica.Mechanics.MultiBody in version 2.0 Beta 1 of the Modelica package, and adapted all signal connectors to the new definition of Modelica.Blocks.Interfaces.

Version 1.0.1, 2004-07-07

This version fixes only some minor bugs:

- Bug in Modelica.Mechanics.MultiBody.Sensors.**RelativeSensor** fixed:
  When frame resolve was connected and get_v_rel=true or get_a_rel=true, an error was in the code.
- Bug in MultiBodys.Sensors.**Power** fixed:
  A "defineBranch(...)" statement was missing. In certain cases, it was then not possible to generate code.
- Wrong icon of MultiBody and of Modelica.Mechanics.MultiBody.Interfaces package corrected.

Version 1.0, 2004-03-03

This version is **not** backward compatible to version 0.99. Models generated with previous MultiBody versions are automatically converted to the new release.The incompatible changes are due to improving the efficiency of the library. This required to change function interfaces in Modelica.Mechanics.MultiBody.Frames: angularVelocity1, angularVelocity2, planarRotation, axisRotation, axesRotations, from_T, from_T_inv, from_Q. Furthermore, Modelica.Mechanics.MultiBody.Frames.Orientation has changed. As a consequence, all code that accesses objects of this type directly is no longer valid. Otherwise, the following changes have

been made:

- **Faster code**:
  Due to a different strategy to handle orientation objects, the number of arithmetic expressions in the generated code is reduced (for larger models there is a speed-up of about 2-3).
- Improved **documentation**:
  Tutorial renamed to "User's Guide" and documents restructured and improved. Included 3-dimensional images for nearly every component description where it makes sense. Improved the documentation of the packages.
- New package Modelica.Mechanics.MultiBody.Frames.**TransformationMatrices**:
  This is a copy of the previous Modelica.Mechanics.MultiBody.Frames.XX functions. The new Modelica.Mechanics.MultiBody.Frames.XX functions have a different orientation object that can no longer be directly used, since the angular velocity calculation is implicitly included in the functions to improve efficiency significantly
- New function Modelica.Mechanics.MultiBody.Frames.**resolveRelative**:
  Transform vector from frame 1 to frame 2 using absolute orientation objects of frame 1 and of frame 2.
- Improved handling of orientation in Modelica.Mechanics.MultiBody.Joints.**Spherical**, Modelica.Mechanics.MultiBody.Joints.**FreeMotion**, Modelica.Mechanics.MultiBody.Parts.**Body**:
  If states are used from these components, the orientation is either described by quaternions or by 3 angles along desired sequence axes. If 3 angles are used, the code is such that symbolic transformation methods for inline integration can generate efficient code. In "Spherical" and "Body" the switching between different sets of 3 angles has been removed since this may lead to difficulties for the symbolic transformations. In "FreeMotion" the option to use quaternions has been added.
- New option for "MultiBody.Sensors.**AbsoluteSensor**":
  Optionally, the angles to rotate the world frame into frame_a are returned.
- New option for "MultiBody.Sensors.**RelativeSensor**":
  Optionally, the angles to rotate frame_a into frame_b are returned.
- Improved "MultiBody.Examples.Loops.**EngineV6_analytic**":
  Model restructured and animated with CAD data.
- **Specular coefficient** introduced with default 0.7:
  The components in Visualizers.Advanced have as additional input the specular coefficient describing the reflection of ambient light. The default is 0.7. Coordinate systems have a specular coefficient of 0 (= light is not reflected).
- Improved Modelica.Mechanics.MultiBody.Frames.**Frame_resolve**:
  The frame outline is with a full line. When drawing a connection line from Frame_resolve to another frame, this connection line is dotted.

Version 0.99, 2004-02-16

This version is **fully** backward compatible to version 0.98. The following changes have been made:

- New options for "MultiBody.Joints.**Spherical**".
  This joint may have optionally Cardan angles or quaternions as states. Additionally, an "Initialization" tab has been added in order that the relative coordinates of the joint can be used for initialization.
- New model "MultiBody.Joints.**GearConstraint**"
  to model the 3-dim. constraint of a gear box.
- New model "MultiBody.Parts.**Mounting1D**"
  to propagate 1-dim. support torques to a 3-dim. mounting.
- New model "MultiBody.Parts.**Rotor1D**"
  to attach a 1-dim. inertia on 3-dim. mountings without neglecting dynamic effects.
- New model "MultiBody.Parts.**BevelGear1D**"
  that describes a 1-dim. gearbox with arbitrary shaft directions and that is attached on a 3-dim. mounting.
- New package "MultiBody.Examples.Systems.**RobotR3**".
  The models of this package are used to demonstrate in which way complex robot models might be built up by testing first the component models individually before composing them together. Furthermore, it is shown how CAD data can be used for animation.
- New model "MultiBody.Sensors.**AbsoluteSensor**"

to provide kinematic quantities of a frame as output signals resolved in the local frame, in the world frame or in a frame that is connected to the AbsoluteSensor object via connector frame_resolve.
- New model "MultiBody.**Sensors.Distance**"
  to provide the distance between frame_a and frame_b as output signal. Derivatives of the distance can be easily obtained by connecting block Modelica.Blocks.Continuous.Der to the outPort.
- New models "MultiBody.**Sensors.Force/.Torque/.ForceAndTorque**"
  to provide cut forces and cut torques between two frames as output signals resolved in the local frame, in the world frame or in a frame that is connected to the objects via connector frame_resolve.
- Improved model "MultiBody.**Sensors.RelativeSensor**"
  A "frame_resolve" connector has been added, in order that the measured kinematic quantities can be resolved in the frame that is connected to frame_resolve.
- New connector "MultiBody.Interfaces.**Frame_resolve**"
  that should be used, if a frame provides solely orientation information. This connector is identical to the connectors "Frame", Frame_a" and "Frame_b". Only the icon is different: The border line is dotted with default line width and not with double default line width as for the other frames. When drawing a connection, the connection line is usually by default using the line style of the connector. As a result, the connection line will be a thin, dotted line. Frame_resolve is now used in all components where the orientation of another frame is needed (e.g., in Modelica.Mechanics.MultiBody.Forces.Force to define in which frame the force is provided).
- New function "MultiBody.Frames.**AngularVelocity1**"
  to compute the angular velocity of frame 2 with respect to frame 1 resolved in frame 1 (previously, only function AngularVelocity2 was provided to resolve the angular velocity in frame 2. Transforming this vector in to frame 1 is less efficient as directly computing the angular velocity resolved in frame 1).
- New conversion script "MultiBody\Scripts**ConvertOldToNewMultiBody.mos**"
  to convert models from the "old" ModelicaAdditions.MultiBody library to the "new" MultiBody" library. For details, see Modelica.Mechanics.MultiBody.UsersGuide.Upgrade.

Version 0.98, 2003-10-27

This version is **not backward** compatible to the previous MultiBody versions, since the initialization has changed. Parameter "startValuesFixed" is no longer present and is replaced by parameter "initType" (see below). Models generated with previous MultiBody versions are automatically converted to the new release.

- **Tutorial** improved.
- New model "MultiBody.Joints.Assemblies.**JointSSP**"
  Spherical - spherical - prismatic joint aggregation with mass (no constraints, no potential states) for analytic loop handling.
- New model "MultiBody.Forces.**LineForceWithTwoMasses**"
  General line force component with two optional point masses on the connection line.
- New model "MultiBody.Forces.**Force**"
  Force acting between two frames, defined by 3 input signals and resolved in frame_b or in frame_resolve.
- New model "MultiBody.Forces.**Torque**"
  Torque acting between two frames, defined by 3 input signals and resolved in frame_b or in frame_resolve
- New model "MultiBody.Forces.**ForceAndTorque**"
  Force and torque acting between two frames, defined by 6 input signals and resolved in frame_b or in frame_resolve.
- New model "MultiBody.**Examples.ForceAndTorque**"
  Demonstrate usage of ForceAndTorque element.
- New model "MultiBody.**Examples.LineForceWithTwoMasses**"
  Demonstrate line force with two point masses using a JointUPS and alternatively a LineForceWithTwoMasses component.
- New model "MultiBody.**Examples.InitSpringConstant**"
  Determine spring constant such that system is in steady state at given position.
- Removed model "MultiBody.Examples.ForceWithMass"
  This demo is included in the new LineForceWithTwoMasses example.

- Change in "MultiBody.Parts.**Body**"
  The default of parameter "useQuaternions" in the "Advanced" menu is changed from false to true. This means that by default quaternions are used as body states.
- Change in all models that have potential states:
  New initialization introduced replacing the previous "startValuesFixed" parameter. The new parameter "initType" can have the following values:

```
 initType = Modelica.Mechanics.MultiBody.Types.Init
  .Free                          : no initialization
                                   (= same as previous startValuesFixed =
false)
  .PositionVelocity              : initialize generalized position and
velocity variables
                                   (= same as previous startValuesFixed =
true)
  .SteadyState                   : initialize in steady state
                                   (velocity and acceleration are zero)
  .Position                      : initialize only generalized position
variable(s)
  .Velocity                      : initialize only generalized velocity
variable(s)
  .VelocityAcceleration          : initialize generalized velocity and
                                   acceleration variables
  .PositionVelocityAcceleration: initialize generalized position, velocity
                                   and acceleration variables
```

Version 0.97, 2003-09-10

Bug fixed in model "MultiBody.Parts.Body": There was an error when switching from one set of Cardan angles to another one when the actual Cardan angles are close to their singularity. This has been corrected.

Version 0.96, 2003-08-04

This was the first version delivered with Dymola.

Modelica.Mechanics.MultiBody.UsersGuide.Literature

Literature

- Technical details of this library are described in the 20 page paper:

  Otter M., Elmqvist H., and Mattsson S.E.:
  The New Modelica MultiBody Library. Modelica 2003 Conference, Linköping, Sweden, pp. 311-330, Nov. 3-4, 2003. Download from:
  http://www.modelica.org/Conference2003/papers/h37_Otter_multibody.pdf
- The method how to describe drive trains with 1-dimensional mechanics and to mount them on 3-dimensional components without neglecting dynamical effects is described in:

  Schweiger C., and Otter M.:
  Modelling 3-dim. Mechanical Effects of 1-dim. Powertrains. Modelica 2003 Conference, Linköping, Sweden, pp. 149-158, Nov. 3-4, 2003. Download from:
  http://www.modelica.org/Conference2003/papers/h06_Schweiger_powertrains_v5.pdf
- The method to solve a certain class of kinematic loops analytically is based on:

  Woernle C.:
  Ein systematisches Verfahren zur Aufstellung der geometrischen Schliessbedingungen in kinematischen Schleifen mit Anwendung bei der Rückwärtstransformation für Industrieroboter.

Fortschritt-Berichte VDI, Reihe 18, Nr. 59, Duesseldorf: VDI-Verlag 1988, ISBN 3-18-145918-6.

Hiller M., and Woernle C.:**A Systematic Approach for Solving the Inverse Kinematic Problem of Robot Manipulators**.
Proceedings 7th World Congress Th. Mach. Mech., Sevilla 1987.

Modelica.Mechanics.MultiBody.UsersGuide.Contact

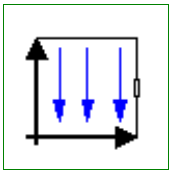
Contact

Main Author:
Martin Otter
Deutsches Zentrum für Luft und Raumfahrt e.V. (DLR)
Institut für Robotik und Mechatronik
Abteilung für Entwurfsorientierte Regelungstechnik
Postfach 1116
D-82230 Wessling
Germany
email: Martin.Otter@dlr.de

Acknowledgements:

- The central idea to handle a certain class of overdetermined, consistent set of differential algebraic equations (i.e., there are more equations than unknowns) with symbolic transformation algorithms was developed together with Hilding Elmqvist and Sven Erik Mattsson from Dynasim AB, Lund, Sweden. The MultiBody library is heavily relying on this feature which is a prerequiste for a truely "object-oriented" multi-body systems library, where components can be connected together in any meaningful way.
- The Examples.Loops.EngineV6 demo of a six cylinder V6 engine with 6 planar loops and 1 degree of freedom is from Hilding Elmqvist and Sven Erik Mattsson.
- Modelica.Mechanics.MultiBody.Forces.LineForceWithMass is based on model "RelativeDistance" from the Modelica VehicleDynamics library of Johan Andreasson from Royal Institute of Technology, Stockholm, Sweden.
- The 1-dim. components (Parts.Rotor1D, Parts.BevelGear1D, Mounting1D) and Joints.GearConstraints are from Christian Schweiger.
- The design of this library is based on work carried out in the EU RealSim project (Real-time Simulation for Design of Multi-physics Systems) funded by the European Commission within the Information Societies Technology (IST) programme under contract number IST 1999-11979.

Modelica.Mechanics.MultiBody.World



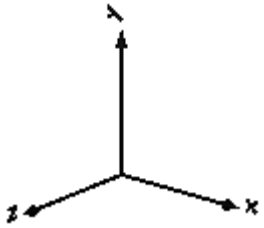
World coordinate system + gravity field + default animation definition

Information

Model **World** represents a global coordinate system fixed in ground. This model serves several purposes:

- It is used as **inertial system** in which the equations of all elements of the MultiBody library are defined.
- It is the world frame of an **animation window** in which all elements of the MultiBody library are visualized.
- It is used to define the **gravity field** in which a multi-body model is present. Default is a uniform gravity field where the gravity acceleration vector g is the same at every position. Additionally, a point gravity field can be selected.

- It is used to define **default settings** of animation properties (e.g. the diameter of a sphere representing by default the center of mass of a body, or the diameters of the cylinders representing a revolute joint).
- It is used to define a **visual representation** of the world model (= 3 coordinate axes with labels) and of the defined gravity field.



Since the gravity field function is required from all bodies with mass and the default settings of animation properties are required from nearly every component, exactly one instance of model World needs to be present in every model on the top level. The basic declaration needs to be:

```
inner Modelica.Mechanics.MultiBody.World world
```

Note, it must be an **inner** declaration with instance name **world** in order that this world object can be accessed from all objects in the model. When dragging the "World" object from the package browser into the diagram layer, this declaration is automatically generated (this is defined via annotations in model World).

All vectors and tensors of a mechanical system are resolved in a frame that is local to the corresponding component. Usually, if all relative joint coordinates vanish, the local frames of all components are parallel to each other, as well as to the world frame (this holds as long as a Parts.FixedRotation, component is **not** used). In this "reference configuration" it is therefore alternatively possible to resolve all vectors in the world frame, since all frames are parallel to each other. This is often very convenient. In order to give some visual support in such a situation, in the icon of a World instance two axes of the world frame are shown and the labels of these axes can be set via parameters.

Parameters

Type	Name	Default	Description
Boolean	enableAnimation	true	= true, if animation of all components is enabled
Boolean	animateWorld	true	= true, if world coordinate system shall be visualized
Boolean	animateGravity	true	= true, if gravity field shall be visualized (acceleration vector or field center)
AxisLabel	label1	"x"	Label of horizontal axis in icon
AxisLabel	label2	"y"	Label of vertical axis in icon
Temp	gravityType	GravityTypes.UniformGravity	Type of gravity field
Acceleration	g	9.81	Constant gravity acceleration [m/s2]
Axis	n	{0,-1,0}	Direction of gravity resolved in world frame (gravity = g*n/length(n))
Real	mue	3.986e14	Gravity field constant (default = field constant of earth) [m3/s2]
Boolean	driveTrainMechanics3D	false	= true, if 3-dim. mechanical effects of

			Parts.Mounting1D/Rotor1D/BevelGear1D shall be taken into account
Animation			
if animateWorld = true			
Distance	axisLength	nominalLength/2	Length of world axes arrows [m]
Distance	axisDiameter	axisLength/defaultFrameDiame...	Diameter of world axes arrows [m]
Boolean	axisShowLabels	true	= true, if labels shall be shown
Color	axisColor_x	Modelica.Mechanics.MultiBody...	Color of x-arrow
Color	axisColor_y	axisColor_x	
Color	axisColor_z	axisColor_x	Color of z-arrow
if animateGravity = true and gravityType = UniformGravity			
Position	gravityArrowTail[3]	{0,0,0}	Position vector from origin of world frame to arrow tail, resolved in world frame [m]
Length	gravityArrowLength	axisLength/2	Length of gravity arrow [m]
Diameter	gravityArrowDiameter	gravityArrowLength/defaultWi...	Diameter of gravity arrow [m]
Color	gravityArrowColor	{0,230,0}	Color of gravity arrow
if animateGravity = true and gravityType = PointGravity			
Diameter	gravitySphereDiameter	12742000	Diameter of sphere representing gravity center (default = mean diameter of earth) [m]
Color	gravitySphereColor	{0,230,0}	Color of gravity sphere
Defaults			
Length	nominalLength	1	"Nominal" length of multi-body system [m]
Length	defaultAxisLength	nominalLength/5	Default for length of a frame axis (but not world frame) [m]
Length	defaultJointLength	nominalLength/10	Default for the fixed length of a shape representing a joint [m]
Length	defaultJointWidth	nominalLength/20	Default for the fixed width of a shape representing a joint [m]
Length	defaultForceLength	nominalLength/10	Default for the fixed length of a shape representing a force (e.g. damper) [m]
Length	defaultForceWidth	nominalLength/20	Default for the fixed width of a shape represening a force (e.g. spring, bushing) [m]
Length	defaultBodyDiameter	nominalLength/9	Default for diameter of sphere representing the center of mass of a body [m]
Real	defaultWidthFraction	20	Default for shape width as a fraction of shape length (e.g., for Parts.FixedTranslation)
Length	defaultArrowDiameter	nominalLength/40	Default for arrow diameter

			(e.g., of forces, torques, sensors) [m]
Real	defaultFrameDiameterFraction	40	Default for arrow diameter of a coordinate system as a fraction of axis length
Real	defaultSpecularCoefficient	0.7	Default reflection of ambient light (= 0: light is completely absorbed)
Real	defaultN_to_m	1000	Default scaling of force arrows (length = force/defaultN_to_m) [N/m]
Real	defaultNm_to_m	1000	Default scaling of torque arrows (length = torque/defaultNm_to_m) [N.m/m]

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed in the origin of the world frame

---

Modelica.Mechanics.MultiBody.Examples

Examples that demonstrate the usage of the MultiBody library

Information

This package contains example models to demonstrate the usage of the MultiBody package. Open the models and simulate them according to the provided description in the models.

Package Content

Name	Description
Elementary	Elementary examples to demonstrate various features of the MultiBody library
Loops	Examples with kinematic loops
Systems	Examples of complete system models including 3-dimensional mechanics

---

Modelica.Mechanics.MultiBody.Examples.Elementary

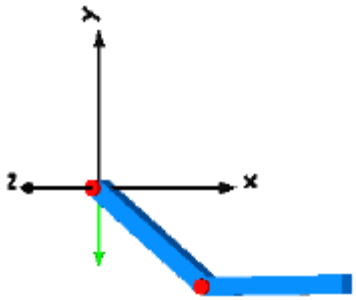
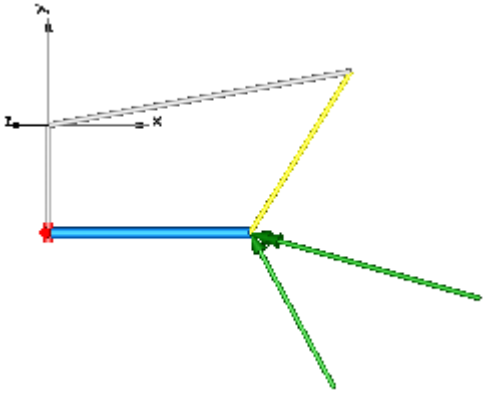
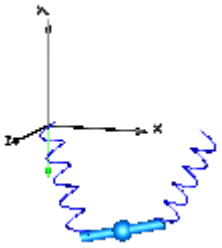
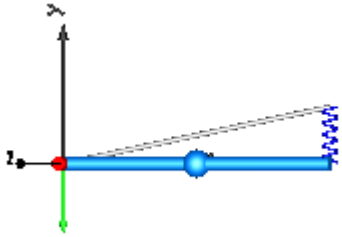
Elementary examples to demonstrate various features of the MultiBody library

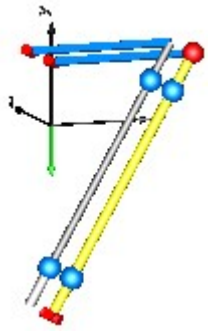
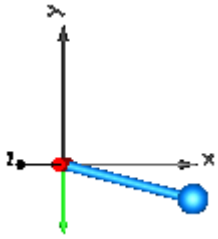
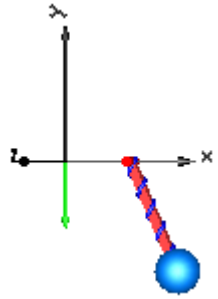
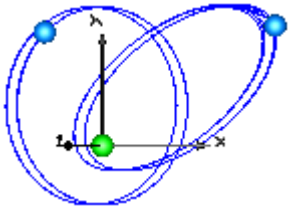
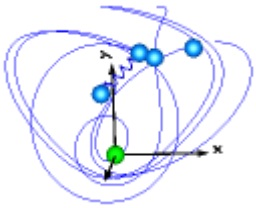
Information

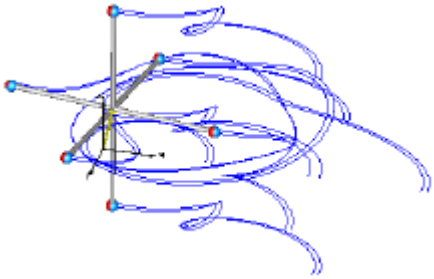
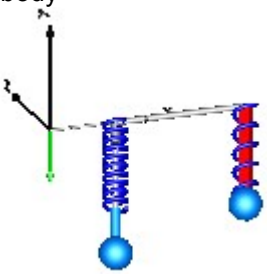
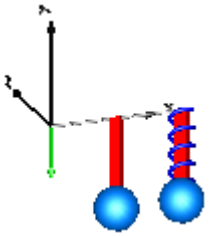
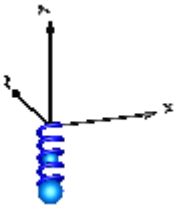
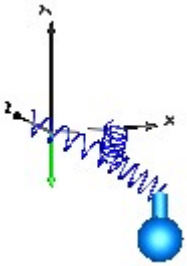
This package contains elementary example models to demonstrate the usage of the MultiBody library

Content

Model	*Description*
DoublePendulum	Simple double pendulum with two revolute joints and two bodies.

	
ForceAndTorque	Demonstrates usage of Forces.ForceAndTorque element. 
FreeBody	Free flying body attached by two springs to environment. 
InitSpringConstant	Determine spring constant such that system is in steady state at given position. 
LineForceWithTwoMasses	Demonstrates a line force with two point masses using a Joints.Assemblies.JointUPS and alternatively a Forces.LineForceWithTwoMasses component.

	
Pendulum	Simple pendulum with one revolute joint and one body. 
PendulumWithSpringDamper	Simple spring/damper/mass system 
PointGravity	Two bodies in a point gravity field 
PointGravityWithPointMasses	Two point masses in a point gravity field (rotation of bodies is neglected) 
PointGravityWithPointMasses2	Rigidly connected point masses in a point gravity field

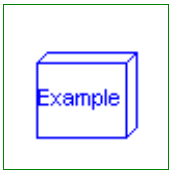
	
SpringDamperSystem	Spring/damper system with a prismatic joint and attached on free flying body 
SpringMassSystem	Mass attached via a prismatic joint and a spring to the world frame 
SpringWithMass	Point mass hanging on a spring 
ThreeSprings	3-dimensional springs in series and parallel connection 

Package Content

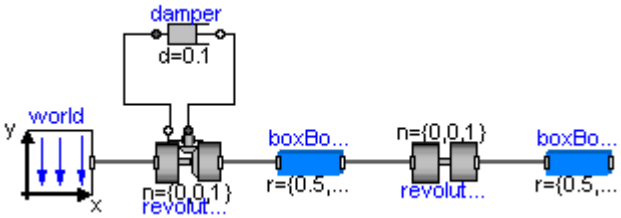
Name	Description
DoublePendulum	Simple double pendulum with two revolute joints and two bodies
ForceAndTorque	Demonstrate usage of ForceAndTorque element
FreeBody	Free flying body attached by two springs to environment

InitSpringConstant	Determine spring constant such that system is in steady state at given position
LineForceWithTwoMasses	Demonstrate line force with two point masses using a JointUPS and alternatively a LineForceWithTwoMasses component
Pendulum	Simple pendulum with one revolute joint and one body
PendulumWithSpringDamper	Simple spring/damper/mass system
PointGravity	Two point masses in a point gravity field
PointGravityWithPointMasses	Two point masses in a point gravity field (rotation of bodies is neglected)
PointGravityWithPointMasses2	Rigidly connected point masses in a point gravity field
SpringDamperSystem	Simple spring/damper/mass system
SpringMassSystem	Mass attached with a spring to the world frame
SpringWithMass	Point mass hanging on a spring
ThreeSprings	3-dim. springs in series and parallel connection

Modelica.Mechanics.MultiBody.Examples.Elementary.DoublePendulum

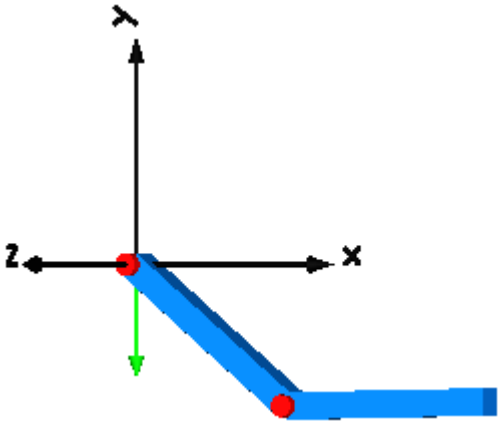


Simple double pendulum with two revolute joints and two bodies

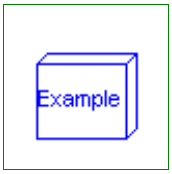


Information

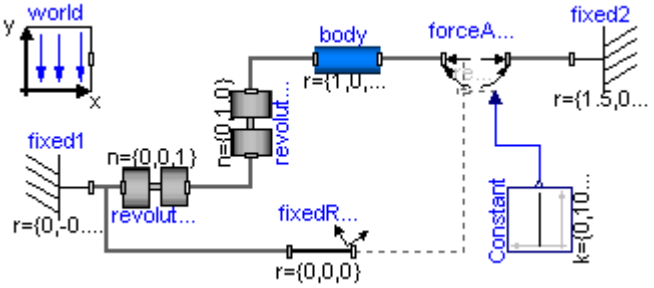
This example demonstrates that by using joint and body elements animation is automatically available. Also the revolute joints are animated. Note, that animation of every component can be switched of by setting the first parameter **animation** to **false** or by setting **enableAnimation** in the **world** object to **false** to switch off animation of all components.



Modelica.Mechanics.MultiBody.Examples.Elementary.ForceAndTorque



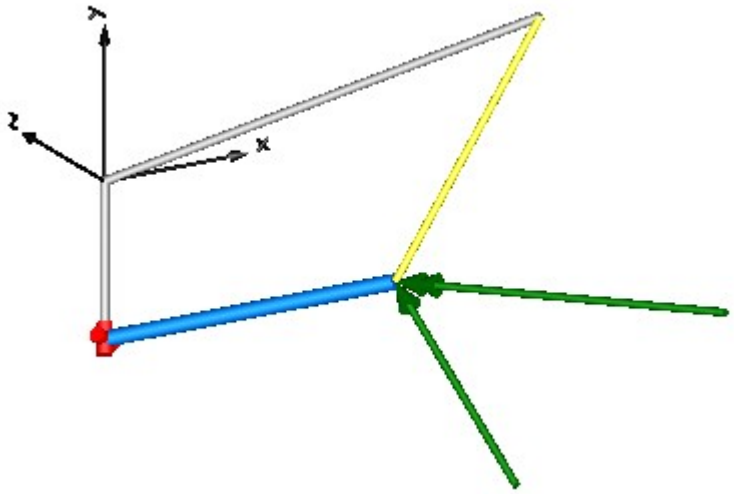
Demonstrate usage of ForceAndTorque element



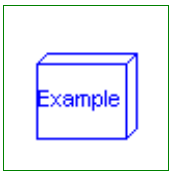
Information

In this example the usage of the general force element "ForceAndTorque" is shown. A "ForceAndTorque" element is connected between a body and a fixed point in the world system. The force and torque is defined by the "Constant" block. The two vectors are resolved in the coordinate system defined by the "fixedRotation" component that is fixed in the world system:

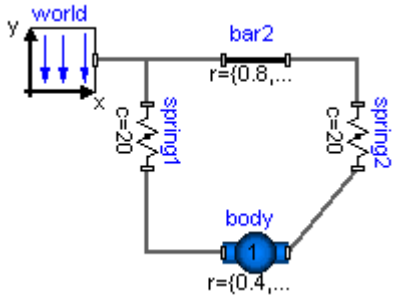
The animation view at time = 0 is shown in the figure below. The yellow line is directed from frame_a to frame_b of the forceAndTorque component. The green arrow characterizes the force acting at the body whereas the green double arrow characterizes the torque acting at the body. The lengths of the two vectors are proportional to the lengths of the force and torque vectors (constant scaling factors are defined as parameters in the forceAndTorque component):



Modelica.Mechanics.MultiBody.Examples.Elementary.FreeBody



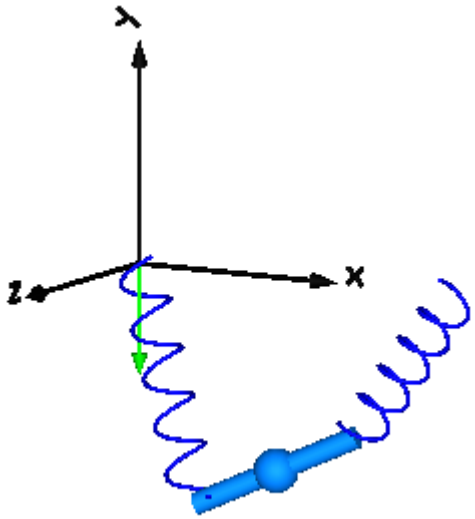
Free flying body attached by two springs to environment



Information

This example demonstrates:

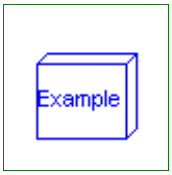
- The animation of spring and damper components
- A body can be freely moving without any connection to a joint. In this case body coordinates are used automatically as states (whenever joints are present, it is first tried to use the generalized coordinates of the joints as states).
- If a body is freely moving, the initial position and velocity of the body can be defined with the "Initialization" menu as shown with the body "body1" in the left part (click on "Initialization").



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled

Modelica.Mechanics.MultiBody.Examples.Elementary.InitSpringConstant



Determine spring constant such that system is in steady state at given position

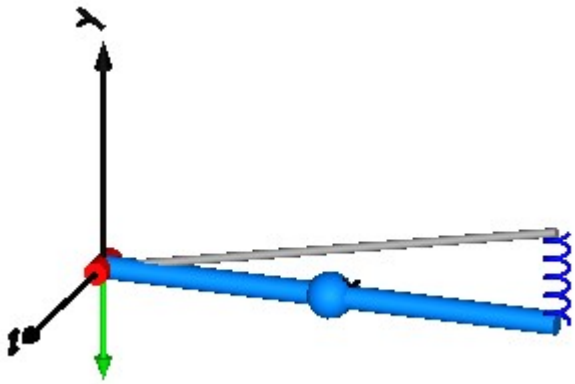
Information

This example demonstrates a non-standard type of initialization by calculating a spring constant such that a simple pendulum is at a defined position in steady state.

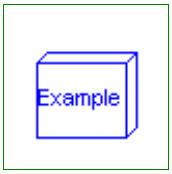
The goal is that the pendulum should be in steady state when the rotation angle of the pendulum is zero. The spring constant of the spring shall be calculated during initialization such that this goal is reached.

The pendulum has one degree of freedom, i.e., two states. Therefore, two additional equations have to be provided for initialization. However, parameter "c" of the spring component is defined with attribute "fixed = **false**", i.e., the value of this parameter is computed during initialization. Therefore, there is one additional equation required during initialization. The 3 initial equations are the rotational angle of the revolute joint and its first and second derivative. The latter one are zero, in order to initialize in steady state. By setting parameter initType of the revolute joint "rev" to "MultiBody.Types.Init.PositionVelocityAcceleration", the required 3 initial equations are defined.

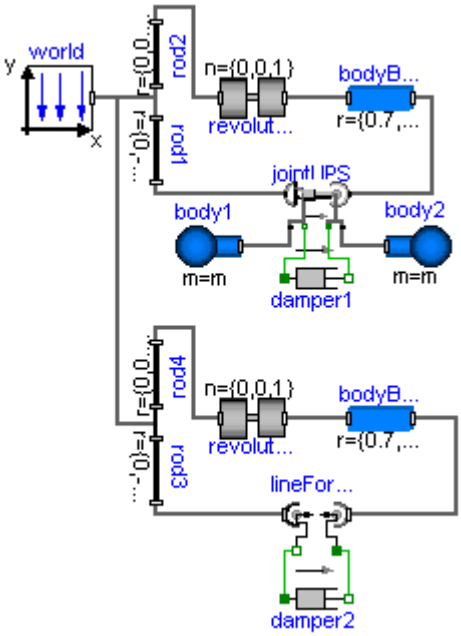
After translation, this model is initialized in steady-state. The spring constant is computed as c = 49.05 N/m. An animation of this simulation is shown in the figure below.



Modelica.Mechanics.MultiBody.Examples.Elementary.LineForceWithTwoMasses



Demonstrate line force with two point masses using a JointUPS and alternatively a LineForceWithTwoMasses component



Information

It is demonstrated how to implement line force components that shall have mass properties. Two alternative implementations are given:

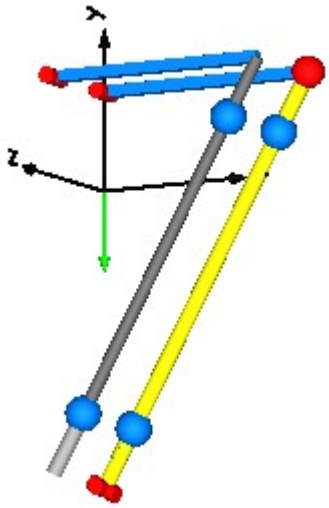
- With JointUPS:

Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUPS is an aggregation of a universal, a prismatic and a spherical joint that approximates a real force component, such as a hydraulic cylinder. At the two frames of the prismatic joint (frame_ia, frame_ib of jointUPS) two bodies are attached. The parameters are selected such that the center of masses of the two bodies are located on the line connecting frame_a and frame_b of the jointUPS component. Both bodies have the same mass and the inertia tensor is set to zero, i.e., the two bodies are treated as point masses.

- With LineForceWithTwoMasses: Modelica.Mechanics.MultiBody.Forces.LineForceWithTwoMasses is a line force component with the built-in property that two point masses are located on the line on which the line force is acting. The parameters are selected in such a way that the same system as with the jointUPS component is described.

In both cases, a linear 1-dimensional translational damper from the Modelica.Mechanics.Translational library is used as line force between the two attachment points. Simulate this system and plot the differences of the cut forces at both sides of the line force component ("rod_f_diff" and "body_f_diff"). Both vectors should be zero (depending on the choosen relative tolerance of the integration, the difference is in the order of 1.e-10 ... 1.e-15).

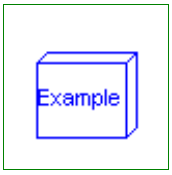
Note, that the implementation with the LineForceWithTwoMasses component is simpler and more convenient. An animation of this simulation is shown in the figure below. The system on the left side in the front is the animation with the LineForceWithTwoMasses component whereas the system on the right side in the back is the animation with the JointUPS component.



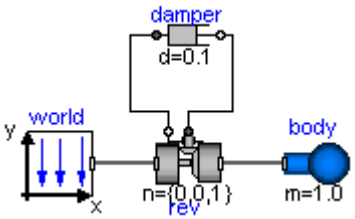
Parameters

Type	Name	Default	Description
Mass	m	1	Mass of point masses [kg]

Modelica.Mechanics.MultiBody.Examples.Elementary.Pendulum

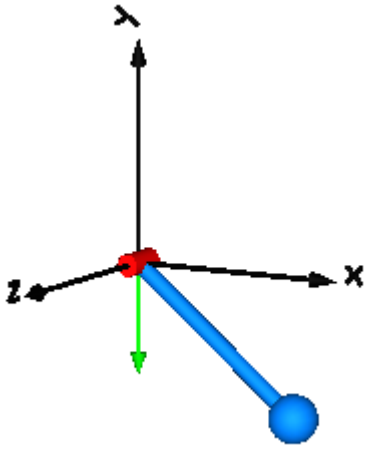


Simple pendulum with one revolute joint and one body



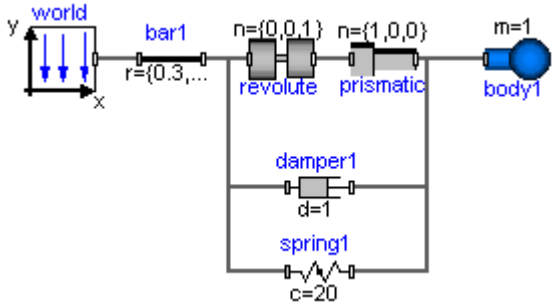
Information

This simple model demonstrates that by just dragging components default animation is defined that shows the structure of the assembled system.



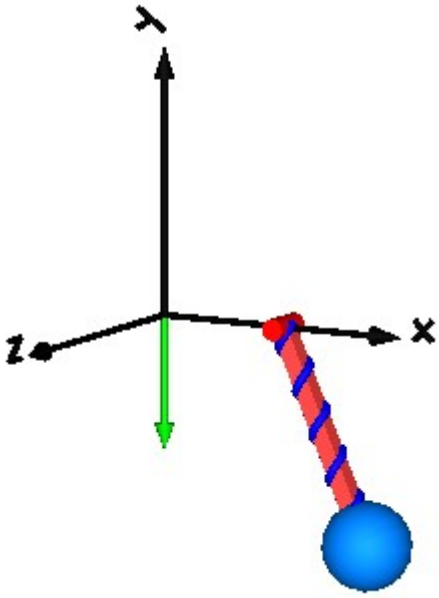
Modelica.Mechanics.MultiBody.Examples.Elementary.PendulumWithSpringDamper

Simple spring/damper/mass system



Information

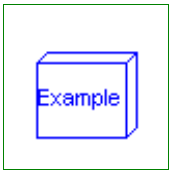
A body is attached on a revolute and prismatic joint. A 3-dim. spring and a 3-dim. damper are connected between the body and a point fixed in the world frame:



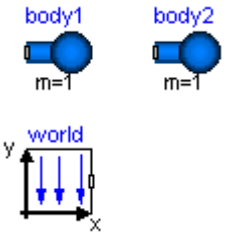
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled

Modelica.Mechanics.MultiBody.Examples.Elementary.PointGravity

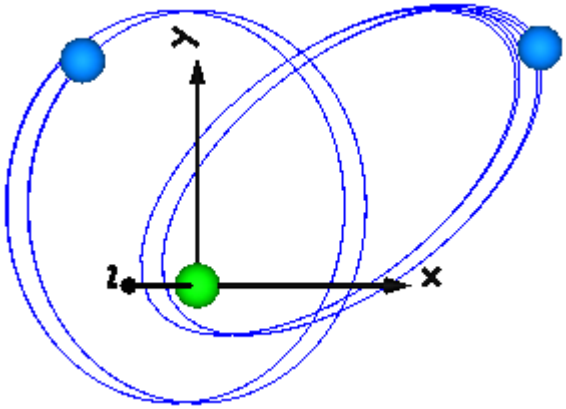


Two point masses in a point gravity field

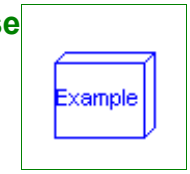


Information

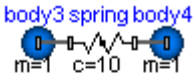
This model demonstrates a point gravity field. Two bodies are placed in the gravity field. The initial positions and velocities of these bodies are selected such that one body rotates on a circle and the other body rotates on an ellipse around the center of the point gravity field.

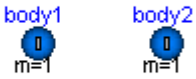


Modelica.Mechanics.MultiBody.Examples.Elementary.PointGravityWithPointMasses



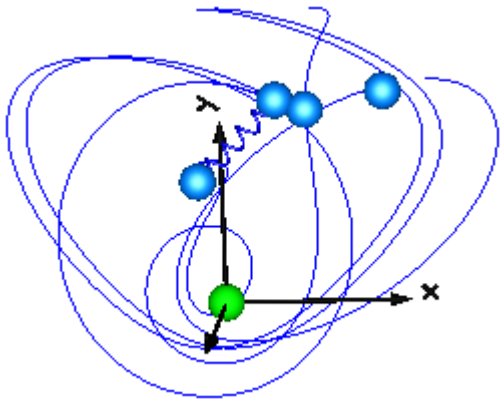
Two point masses in a point gravity field (rotation of bodies is neglected)



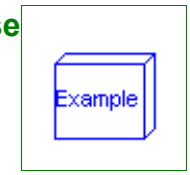


Information

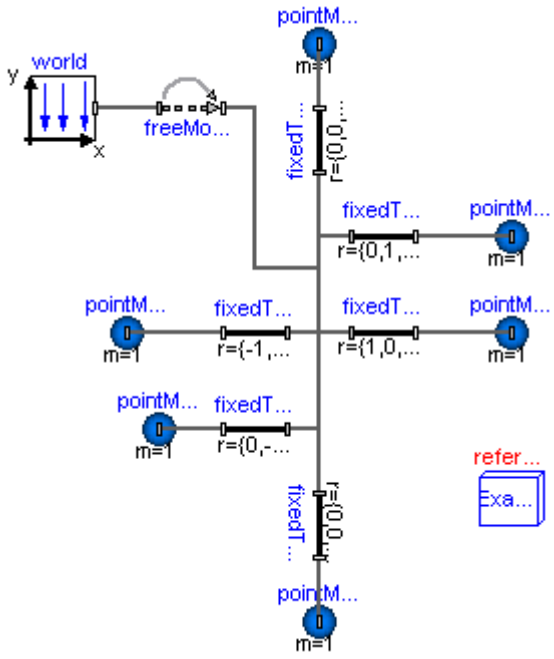
This model demonstrates the usage of model Parts.PointMass in a point gravity field. The PointMass model has the feature that that rotation is not taken into account and can therefore also not be calculated. This example demonstrates two cases where this does not matter: If a PointMass is not connected (body1, body2), the orientation object in these point masses is set to a unit rotation. If a PointMass is connected by a line force element, such as the used Forces.LineForceWithMass component, then the orientation object is set to a unit rotation within the line force element. These are the two cases where the rotation is automatically set to a default value, when the physical system does not provide the equations.



Modelica.Mechanics.MultiBody.Examples.Elementary.PointGravityWithPointMasses2



Rigidly connected point masses in a point gravity field

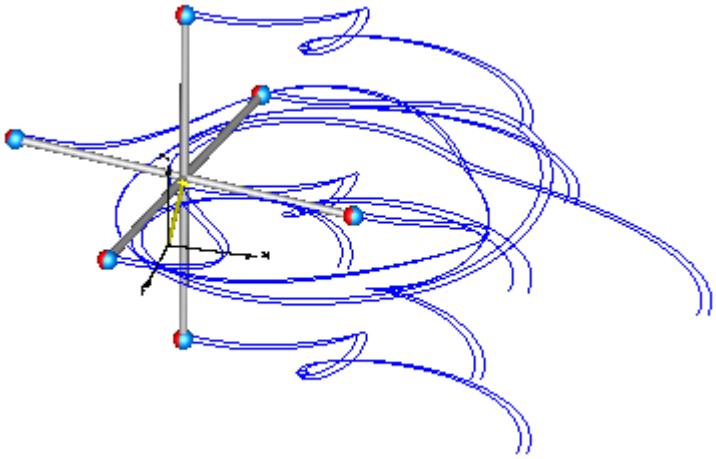


Information

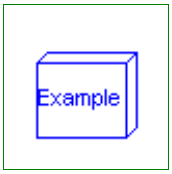
This model demonstrates the usage of model Parts.PointMass in a point gravity field. 6 point masses are connected rigidly together. Translating such a model results in an error, because point masses do not define an orientation object. The example demonstrates that in such a case (when the orientation object is not defined by an object that is connected to a point mass), a "MultiBody.Joints.FreeMotion" joint has to be used, to define the the degrees of freedom of this structure.

In order to demonstrate that this approach is correct, in model "referenceSystem", the same system is again provided, but this time modeled with a generic body (Parts.Body) where the inertia tensor is set to zero. In this case, no FreeMotion object is needed because every body provides its absolute translational and rotational position and velocity as potential states.

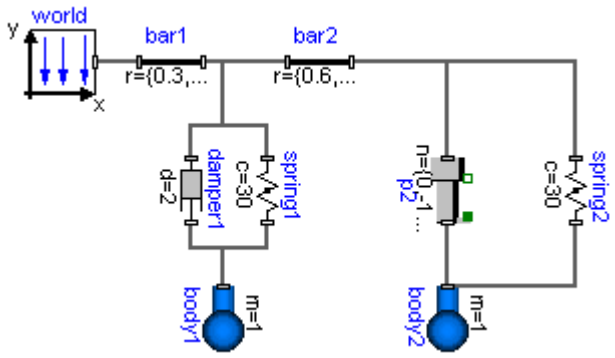
The two systems should move exactly in the same way. The system with the PointMasses object visulizes the point masses in "red", whereas the "referenceSystem" shows its bodies in "blue".



Modelica.Mechanics.MultiBody.Examples.Elementary.SpringDamperSystem



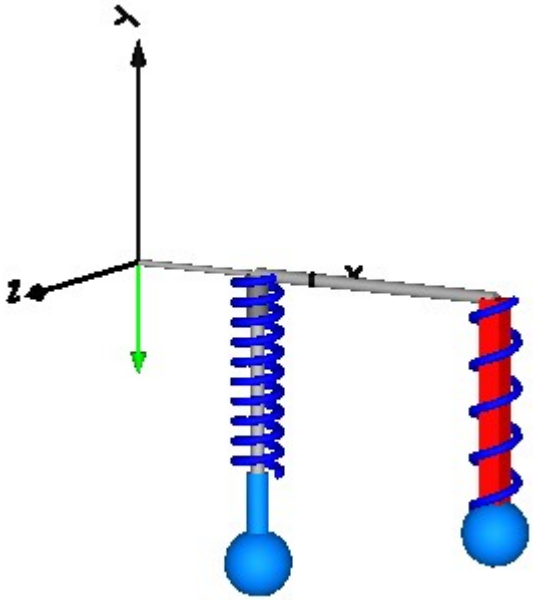
Simple spring/damper/mass system



Information

This example demonstrates:

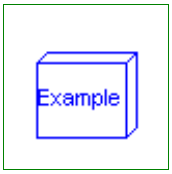
- The animation of spring and damper components
- A body can be freely moving without any connection to a joint. In this case body coordinates are used automatically as states (whenever joints are present, it is first tried to use the generalized coordinates of the joints as states).
- If a body is freely moving, the initial position and velocity of the body can be defined with the "Initialization" menu as shown with the body "body1" in the left part (click on "Initialization").



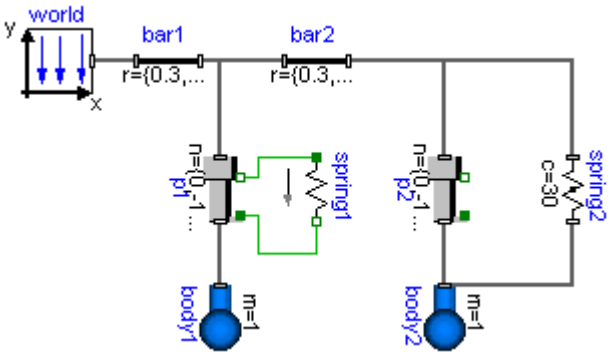
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled

Modelica.Mechanics.MultiBody.Examples.Elementary.SpringMassSystem



Mass attached with a spring to the world frame

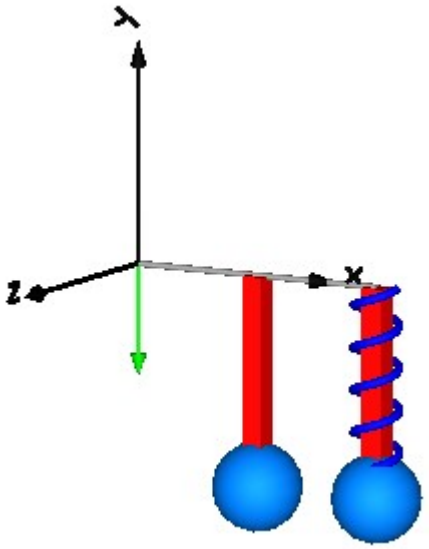


Information

This example shows the two different ways how force laws can be utilized:

- In the left system a body is attached via a prismatic joint to the world frame. The prismatic joint has two 1-dimensional translational flanges (called "bearing" and "axis") that allows to connect elements from the Modelica.Mechanics.Translational library between the bearing and the axis connector. The effect is that the force generated by the 1-dimensional elements acts as driving force in the axis of the prismatic joint. In the example a simple spring is used.
  The advantage of this approach is that the many elements from the Translational library can be easily used here and that this implementation is usually more efficient as when using 3-dimensional springs.
- In the right system the same model is defined. The difference is that a 3-dimensional spring from the

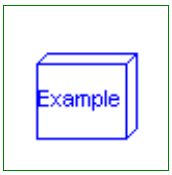
Modelica.Mechanics.MultiBody.Forces library is used. This has the advantage to get a nice animation of the force component.



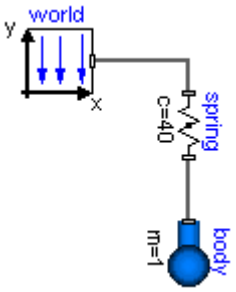
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled

Modelica.Mechanics.MultiBody.Examples.Elementary.SpringWithMass

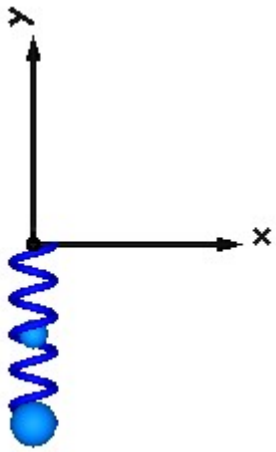


Point mass hanging on a spring

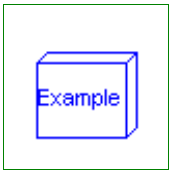


Information

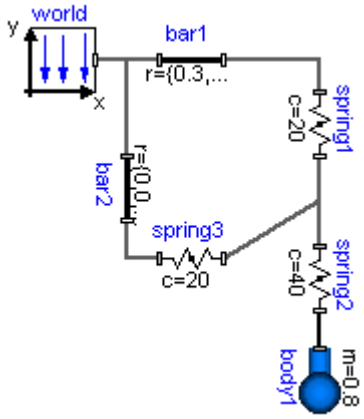
This example shows that a force component may have a mass. The 3-dimensional spring as used in this example, has an optional point mass between the two points where the spring is attached. In the animation, this point mass is represented by a small, light blue, sphere.



Modelica.Mechanics.MultiBody.Examples.Elementary.ThreeSprings

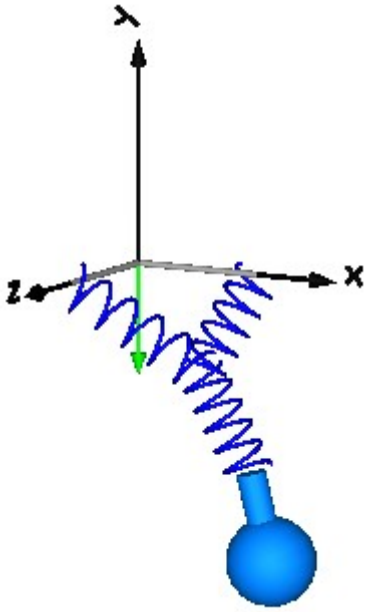


3-dim. springs in series and parallel connection



Information

This example demonstrates that **3-dimensional line force** elements (here: Modelica.Mechanics.MultiBody.Forces.Spring elements) can be connected together in **series** without having a body with mass at the connection point (as usually required by multi-body programs). This is advantageous since stiff systems can be avoided, say, due to a stiff spring and a small mass at the connection point.



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled

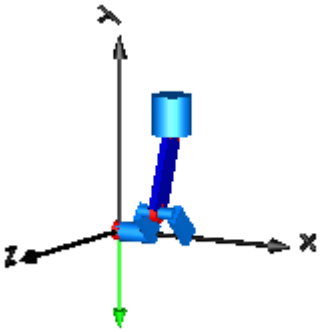
Modelica.Mechanics.MultiBody.Examples.Loops

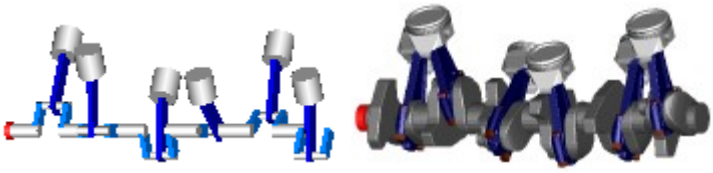
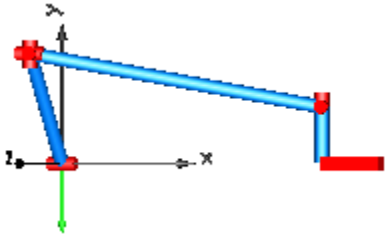
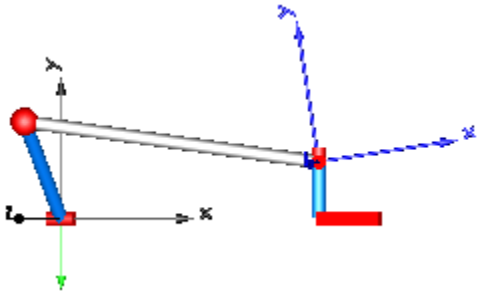
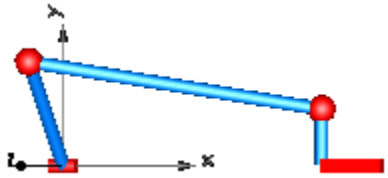
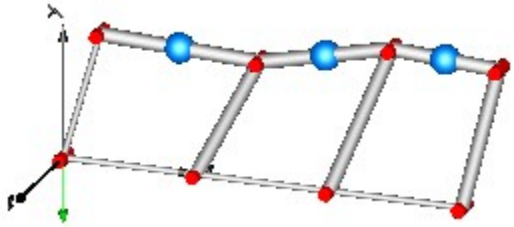
Examples with kinematic loops

Information

This package contains different examples to show how mechanical systems with kinematic loops can be modeled.

Content

Model	*Description*
Engine1a Engine1b Engine1b_analytic	Model of one cylinder engine (Engine1a: simple, without combustion; Engine1b: with combustion; Engine1b_analytic: same as Engine1b but analytic loop handling) 
EngineV6 EngineV6_analytic	V6 engine with 6 cylinders, 6 planar loops and 1 degree-of-freedom. Second version with analytic handling of kinematic loops and CAD data animation.

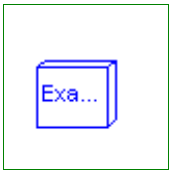
	
Fourbar1	One kinematic loop with four bars (with only revolute joints; 5 non-linear equations) 
Fourbar2	One kinematic loop with four bars (with UniversalSpherical joint; 1 non-linear equation) 
Fourbar_analytic	One kinematic loop with four bars (with JointSSP joint; analytic solution of non-linear algebraic loop) 
PlanarLoops_analytic	Mechanism with three planar kinematic loops and one degree-of-freedom with analytic loop handling (with JointRRR joints) 

Package Content

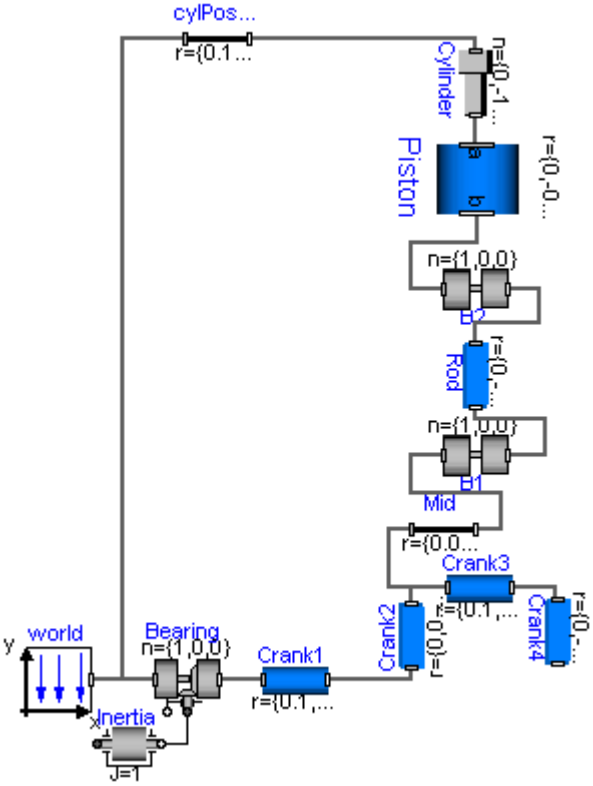
Name	Description
Engine1a	Model of one cylinder engine
Engine1b	Model of one cylinder engine with gas force and preparation for assembly joint JointRRP
Engine1b_analytic	Model of one cylinder engine with gas force and analytic loop handling
EngineV6	V6 engine with 6 cylinders, 6 planar loops and 1 degree-of-freedom
EngineV6_analytic	V6 engine with 6 cylinders, 6 planar loops, 1 degree-of-freedom and analytic

	handling of kinematic loops
Fourbar1	One kinematic loop with four bars (with only revolute joints; 5 non-linear equations)
Fourbar2	One kinematic loop with four bars (with UniversalSpherical joint; 1 non-linear equation)
Fourbar_analytic	One kinematic loop with four bars (with JointSSP joint; analytic solution of non-linear algebraic loop)
PlanarLoops_analytic	Mechanism with three planar kinematic loops and one degree-of-freedom with analytic loop handling (with JointRRR joints)
Utilities	Utility models for Examples.Loops

Modelica.Mechanics.MultiBody.Examples.Loops.Engine1a



Model of one cylinder engine

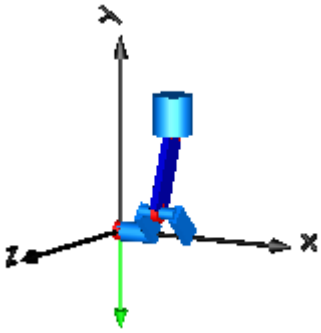


Information

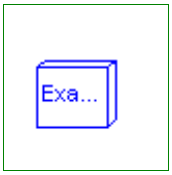
This is a model of the mechanical part of one cylinder of an engine. The combustion is not modelled. The "inertia" component at the lower left part is the output inertia of the engine driving the gearbox. The angular velocity of the output inertia has a start value of 10 rad/s in order to demonstrate the movement of the engine.

The engine is modeled solely by revolute and prismatic joints. Since this results in a **planar** loop there is the well known difficulty that the cut-forces perpendicular to the loop cannot be uniquely computed, as well as the cut-torques within the plane. This ambiguity is resolved by using the option **planarCutJoint** in the **Advanced** menu of one revolute joint in every planar loop (here: joint B1). This option sets the cut-force in direction of the axis of rotation, as well as the cut-torques perpendicular to the axis of rotation at this joint to zero and makes the problem mathematically well-formed.

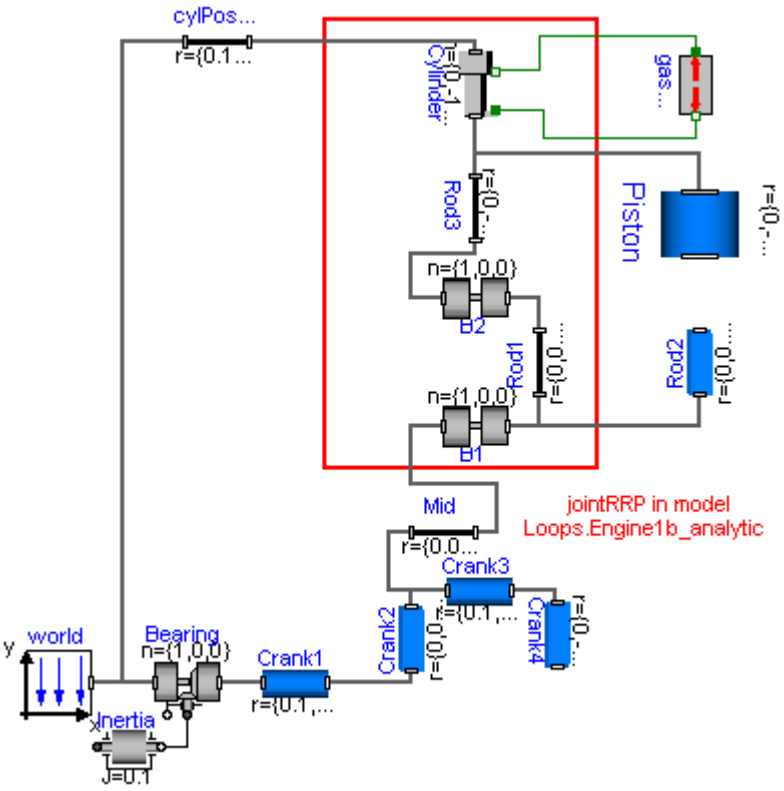
An animation of this example is shown in the figure below.



Modelica.Mechanics.MultiBody.Examples.Loops.Engine1b



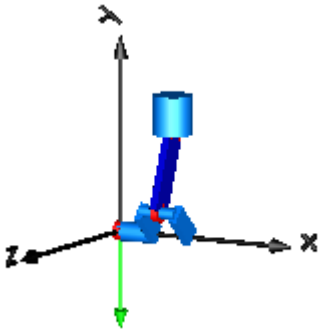
Model of one cylinder engine with gas force and preparation for assembly joint JointRRP



Information

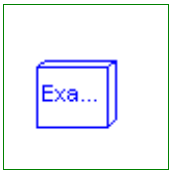
This is a model of the mechanical part of one cylinder of an engine. It is similiar to Loops.Engine1a. The difference is that a simple model for the gas force in the cylinder is added and that the model is restructured in such a way, that the central part of the planar kinematic loop can be easily replaced by the assembly joint "Modelica.Mechanics.MultiBody.Joints.Assemblies.**JointRRP**". This exchange of the kinematic loop is shown in Loops.Engine1b_analytic. The advantage of using JointRRP is, that the non-linear algebraic equation of this loop is solved analytically, and not numerically as in this model (Engine1b).

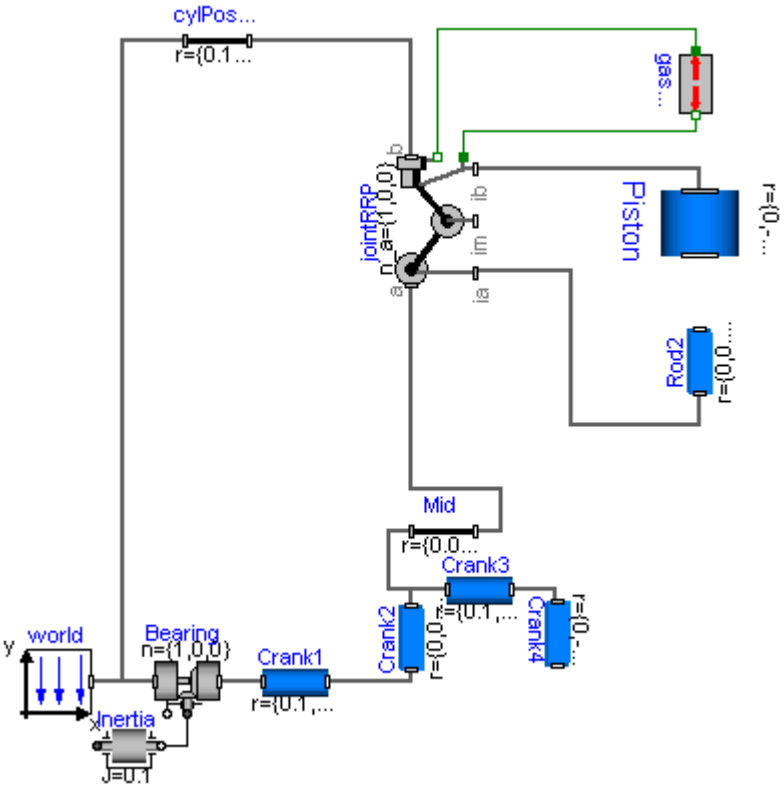
An animation of this example is shown in the figure below.



Modelica.Mechanics.MultiBody.Examples.Loops.Engine1b_analytic

Model of one cylinder engine with gas force and analytic loop handling

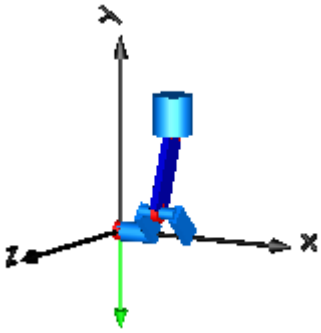




Information

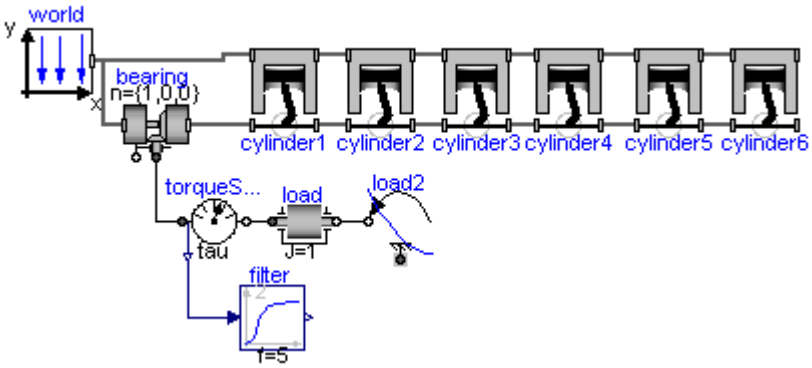
This is the same model as Loops.Engine1b. The only difference is that the central part of the planar kinematic loop has been replaced by the assembly joint "Modelica.Mechanics.MultiBody.Joints.Assemblies.**JointRRP**". The advantage of using JointRRP is, that the non-linear algebraic equation of this loop is solved analytically, and not numerically as in Loops.Engine1b.

An animation of this example is shown in the figure below.



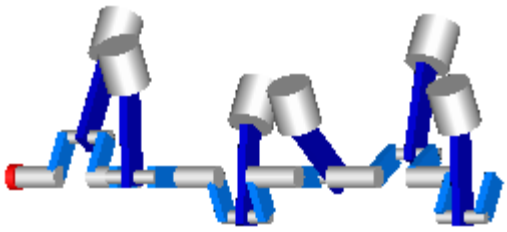
Modelica.Mechanics.MultiBody.Examples.Loops.EngineV6

V6 engine with 6 cylinders, 6 planar loops and 1 degree-of-freedom



Information

This is a V6 engine with 6 cylinders. It is hierarchically built up by using instances of one cylinder. For more details on the modeling of one cylinder, see example Engine1b. An animation of the engine is shown in the figure below.



Simulate for 5 s, and plot the variables **engineSpeed_rpm**, **engineTorque**, and **filteredEngineTorque**. Note, the result file has a size of about 50 Mbyte (for 5000 output intervalls).

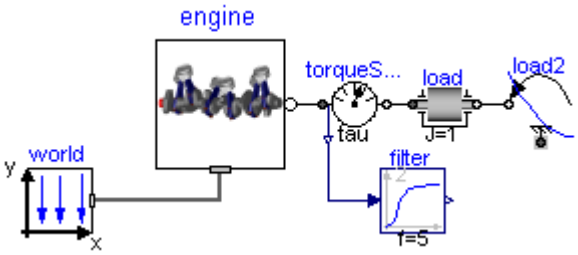
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled

Modelica.Mechanics.MultiBody.Examples.Loops.EngineV6_analytic

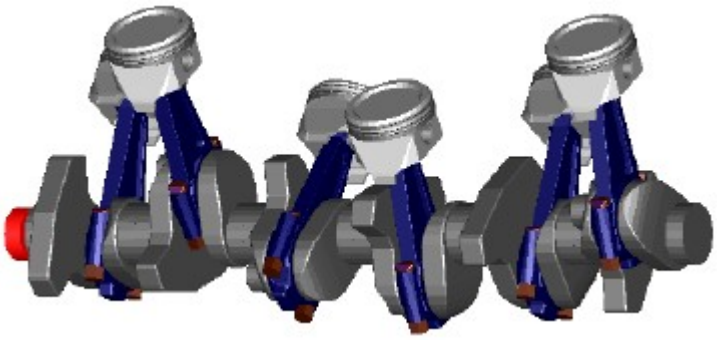
V6 engine with 6 cylinders, 6 planar loops, 1 degree-of-freedom and analytic handling of

kinematic loops



Information

This is a similar model as the example "EngineV6". However, the cylinders have been built up with component Modelica.Mechanics.MultiBody.Joints.Assemblies.JointRRR that solves the non-linear system of equations in an aggregation of 3 revolution joints **analytically** and only one body is used that holds the total mass of the crank shaft:



This model is about 20 times faster as the EngineV6 example and **no** linear or non-linear system of equations occur. In contrast, the "EngineV6" example leads to 6 systems of nonlinear equations (every system has dimension = 5, with Evaluate=false and dimension=1 with Evaluate=true) and a linear system of equations of about 40. This shows the power of the analytic loop handling.

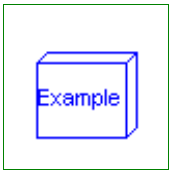
Simulate for 5 s, and plot the variables **engineSpeed_rpm**, **engineTorque**, and **filteredEngineTorque**. Note, the result file has a size of about 50 Mbyte (for 5000 output intervalls).

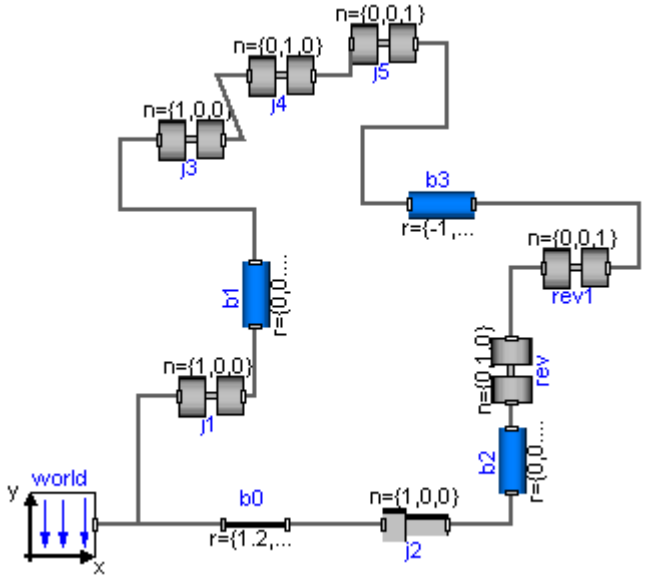
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled

Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar1

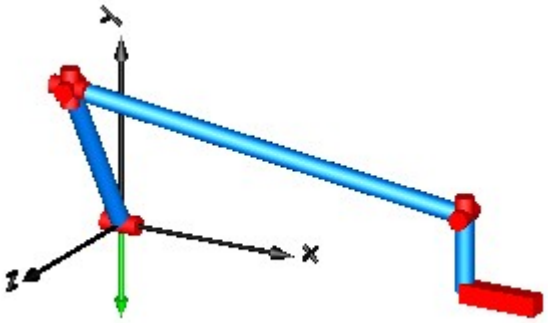
One kinematic loop with four bars (with only revolute joints; 5 non-linear equations)





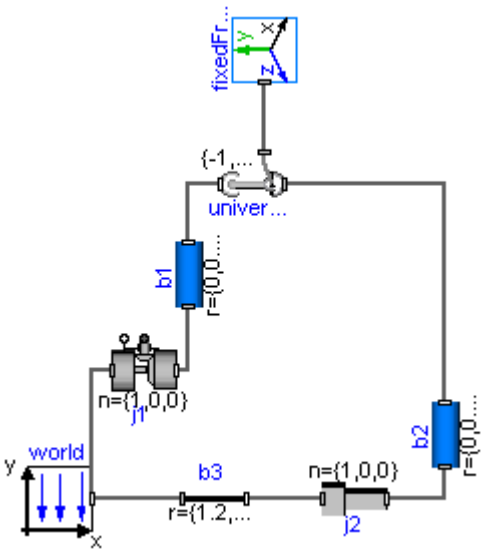
Information

This is a simple kinematic loop consisting of 6 revolute joints, 1 prismatic joint and 4 bars that is often used as basic constructing unit in mechanisms. This example demonstrates that usually no particular knowledge of the user is needed to handle kinematic loops. Just connect the joints and bodies together according to the real system. In particular **no** cut-joints or a spanning tree has to be determined. In this case, the initial condition of the angular velocity of revolute joint j1 is set to 300 deg/s in order to drive this loop.



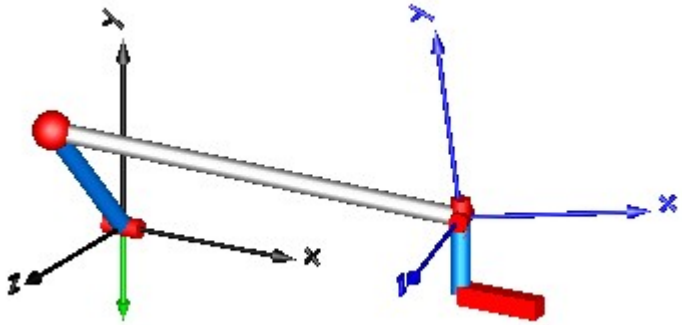
Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar2

One kinematic loop with four bars (with UniversalSpherical joint; 1 non-linear equation)



Information

This is a second version of the "four-bar" mechanism, see figure:



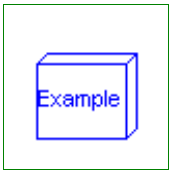
In this case the three revolute joints on the left top-side and the two revolute joints on the right top side have been replaced by the joint **UniversalSpherical** that is a rod connecting a spherical and a universal joint. This joint is defined by **1 constraint** stating that the distance between the two spherical joints is constant. Using this joint in a kinematic loop reduces the sizes of non-linear algebraic equations. For this loop, only one non-linear algebraic system of equations of order 1 remains.

At the UniversalSpherical joint an additional frame_ia fixed to the rod is present where components can be attached to the connecting rod. In this example just a coordinate system is attached to visualize frame_ia (coordinate system on the right in blue color).

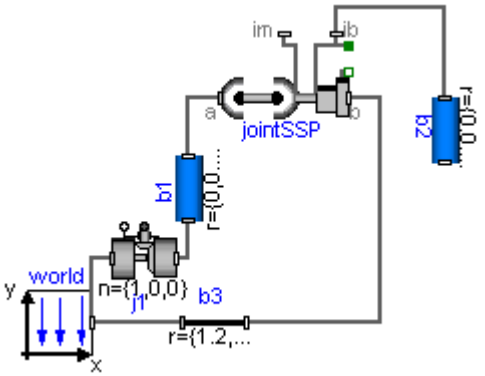
Another feature is that the length of the connecting rod can be automatically calculated during **initialization**. In order to do this, another initialization condition has to be given. In this example, the initial value of the distance of the prismatic joint j2 has been fixed (via the "Initialization" menu) and the rod length of joint "UniversalSpherical" is computed during initialization since parameter **computeLength** = **true** is set in the joint parameter menu. The main advantage is that during initialization no non-linear system of equation is solved and therefore initialization always works. To be precise, the following trivial non-linear equation is actually solved for rodLength:

```
rodLength*rodLength = f(angle of revolute joint, distance of prismatic joint)
```

Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar_analytic

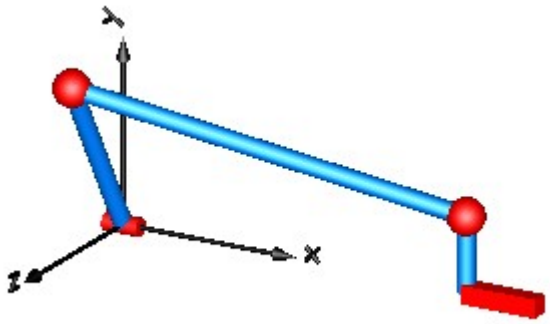


One kinematic loop with four bars (with JointSSP joint; analytic solution of non-linear algebraic loop)



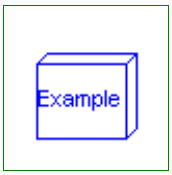
Information

This is a third version of the "four-bar" mechanism, see figure:

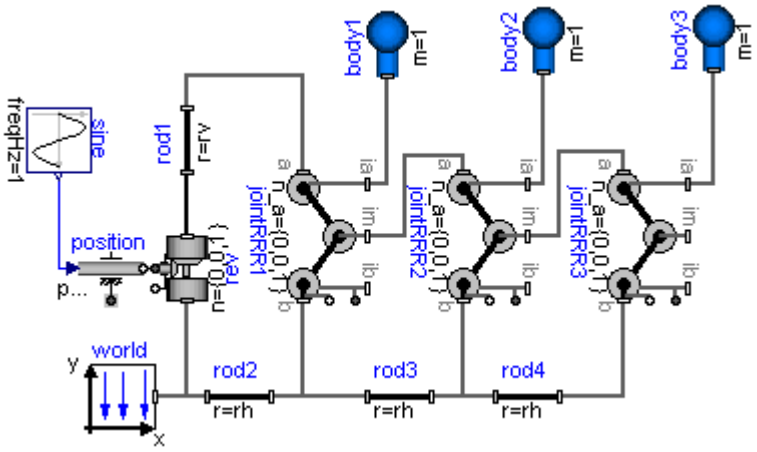


In this case the three revolute joints on the left top-side and the two revolute joints on the right top side have been replaced by the assembly joint **Joints.Assemblies.JointSSP** which consists of two spherical joints and one prismatic joint. Since JointSSP solves the non-linear constraint equation internally analytically, no non-linear equation appears any more and a Modelica translator, such as Dymola, can transform the system into state space form without solving a system of equations. For more details, see MultiBody.UsersGuide.Tutorial.LoopStructures.AnalyticLoopHandling.

Modelica.Mechanics.MultiBody.Examples.Loops.PlanarLoops_analytic

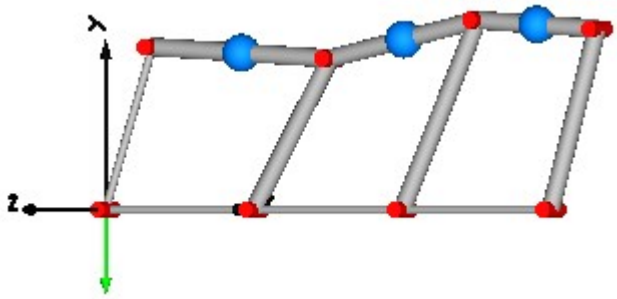


Mechanism with three planar kinematic loops and one degree-of-freedom with analytic loop handling (with JointRRR joints)



Information

It is demonstrated how the Modelica.Mechanics.MultiBody.Joints.Assemblies.JointRRR joint can be used to solve the non-linear equations of coupled planar loops analytically. In the mechanism below no non-linear equation occurs any more from the tool view, since these equations are solved analytically in the JointRRR joints. For more details, see MultiBody.UsersGuide.Tutorial.LoopStructures.AnalyticLoopHandling.



Parameters

Type	Name	Default	Description
Length	rh[3]	{0.5,0,0}	Position vector from 'lower left' revolute to 'lower right' revolute joint for all the 3 loops [m]
Length	rv[3]	{0,0.5,0}	Position vector from 'lower left' revolute to 'upper left' revolute joint [m]
Length	r1b[3]	{0.1,0.5,0}	[m]
Length	r1a[3]	r1b + rh - rv	[m]
Length	r2b[3]	{0.1,0.6,0}	[m]
Length	r2a[3]	r2b + rh - r1b	[m]
Length	r3b[3]	{0,0.55,0}	[m]
Length	r3a[3]	r3b + rh - r2b	[m]

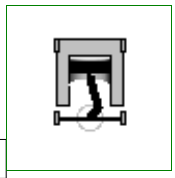
Modelica.Mechanics.MultiBody.Examples.Loops.Utilities

Utility models for Examples.Loops

Package Content

Name	Description
Cylinder	
GasForce	
GasForce2	Rough approximation of gas force in a cylinder
CylinderBase	One cylinder with analytic handling of kinematic loop
Cylinder_analytic_CAD	
EngineV6_analytic	V6 engine with analytic loop handling
Engine1bBase	Model of one cylinder engine with gas force

Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.Cylinder



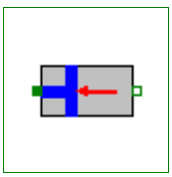
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Length	cylinderTopPosition	0.42	Length from crank shaft to end of cylinder. [m]
Length	pistonLength	0.1	Length of cylinder [m]
Length	rodLength	0.2	Length of rod [m]
Length	crankLength	0.2	Length of crank shaft in x direction [m]
Length	crankPinOffset	0.1	Offset of crank pin from center axis [m]
Length	crankPinLength	0.1	Offset of crank pin from center axis [m]
Angle	cylinderInclination	0	Inclination of cylinder [rad]
Angle	crankAngleOffset	0	Offset for crank angle [rad]
Length	cylinderLength	cylinderTopPosition - (pisto...	Maximum length of cylinder volume [m]

Connectors

Type	Name	Description
Frame_a	cylinder_a	
Frame_a	cylinder_b	
Frame_a	crank_a	
Frame_a	crank_b	

Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.GasForce



Parameters

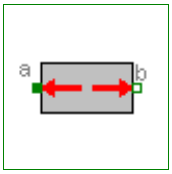
Type	Name	Default	Description
Length	L		Length of cylinder [m]

Length	d		diameter of cylinder [m]
Real	k0	0.01	
Real	k1	1	
Real	k	1	

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.GasForce2



Rough approximation of gas force in a cylinder

Information

The gas force in a cylinder is computed as function of the relative distance of the two flanges. It is required that s_rel = flange_b.s - flange_a.s is in the range

```
0 ≤ s_rel ≤ L
```

where the parameter L is the length of the cylinder. If this assumption is not fulfilled, an error occurs.

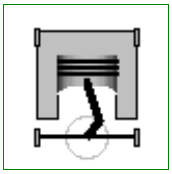
Parameters

Type	Name	Default	Description
Length	L		Length of cylinder [m]
Length	d		diameter of cylinder [m]
Real	k0	0.01	
Real	k1	1	
Real	k	1	

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.CylinderBase



One cylinder with analytic handling of kinematic loop

Parameters

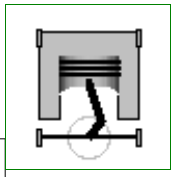
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Length	cylinderTopPosition	0.42	Length from crank shaft to end of cylinder. [m]
Length	crankLength	0.14	Length of crank shaft in x direction [m]
Length	crankPinOffset	0.05	Offset of crank pin from center axis [m]

Length	crankPinLength	0.1	Offset of crank pin from center axis [m]
Angle_deg	cylinderInclination	0	Inclination of cylinder [deg]
Angle_deg	crankAngleOffset	0	Offset for crank angle [deg]
Piston			
Length	pistonLength	0.1	Length of cylinder [m]
Length	pistonCenterOfMass	pistonLength/2	Distance from frame_a to center of mass of piston [m]
Mass	pistonMass	6	Mass of piston [kg]
Inertia	pistonInertia_11	0.0088	Inertia 11 of piston with respect to center of mass frame, parallel to frame_a [kg.m2]
Inertia	pistonInertia_22	0.0076	Inertia 22 of piston with respect to center of mass frame, parallel to frame_a [kg.m2]
Inertia	pistonInertia_33	0.0088	Inertia 33 of piston with respect to center of mass frame, parallel to frame_a [kg.m2]
Rod			
Length	rodLength	0.175	Length of rod [m]
Length	rodCenterOfMass	rodLength/2	Distance from frame_a to center of mass of piston [m]
Mass	rodMass	1	Mass of rod [kg]
Inertia	rodInertia_11	0.006	Inertia 11 of rod with respect to center of mass frame, parallel to frame_a [kg.m2]
Inertia	rodInertia_22	0.0005	Inertia 22 of rod with respect to center of mass frame, parallel to frame_a [kg.m2]
Inertia	rodInertia_33	0.006	Inertia 33 of rod with respect to center of mass frame, parallel to frame_a [kg.m2]

Connectors

Type	Name	Description
Frame_a	cylinder_a	
Frame_a	cylinder_b	
Frame_a	crank_a	
Frame_a	crank_b	

Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.Cylinder_analytic_CAD



Parameters

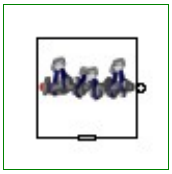
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Length	cylinderTopPosition	0.42	Length from crank shaft to end of cylinder. [m]
Length	crankLength	0.14	Length of crank shaft in x direction [m]
Length	crankPinOffset	0.05	Offset of crank pin from center axis [m]
Length	crankPinLength	0.1	Offset of crank pin from center axis [m]
Angle_deg	cylinderInclination	0	Inclination of cylinder [deg]
Angle_d	crankAngleOffset	0	Offset for crank angle [deg]

eg			
Piston			
Length	pistonLength	0.1	Length of cylinder [m]
Length	pistonCenterOfMass	pistonLength/2	Distance from frame_a to center of mass of piston [m]
Mass	pistonMass	6	Mass of piston [kg]
Inertia	pistonInertia_11	0.0088	Inertia 11 of piston with respect to center of mass frame, parallel to frame_a [kg.m2]
Inertia	pistonInertia_22	0.0076	Inertia 22 of piston with respect to center of mass frame, parallel to frame_a [kg.m2]
Inertia	pistonInertia_33	0.0088	Inertia 33 of piston with respect to center of mass frame, parallel to frame_a [kg.m2]
Rod			
Length	rodLength	0.175	Length of rod [m]
Length	rodCenterOfMass	rodLength/2	Distance from frame_a to center of mass of piston [m]
Mass	rodMass	1	Mass of rod [kg]
Inertia	rodInertia_11	0.006	Inertia 11 of rod with respect to center of mass frame, parallel to frame_a [kg.m2]
Inertia	rodInertia_22	0.0005	Inertia 22 of rod with respect to center of mass frame, parallel to frame_a [kg.m2]
Inertia	rodInertia_33	0.006	Inertia 33 of rod with respect to center of mass frame, parallel to frame_a [kg.m2]

Connectors

Type	Name	Description
Frame_a	cylinder_a	
Frame_a	cylinder_b	
Frame_a	crank_a	
Frame_a	crank_b	

Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.EngineV6_analytic



V6 engine with analytic loop handling

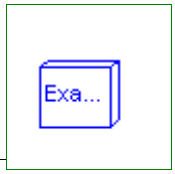
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled

Connectors

Type	Name	Description
Flange_b	flange_b	
Frame_a	frame_a	

Modelica.Mechanics.MultiBody.Examples.Loops.Utilities.Engine1bBase



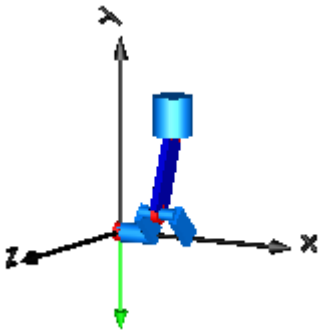
Model of one cylinder engine with gas force

Information

This is a model of the mechanical part of one cylinder of an engine. The combustion is not modelled. The "inertia" component at the lower left part is the output inertia of the engine driving the gearbox. The angular velocity of the output inertia has a start value of 10 rad/s in order to demonstrate the movement of the engine.

The engine is modeled solely by revolute and prismatic joints. Since this results in a **planar** loop there is the well known difficulty that the cut-forces perpendicular to the loop cannot be uniquely computed, as well as the cut-torques within the plane. This ambiguity is resolved by using the option **planarCutJoint** in the **Advanced** menu of one revolute joint in every planar loop (here: joint B1). This option sets the cut-force in direction of the axis of rotation, as well as the cut-torques perpendicular to the axis of rotation at this joint to zero and makes the problem mathematically well-formed.

An animation of this example is shown in the figure below.



Modelica.Mechanics.MultiBody.Examples.Systems

Examples of complete system models including 3-dimensional mechanics

Information

This package contains complete **system models** where components from different domains are used, including 3-dimensional mechanics.

Content

Model	*Description*
RobotR3 RobotR3.oneAxis RobotR3.fullRobot	6 degree of freedom robot with path planning, controllers, motors, brakes, gears and mechanics. "oneAxis" models only one drive train. "fullRobot" is the complete, detailed robot model.

Package Content

Name	Description

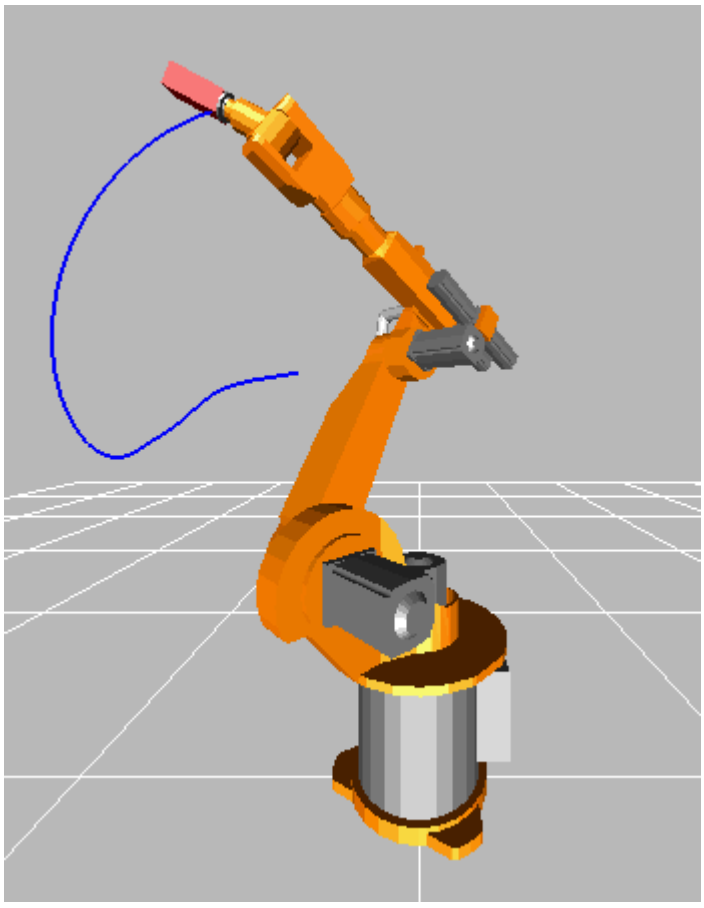
RobotR3	Library to demonstrate robot system models based on the Manutec r3 robot

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3

Library to demonstrate robot system models based on the Manutec r3 robot

Information

This package contains models of the robot r3 of the company Manutec. These models are used to demonstrate in which way complex robot models might be built up by testing first the component models individually before composing them together. Furthermore, it is shown how CAD data can be used for animation.



The following models are available:

oneAxis Test one axis (controller, motor, gearbox).
fullRobot Test complete robot model.

The r3 robot is no longer manufactured. In fact the company Manutec does no longer exist. The parameters of this robot have been determined by measurements in the laboratory of DLR. The measurement procedure is described in:

Tuerk S. (1990): Zur Modellierung der Dynamik von Robotern mit rotatorischen Gelenken. Fortschrittberichte VDI, Reihe 8, Nr. 211, VDI-Verlag 1990.

The robot model is described in detail in

Otter M. (1995): Objektorientierte Modellierung mechatronischer
Systeme am Beispiel geregelter Roboter. Dissertation,
Fortschrittberichte VDI, Reihe 20, Nr. 147, VDI-Verlag 1995.
This report can be downloaded as compressed postscript file
from: http://www.robotic.dlr.de/Martin.Otter/publications.html.

The path planning is performed in a simple way by using essentially the Modelica.Mechanics.Rotational.KinematicPTP block. A user defines a path by start and end angle of every axis. A path is planned such that all axes are moving as fast as possible under the given restrictions of maximum joint speeds and maximum joint accelerations. The actual r3 robot from Manutec had a different path planning strategy. Todays path planning algorithms from robot companies are much more involved.

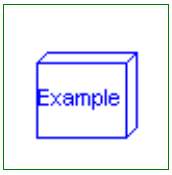
In order to get a nice animation, CAD data from a KUKA robot is used, since CAD data of the original r3 robot was not available. The KUKA CAD data was derived from public data of KUKA available at: http://www.kuka-roboter.de/english/produkte/cad/low_payloads.html. Since dimensions of the corresponding KUKA robot are similar but not identical to the r3 robot, the data of the r3 robot (such as arm lengths) have been modified, such that it matches the CAD data.

In this model, a simplified P-PI cascade controller for every axes is used. The parameters have been manually adjusted by simulations. The original r3 controllers are more complicated. The reason to use simplified controllers is to have a simpler demo.

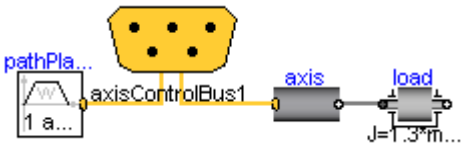
Package Content

Name	Description
oneAxis	Model of one axis of robot (controller, motor, gearbox) with simple load
fullRobot	6 degree of freedom robot with path planning, controllers, motors, brakes, gears and mechanics
Components	Library of components of the robot

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.oneAxis



Model of one axis of robot (controller, motor, gearbox) with simple load



Information

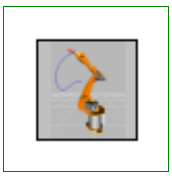
With this model one axis of the r3 robot is checked. The mechanical structure is replaced by a simple load inertia.

Parameters

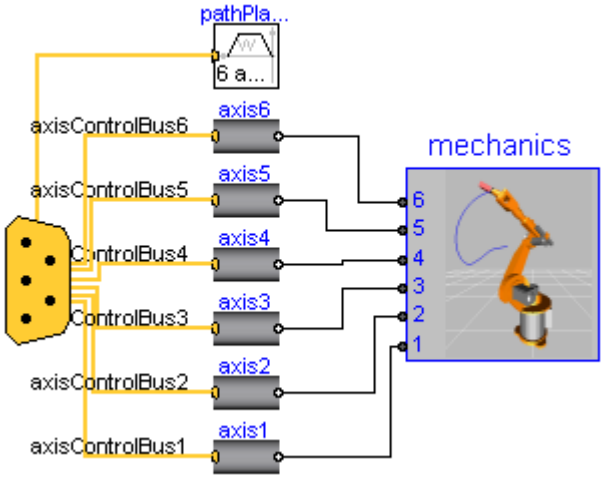
Type	Name	Default	Description
Mass	mLoad	15	Mass of load [kg]
Real	kp	5	Gain of position controller of axis 2
Real	ks	0.5	Gain of speed controller of axis 2
Time	Ts	0.05	Time constant of integrator of speed controller of axis 2 [s]
Real	startAngle	0	Start angle of axis 2 [deg]
Real	endAngle	120	End angle of axis 2 [deg]

Time	swingTime	0.5	Additional time after reference motion is in rest before simulation is stopped [s]
AngularVelocity	refSpeedMax	3	Maximum reference speed [rad/s]
AngularAcceleration	refAccMax	10	Maximum reference acceleration [rad/s2]

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot

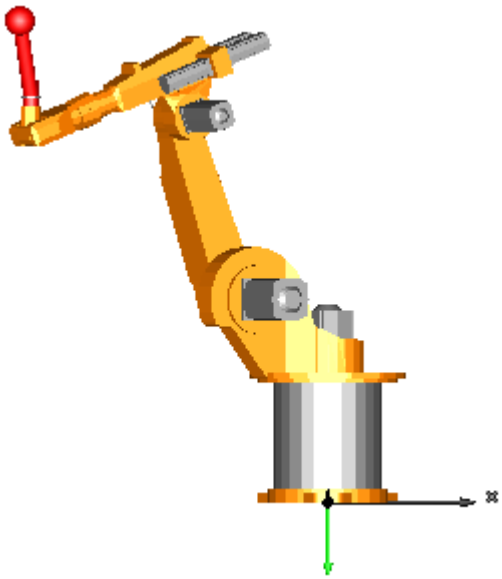


6 degree of freedom robot with path planning, controllers, motors, brakes, gears and mechanics



Information

This is a detailed model of the robot. For animation CAD data is used. Translate and simulate with the default settings (default simulation time = 3 s). Use command script "Scripts\ExamplesfullRobotPlot.mos" to plot variables.



Parameters

Type	Name	Default	Description

Mass	mLoad	15	Mass of load [kg]
Position	rLoad[3]	{0.1,0.25,0.1}	Distance from last flange to load mass [m]
Acceleration	g	9.81	Gravity acceleration [m/s2]
Time	refStartTime	0	Start time of reference motion [s]
Time	refSwingTime	0.7	Additional time after reference motion is in rest before simulation is stopped [s]
Reference			
startAngles			
Real	startAngle1	-60	Start angle of axis 1 [deg]
Real	startAngle2	20	Start angle of axis 2 [deg]
Real	startAngle3	90	Start angle of axis 3 [deg]
Real	startAngle4	0	Start angle of axis 4 [deg]
Real	startAngle5	-110	Start angle of axis 5 [deg]
Real	startAngle6	0	Start angle of axis 6 [deg]
endAngles			
Real	endAngle1	60	End angle of axis 1 [deg]
Real	endAngle2	-70	End angle of axis 2 [deg]
Real	endAngle3	-35	End angle of axis 3 [deg]
Real	endAngle4	45	End angle of axis 4 [deg]
Real	endAngle5	110	End angle of axis 5 [deg]
Real	endAngle6	45	End angle of axis 6 [deg]
Limits			
AngularVelocity	refSpeedMax[6]	{3,1.5,5,3.1,3.1,4.1}	Maximum reference speeds of all joints [rad/s]
AngularAcceleration	refAccMax[6]	{15,15,15,60,60,60}	Maximum reference accelerations of all joints [rad/s2]
Controller			
Axis 1			
Real	kp1	5	Gain of position controller
Real	ks1	0.5	Gain of speed controller
Time	Ts1	0.05	Time constant of integrator of speed controller [s]
Axis 2			
Real	kp2	5	Gain of position controller
Real	ks2	0.5	Gain of speed controller
Time	Ts2	0.05	Time constant of integrator of speed controller [s]
Axis 3			
Real	kp3	5	Gain of position controller
Real	ks3	0.5	Gain of speed controller
Time	Ts3	0.05	Time constant of integrator of speed controller [s]
Axis 4			
Real	kp4	5	Gain of position controller
Real	ks4	0.5	Gain of speed controller
Time	Ts4	0.05	Time constant of integrator of speed controller [s]
Axis 5			
Real	kp5	5	Gain of position controller
Real	ks5	0.5	Gain of speed controller

Time	Ts5	0.05	Time constant of integrator of speed controller [s]
Axis 6			
Real	kp6	5	Gain of position controller
Real	ks6	0.5	Gain of speed controller
Time	Ts6	0.05	Time constant of integrator of speed controller [s]

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components

Library of components of the robot

Information

This library contains the different components of the r3 robot. Usually, there is no need to use this library directly.

Package Content

Name	Description
AxisControlBus	Data bus for one robot axis
ControlBus	Data bus for all axes of robot
PathPlanning1	Generate reference angles for fastest kinematic movement
PathPlanning6	Generate reference angles for fastest kinematic movement
PathToAxisControlBus	Map path planning to one axis control bus
GearType1	Motor inertia and gearbox model for r3 joints 1,2,3
GearType2	Motor inertia and gearbox model for r3 joints 4,5,6
Motor	Motor model including current controller of r3 motors
Controller	P-PI cascade controller for one axis
AxisType1	Axis model of the r3 joints 1,2,3
AxisType2	Axis model of the r3 joints 4,5,6
MechanicalStructure	Model of the mechanical part of the r3 robot (without animation)
InternalConnectors	Internal models that should not be used

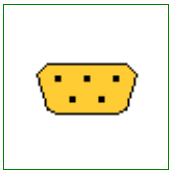
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisControlBus

Data bus for one robot axis

Information

Signal bus that is used to communicate all signals for **one** axis. This is an expandable connector which is "empty". The actual signal content is defined by connecting to an instance of this connector. The signals that are usually used (and are by default listed as choices in the menu that defines the connection to this bus) are defined here.

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.ControlBus

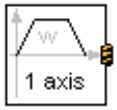


Data bus for all axes of robot

Information

Signal bus that is used to communicate **all signals** of the robot. This is an expandable connector which is "empty". The actual signal content is defined by connecting to an instance of this connector. The sub-buses that are usually used (and are by default listed as choices in the menu that defines the connection to this bus) are defined here.

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.PathPlanning1



Generate reference angles for fastest kinematic movement

Information

Given

- start and end angle of an axis
- maximum speed of the axis
- maximum acceleration of the axis

this component computes the fastest movement under the given constraints. This means, that:

1. The axis accelerates with the maximum acceleration until the maximum speed is reached.
2. Drives with the maximum speed as long as possible.
3. Decelerates with the negative of the maximum acceleration until rest.

The acceleration, constant velocity and deceleration phase are determined in such a way that the movement starts form the start angles and ends at the end angles. The output of this block are the computed angles, angular velocities and angular acceleration and this information is stored as reference motion on the controlBus of the r3 robot.

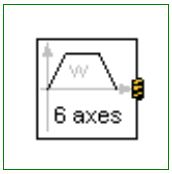
Parameters

Type	Name	Default	Description
Real	angleBegDeg	0	Start angle [deg]
Real	angleEndDeg	1	End angle [deg]
AngularVelocity	speedMax	3	Maximum axis speed [rad/s]
AngularAcceleration	accMax	2.5	Maximum axis acceleration [rad/s2]
Time	startTime	0	Start time of movement [s]
Time	swingTime	0.5	Additional time after reference motion is in rest before simulation is stopped [s]

Connectors

Type	Name	Description
ControlBus	controlBus	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.PathPlanning6



Generate reference angles for fastest kinematic movement

Information

Given

- start and end angles of every axis
- maximum speed of every axis
- maximum acceleration of every axis

this component computes the fastest movement under the given constraints. This means, that:

1. Every axis accelerates with the maximum acceleration until the maximum speed is reached.
2. Drives with the maximum speed as long as possible.
3. Decelerates with the negative of the maximum acceleration until rest.

The acceleration, constant velocity and deceleration phase are determined in such a way that the movement starts form the start angles and ends at the end angles. The output of this block are the computed angles, angular velocities and angular acceleration and this information is stored as reference motion on the controlBus of the r3 robot.

Parameters

Type	Name	Default	Description
Integer	naxis	6	number of driven axis
Real	angleBegDeg[naxis]	zeros(naxis)	Start angles [deg]
Real	angleEndDeg[naxis]	ones(naxis)	End angles [deg]
AngularVelocity	speedMax[naxis]	fill(3, naxis)	Maximum axis speed [rad/s]
AngularAcceleration	accMax[naxis]	fill(2.5, naxis)	Maximum axis acceleration [rad/s2]
Time	startTime	0	Start time of movement [s]
Time	swingTime	0.5	Additional time after reference motion is in rest before simulation is stopped [s]

Connectors

Type	Name	Description
ControlBus	controlBus	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.PathToAxisControlBus

Map path planning to one axis control bus

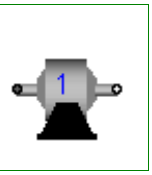
Parameters

Type	Name	Default	Description
Integer	nAxis	6	Number of driven axis
Integer	axisUsed	1	Map path planning of axisUsed to axisControlBus

Connectors

Type	Name	Description
input RealInput	q[nAxis]	
input RealInput	qd[nAxis]	
input RealInput	qdd[nAxis]	
AxisControlBus	axisControlBus	
input BooleanInput	moving[nAxis]	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.GearType1



Motor inertia and gearbox model for r3 joints 1,2,3

Information

Models the gearbox used in the first three joints with all its effects, like elasticity and friction. Coulomb friction is approximated by a friction element acting at the "motor"-side. In reality, bearing friction should be also incorporated at the driven side of the gearbox. However, this would require considerable more effort for the measurement of the friction parameters. Default values for all parameters are given for joint 1. Model relativeStates is used to define the relative angle and relative angular velocity across the spring (=gear elasticity) as state variables. The reason is, that a default initial value of zero of these states makes always sense. If the absolute angle and the absolute angular velocity of model Jmotor would be used as states, and the load angle (= joint angle of robot) is NOT zero, one has always to ensure that the initial values of the motor angle and of the joint angle are modified correspondingly. Otherwise, the spring has an unrealistic deflection at initial time. Since relative quantities are used as state variables, this simplifies the definition of initial values considerably.

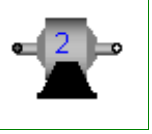
Parameters

Type	Name	Default	Description
Real	i	-105	gear ratio
Real	c	43	Spring constant [N.m/rad]
Real	d	0.005	Damper constant [N.m.s/rad]
Torque	Rv0	0.4	Viscous friction torque at zero velocity [N.m]
Real	Rv1	(0.13/160)	Viscous friction coefficient (R=Rv0+Rv1*abs(qd)) [N.m.s/rad]
Real	peak	1	Maximum static friction torque is peak*Rv0 (peak >= 1)

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.GearType2



Motor inertia and gearbox model for r3 joints 4,5,6

Information

The elasticity and damping in the gearboxes of the outermost three joints of the robot is neglected. Default values for all parameters are given for joint 4.

Parameters

Type	Name	Default	Description
Real	i	-99	Gear ratio
Torque	Rv0	21.8	Viscous friction torque at zero velocity [N.m]
Real	Rv1	9.8	Viscous friction coefficient in [Nms/rad] (R=Rv0+Rv1*abs(qd))
Real	peak	(26.7/21.8)	Maximum static friction torque is peak*Rv0 (peak >= 1)

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.Motor

Motor model including current controller of r3 motors

Information

Default values are given for the motor of joint 1. The input of the motor is the desired current (the actual current is proportional to the torque produced by the motor).

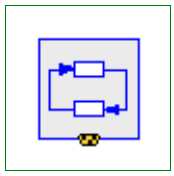
Parameters

Type	Name	Default	Description
Inertia	J	0.0013	Moment of inertia of motor [kg.m2]
Real	k	1.1616	Gain of motor
Real	w	4590	Time constant of motor
Real	D	0.6	Damping constant of motor
AngularVelocity	w_max	315	Maximum speed of motor [rad/s]
Current	i_max	9	Maximum current of motor [A]

Connectors

Type	Name	Description
Flange_b	flange_motor	
AxisControlBus	axisControlBus	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.Controller



P-PI cascade controller for one axis

Information

This controller has an inner PI-controller to control the motor speed, and an outer P-controller to control the motor position of one axis. The reference signals are with respect to the gear-output, and the gear ratio is used in the controller to determine the motor reference signals. All signals are communicated via the "axisControlBus".

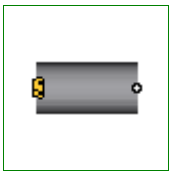
Parameters

Type	Name	Default	Description
Real	kp	10	Gain of position controller
Real	ks	1	Gain of speed controller
Time	Ts	0.01	Time constant of integrator of speed controller [s]
Real	ratio	1	Gear ratio of gearbox

Connectors

Type	Name	Description
AxisControlBus	axisControlBus	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisType1



Axis model of the r3 joints 1,2,3

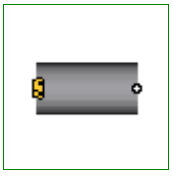
Parameters

Type	Name	Default	Description
Controller			
Real	kp	10	Gain of position controller
Real	ks	1	Gain of speed controller
Time	Ts	0.01	Time constant of integrator of speed controller [s]
Motor			
Real	k	1.1616	Gain of motor
Real	w	4590	Time constant of motor
Real	D	0.6	Damping constant of motor
Inertia	J	0.0013	Moment of inertia of motor [kg.m2]
Gear			
Real	ratio	-105	Gear ratio
Torque	Rv0	0.4	Viscous friction torque at zero velocity in [Nm] [N.m]
Real	Rv1	(0.13/160)	Viscous friction coefficient in [Nms/rad] [N.m.s/rad]
Real	peak	1	Maximum static friction torque is peak*Rv0 (peak >= 1)
Real	c	43	Spring constant [N.m/rad]
Real	cd	0.005	Damper constant [N.m.s/rad]

Connectors

Type	Name	Description
Flange_b	flange	
AxisControlBus	axisControlBus	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisType2



Axis model of the r3 joints 4,5,6

Information

The axis model consists of the **controller**, the **motor** including current controller and the **gearbox** including gear elasticity and bearing friction. The only difference to the axis model of joints 4,5,6 (= model axisType2) is that elasticity and damping in the gear boxes are not neglected.

The input signals of this component are the desired angle and desired angular velocity of the joint. The reference signals have to be "smooth" (position has to be differentiable at least 2 times). Otherwise, the gear elasticity leads to significant oscillations.

Default values of the parameters are given for the axis of joint 1.

Parameters

Type	Name	Default	Description
GearType2	gear	redeclare GearType2 gear(Rv0...	
Controller			
Real	kp	10	Gain of position controller
Real	ks	1	Gain of speed controller
Time	Ts	0.01	Time constant of integrator of speed controller [s]
Motor			
Real	k	1.1616	Gain of motor
Real	w	4590	Time constant of motor
Real	D	0.6	Damping constant of motor
Inertia	J	0.0013	Moment of inertia of motor [kg.m2]
Gear			
Real	ratio	-105	Gear ratio
Torque	Rv0	0.4	Viscous friction torque at zero velocity in [Nm] [N.m]
Real	Rv1	(0.13/160)	Viscous friction coefficient in [Nms/rad] [N.m.s/rad]
Real	peak	1	Maximum static friction torque is peak*Rv0 (peak >= 1)

Connectors

Type	Name	Description
Flange_b	flange	
AxisControlBus	axisControlBus	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.Mechanical Structure

Model of the mechanical part of the r3 robot (without animation)

Information

This model contains the mechanical components of the r3 robot (multibody system).

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Mass	mLoad	15	Mass of load [kg]

Position	rLoad[3]	{0,0.25,0}	Distance from last flange to load mass> [m]
Acceleration	g	9.81	Gravity acceleration [m/s2]

Connectors

Type	Name	Description
Flange_a	axis1	
Flange_a	axis2	
Flange_a	axis3	
Flange_a	axis4	
Flange_a	axis5	
Flange_a	axis6	

Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.InternalConnectors

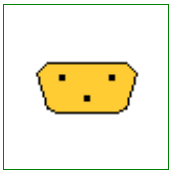
Internal models that should not be used

Information

This package contains the "actual" default bus definitions needed for the robot example. The bus definitions in this package are the default definitions shown in the bus menu when connecting a signal to an expandable connector (here: ControlBus or AxisControlBus). Usually, the connectors of this package should not be utilized by a user.

Package Content

Name	Description
AxisControlBus	Data bus for one robot axis
ControlBus	Data bus for all axes of robot



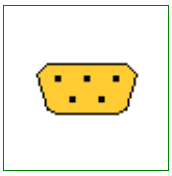
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.InternalConnectors.AxisControlBus

Data bus for one robot axis

Contents

Type	Name	Description
Boolean	motion_ref	= true, if reference motion is not in rest
Angle	angle_ref	Reference angle of axis flange [rad]
Angle	angle	Angle of axis flange [rad]
AngularVelocity	speed_ref	Reference speed of axis flange [rad/s]
AngularVelocity	speed	Speed of axis flange [rad/s]
AngularAcceleration	acceleration_ref	Reference acceleration of axis flange [rad/s2]

AngularAcceleration	acceleration	Acceleration of axis flange [rad/s2]
Current	current_ref	Reference current of motor [A]
Current	current	Current of motor [A]
Angle	motorAngle	Angle of motor flange [rad]
AngularVelocity	motorSpeed	Speed of motor flange [rad/s]



Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.InternalConnectors.ControlBus

Data bus for all axes of robot

Contents

Type	Name	Description
AxisControlBus	axisControlBus1	Bus of axis 1
AxisControlBus	axisControlBus2	Bus of axis 2
AxisControlBus	axisControlBus3	Bus of axis 3
AxisControlBus	axisControlBus4	Bus of axis 4
AxisControlBus	axisControlBus5	Bus of axis 5
AxisControlBus	axisControlBus6	Bus of axis 6

Modelica.Mechanics.MultiBody.Forces

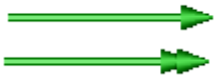
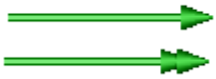
Components that exert forces and/or torques between frames

Information

This package contains components that exert forces and torques between two frame connectors, e.g., between two parts.

Content

Model	*Description*
WorldForce	External force acting at the frame to which this component is connected and defined by 3 input signals, that are interpreted as one vector resolved in the world frame.
WorldTorque	External torque acting at the frame to which this component is connected and defined by 3 input signals, that are interpreted as one vector resolved in the world frame.
WorldForceAndTorque	External force and external torque acting at the frame to which this component is connected and defined by 6 input signals, that are interpreted as a force and as

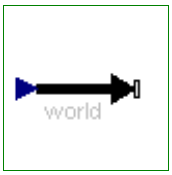
	a torque vector resolved in the world frame. 
FrameForce	External force acting at the frame to which this component is connected and defined by 3 input signals, that are interpreted as one vector resolved in the local frame or in "frame_resolve", if connected.
FrameTorque	External torque acting at the frame to which this component is connected and defined by 3 input signals, that are interpreted as one vector resolved in the local frame or in "frame_resolve", if connected.
FrameForceAndTorque	External force and torque acting at the frame to which this component is connected and defined by 6 input signals, that are interpreted as one force and one torque vector resolved in the local frame or in "frame_resolve", if connected. 
Force	Force acting between two frames defined by 3 input signals resolved in the local frame or in "frame_resolve", if connected.
Torque	Torque acting between two frames defined by 3 input signals resolved in the local frame or in "frame_resolve", if connected.
ForceAndTorque	Force and torque acting between two frames defined by 6 input signals resolved in the local frame or in "frame_resolve", if connected.
LineForceWithMass	General line force component with an optional point mass on the connection line. The force law can be defined by a component of Modelica.Mechanics.Translational
LineForceWithTwoMasses	General line force component with two optional point masses on the connection line. The force law can be defined by a component of Modelica.Mechanics.Translational
Spring	Linear translational spring with optional mass
Damper	Linear (velocity dependent) damper
SpringDamperParallel	Linear spring and damper in parallel connection
SpringDamperSeries	Linear spring and damper in series connection

Package Content

Name	Description

WorldForce	External force acting at frame_b, defined by 3 input signals and resolved in world frame
WorldTorque	External torque acting at frame_b, defined by 3 input signals and resolved in world frame
WorldForceAndTorque	External force and torque acting at frame_b, defined by 6 input signals and resolved in world frame
FrameForce	External force acting at frame_b, defined by 3 input signals and resolved in frame_b or in frame_resolve
FrameTorque	External torque acting at frame_b, defined by 3 input signals and resolved in frame_b or in frame_resolve
FrameForceAndTorque	External force and torque acting at frame_b, defined by 6 input signals and resolved in frame_b or in frame_resolve
Force	Force acting between two frames, defined by 3 input signals and resolved in frame_b or in frame_resolve
Torque	Torque acting between two frames, defined by 3 input signals and resolved in frame_b or in frame_resolve
ForceAndTorque	Force and torque acting between two frames, defined by 6 input signals and resolved in frame_b or in frame_resolve
LineForceWithMass	General line force component with an optional point mass on the connection line
LineForceWithTwoMasses	General line force component with two optional point masses on the connection line
Spring	Linear translational spring with optional mass
Damper	Linear (velocity dependent) damper
SpringDamperParallel	Linear spring and linear damper in parallel
SpringDamperSeries	Linear spring and linear damper in series connection

Modelica.Mechanics.MultiBody.Forces.WorldForce



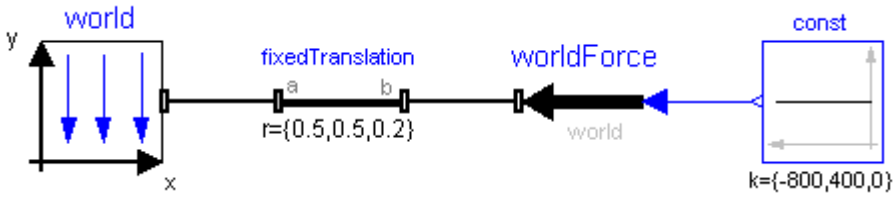
External force acting at frame_b, defined by 3 input signals and resolved in world frame

Information

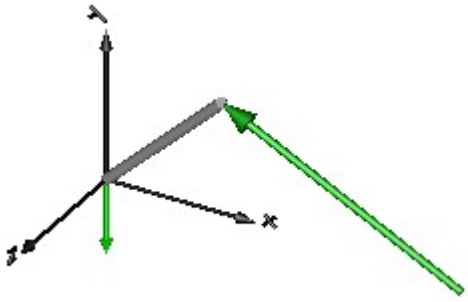
The **3** signals of the **force** connector are interpreted as the x-, y- and z-coordinates of a **force** resolved in the **world frame** and acting at the frame connector to which this component is attached.

This force component is by default visualized as an arrow acting at the connector to which it is connected. The diameter and color of the arrow are fixed and can be defined via parameters **diameter** and **color**. The arrow points in the direction defined by the inPort.signal signals. The length of the arrow is proportional to the length of the force vector using parameter **N_to_m** as scaling factor. For example, if N_to_m = 100 N/m, then a force of 350 N is displayed as an arrow of length 3.5 m.

An example how to use this model is given in the following figure:



This leads to the following animation



Parameters

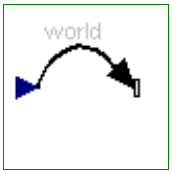
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Real	N_to_m	world.defaultN_to_m	Force arrow scaling (length = force/N_to_m) [N/m]
Diameter	diameter	world.defaultArrowDiameter	Diameter of force arrow [m]
Color	color	Modelica.Mechanics.MultiBody...	Color of arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
input RealInput	force[3]	x-, y-, z-coordinates of force resolved in world frame

---

Modelica.Mechanics.MultiBody.Forces.WorldTorque



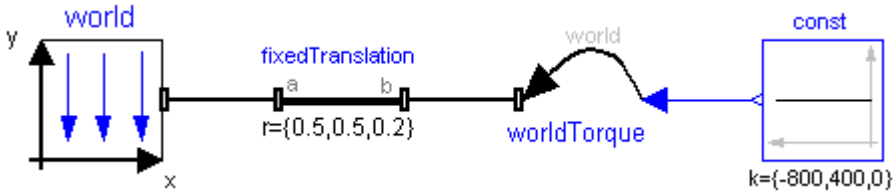
External torque acting at frame_b, defined by 3 input signals and resolved in world frame

Information

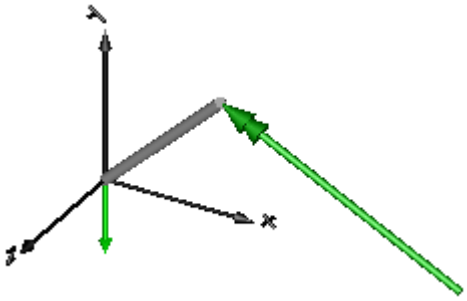
The **3** signals of the **torque** connector are interpreted as the x-, y- and z-coordinates of a **torque** resolved in the **world frame** and acting at the frame connector to which this component is attached.

This torque component is by default visualized as a **double arrow** acting at the connector to which it is connected. The diameter and color of the arrow are fixed and can be defined via parameters **diameter** and **color**. The double arrow points in the direction defined by the inPort.signal signals. The length of the double arrow is proportional to the length of the torque vector using parameter **Nm_to_m** as scaling factor. For example, if Nm_to_m = 100 Nm/m, then a torque of 350 Nm is displayed as an arrow of length 3.5 m.

An example how to use this model is given in the following figure:



This leads to the following animation



Parameters

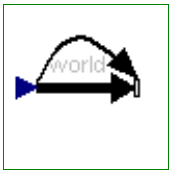
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Real	Nm_to_m	world.defaultNm_to_m	Torque arrow scaling (length = torque/Nm_to_m) [N.m/m]
Diameter	diameter	world.defaultArrowDiameter	Diameter of torque arrow [m]
Color	color	Modelica.Mechanics.MultiBody...	Color of arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
input RealInput	torque[3]	x-, y-, z-coordinates of torque resolved in world frame

---

Modelica.Mechanics.MultiBody.Forces.WorldForceAndTorque



External force and torque acting at frame_b, defined by 6 input signals and resolved in world frame

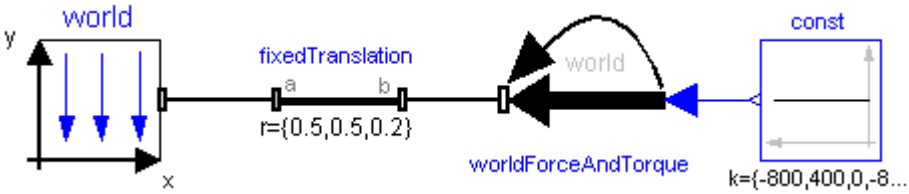
Information

The **6** signals of the **load** connector are interpreted as the x-, y- and z-coordinates of a **force** and as the x-, y-, and z-coordinates of a **torque** resolved in the **world frame** and acting at the frame connector to which this component is attached. The input signals are mapped to the force and torque in the following way:

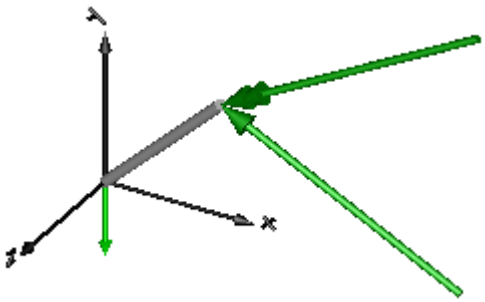
```
force  = load[1:3]
torque = load[4:6]
```

The force and torque are by default visualized as an arrow (force) and as a double arrow (torque) acting at the connector to which they are connected. The diameters and colors of the arrows are fixed and can be defined via parameters **forceDiameter**, **torqueDiameter**, **forceColor** and **torqueColor**. The arrows point in the directions defined by the inPort.signal signals. The lengths of the arrows are proportional to the length of the force and torque vectors, respectively, using parameters **N_to_m** and **Nm_to_m** as scaling factors. For example, if N_to_m = 100 N/m, then a force of 350 N is displayed as an arrow of length 3.5 m.

An example how to use this model is given in the following figure:



This leads to the following animation



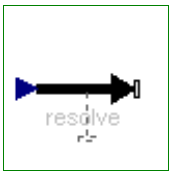
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Real	N_to_m	world.defaultN_to_m	Force arrow scaling (length = force/N_to_m) [N/m]
Real	Nm_to_m	world.defaultNm_to_m	Torque arrow scaling (length = torque/Nm_to_m) [N.m/m]
Diameter	forceDiameter	world.defaultArrowDiameter	Diameter of force arrow [m]
Diameter	torqueDiameter	forceDiameter	Diameter of torque arrow [m]
Color	forceColor	Modelica.Mechanics.MultiBody...	Color of force arrow
Color	torqueColor	Modelica.Mechanics.MultiBody...	Color of torque arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
input RealInput	load[6]	[1:6] = x-, y-, z-coordinates of force and x-, y-, z-coordiantes of torque resolved in world frame

Modelica.Mechanics.MultiBody.Forces.FrameForce



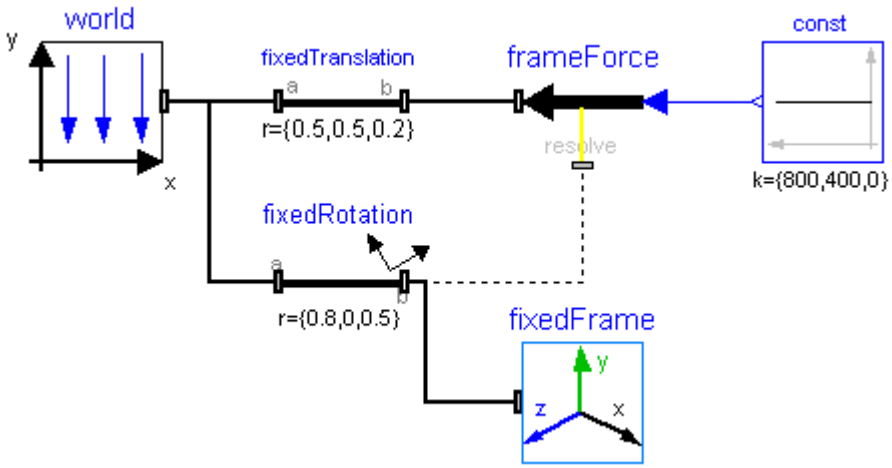
External force acting at frame_b, defined by 3 input signals and resolved in frame_b or in frame_resolve

Information

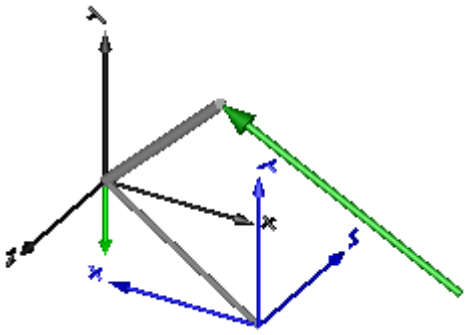
The 3 signals of the **force** connector are interpreted as the x-, y- and z-coordinates of a **force** acting at the frame connector to which this component is attached. If connector **frame_resolve** is **not** connected, the force coordinates are with respect to **frame_b**. If connector **frame_resolve** is connected, the force coordinates are with respect to **frame_resolve**. In this case the force and torque in connector frame_resolve are set to zero, i.e., this connector is solely used to provide the information of the coordinate system, in which the force coordinates are defined.

This force component is by default visualized as an arrow acting at the connector to which it is connected. The diameter and color of the arrow are fixed and can be defined via parameters **diameter** and **color**. The arrow points in the direction defined by the inPort.signal signals. The length of the arrow is proportional to the length of the force vector using parameter **N_to_m** as scaling factor. For example, if N_to_m = 100 N/m, then a force of 350 N is displayed as an arrow of length 3.5 m.

An example how to use this model is given in the following figure:



This leads to the following animation



Parameters

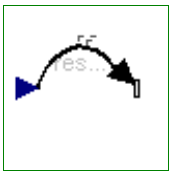
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be

			enabled
if animation = true			
Real	N_to_m	world.defaultN_to_m	Force arrow scaling (length = force/N_to_m) [N/m]
Diameter	diameter	world.defaultArrowDiameter	Diameter of force arrow [m]
Color	color	Modelica.Mechanics.MultiBody...	Color of arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the input signals are resolved in this frame
input RealInput	force[3]	x-, y-, z-coordinates of force resolved in frame_b or frame_resolve (if connected)

Modelica.Mechanics.MultiBody.Forces.FrameTorque



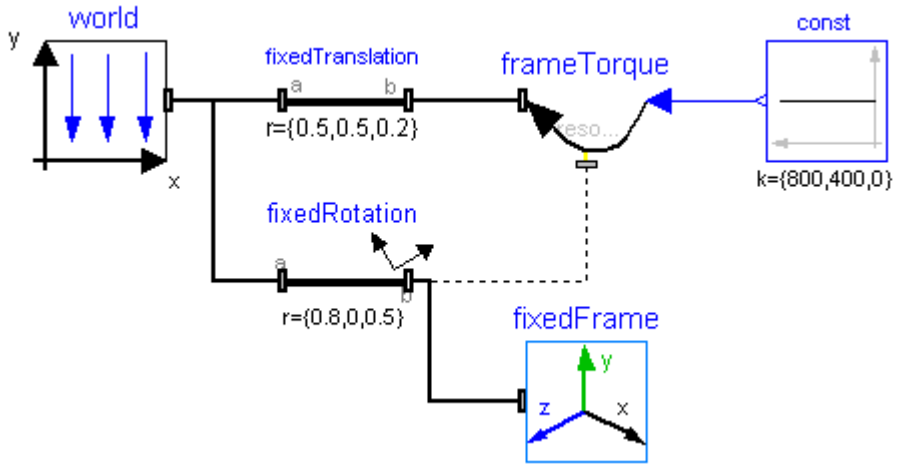
External torque acting at frame_b, defined by 3 input signals and resolved in frame_b or in frame_resolve

Information

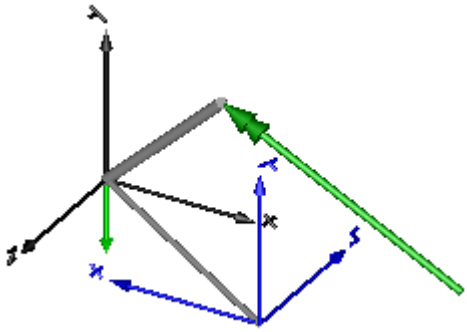
The 3 signals of the **torque** connector are interpreted as the x-, y- and z-coordinates of a **torque** acting at the frame connector to which this component is attached. If connector **frame_resolve** is **not** connected, the torque coordinates are with respect to **frame_b**. If connector **frame_resolve** is connected, the torque coordinates are with respect to **frame_resolve**. In this case the force and torque in connector frame_resolve are set to zero, i.e., this connector is solely used to provide the information of the coordinate system, in which the force coordinates are defined.

This torque component is by default visualized as an arrow acting at the connector to which it is connected. The diameter and color of the arrow are fixed and can be defined via parameters **diameter** and **color**. The arrow points in the direction defined by the inPort.signal signals. The length of the arrow is proportional to the length of the torque vector using parameter **Nm_to_m** as scaling factor. For example, if Nm_to_m = 100 N/m, then a torque of 350 Nm is displayed as an arrow of length 3.5 m.

An example how to use this model is given in the following figure:



This leads to the following animation



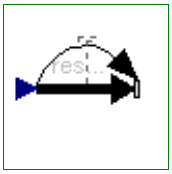
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Real	Nm_to_m	world.defaultNm_to_m	Torque arrow scaling (length = torque/Nm_to_m) [N.m/m]
Diameter	diameter	world.defaultArrowDiameter	Diameter of torque arrow [m]
Color	color	Modelica.Mechanics.MultiBody...	Color of arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the input signals are resolved in this frame
input RealInput	torque[3]	x-, y-, z-coordinates of torque resolved in frame_b or frame_resolve (if connected)

Modelica.Mechanics.MultiBody.Forces.FrameForceAndTorque



External force and torque acting at frame_b, defined by 6 input signals and resolved in frame_b or in frame_resolve

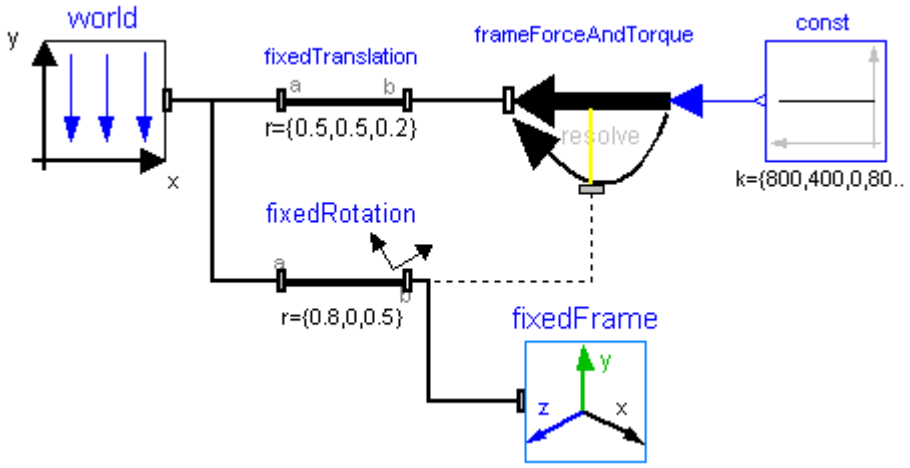
Information

The **6** signals of the **load** connector are interpreted as the x-, y- and z-coordinates of a **force** and as the x-, y-, and z-coordinates of a **torque** acting at the frame connector to which this component is attached. If connector **frame_resolve** is **not** connected, the force and torque coordinates are with respect to **frame_b**. If connector **frame_resolve** is connected, the force and torque coordinates are with respect to **frame_resolve**. In this case the force and torque in connector frame_resolve are set to zero, i.e., this connector is solely used to provide the information of the coordinate system, in which the force coordinates are defined. The input signals are mapped to the force and torque in the following way:

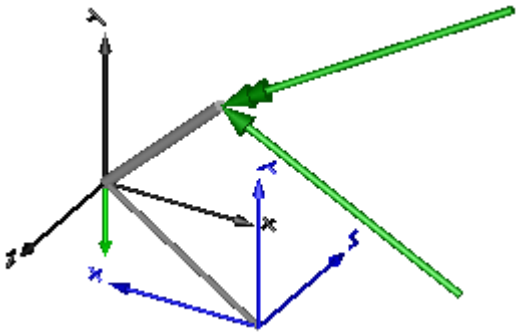
```
force  = load[1:3]
torque = load[4:6]
```

The force and torque are by default visualized as an arrow (force) and as a double arrow (torque) acting at the connector to which they are connected. The diameters and colors of the arrows are fixed and can be defined via parameters **forceDiameter**, **torqueDiameter**, **forceColor** and **torqueColor**. The arrows point in the directions defined by the inPort.signal signals. The lengths of the arrows are proportional to the length of the force and torque vectors, respectively, using parameters **N_to_m** and **Nm_to_m** as scaling factors. For example, if N_to_m = 100 N/m, then a force of 350 N is displayed as an arrow of length 3.5 m.

An example how to use this model is given in the following figure:



This leads to the following animation



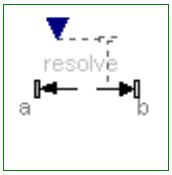
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Real	N_to_m	world.defaultN_to_m	Force arrow scaling (length = force/N_to_m) [N/m]
Real	Nm_to_m	world.defaultNm_to_m	Torque arrow scaling (length = torque/Nm_to_m) [N.m/m]
Diameter	forceDiameter	world.defaultArrowDiameter	Diameter of force arrow [m]
Diameter	torqueDiameter	forceDiameter	Diameter of torque arrow [m]
Color	forceColor	Modelica.Mechanics.MultiBody...	Color of force arrow
Color	torqueColor	Modelica.Mechanics.MultiBody...	Color of torque arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the input signals are resolved in this frame
input RealInput	load[6]	[1:6] = x-, y-, z-coordinates of force and x-, y-, z-coordiantes of torque resolved in frame_b or frame_resolved (if connected)

Modelica.Mechanics.MultiBody.Forces.Force



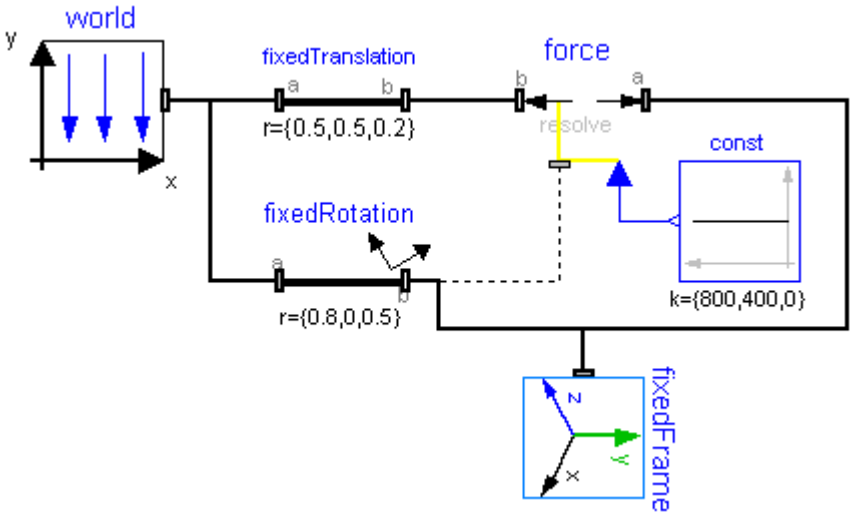
Force acting between two frames, defined by 3 input signals and resolved in frame_b or in frame_resolve

Information

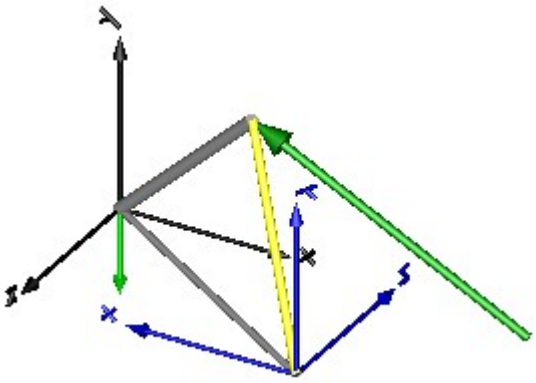
The **3** signals of the **force** connector are interpreted as the x-, y- and z-coordinates of a **force** acting at the frame connector to which frame_b of this component is attached. If connector **frame_resolve** is **not** connected, the force coordinates of the inPort connector are with respect to **frame_b**. If connector **frame_resolve** is connected, the force coordinates of the inPort connector are with respect to **frame_resolve**. In this case the force in connector frame_resolve is set to zero, i.e., this connector is solely used to provide the information of the coordinate system, in which the force coordinates are defined.

Note, the cut-torque in frame_b (frame_b.t) is set to zero. Additionally, a force and torque acts on frame_a in such a way that the force and torque balance between frame_a and frame_b is fulfilled.

An example how to use this model is given in the following figure:



This leads to the following animation (the yellow cylinder characterizes the line between frame_a and frame_b of the Force component, i.e., the force acts with negative sign also on the opposite side of this cylinder, but for clarity this is not shown in the animation):



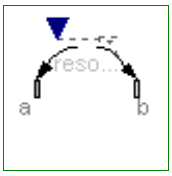
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Real	N_to_m	world.defaultN_to_m	Force arrow scaling (length = force/N_to_m) [N/m]
Diameter	forceDiameter	world.defaultArrowDiameter	Diameter of force arrow [m]
Diameter	connectionLineDiameter	forceDiameter	Diameter of line connecting frame_a and frame_b [m]
Color	forceColor	Modelica.Mechanics.MultiBody...	Color of force arrow
Color	connectionLineColor	Modelica.Mechanics.MultiBody...	Color of line connecting frame_a and frame_b
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the input signals are resolved in this frame
input RealInput	force[3]	x-, y-, z-coordinates of force resolved in frame_b or frame_resolved (if connected)

Modelica.Mechanics.MultiBody.Forces.Torque



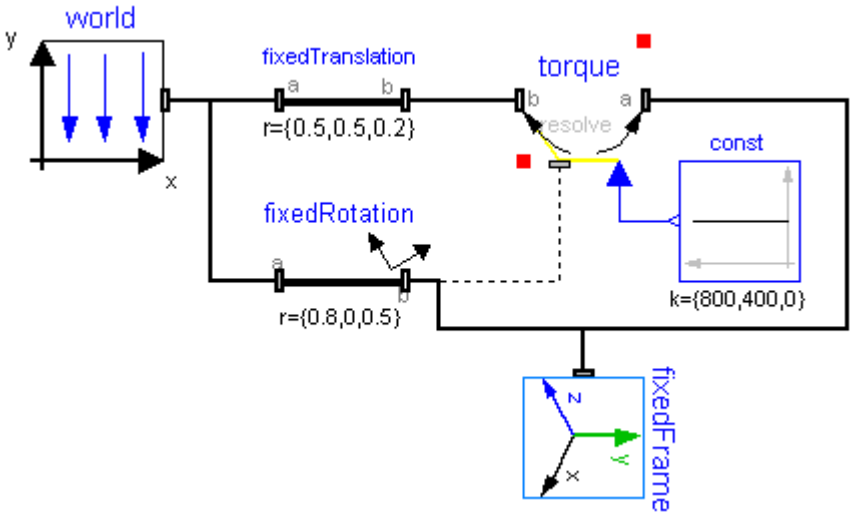
Torque acting between two frames, defined by 3 input signals and resolved in frame_b or in frame_resolve

Information

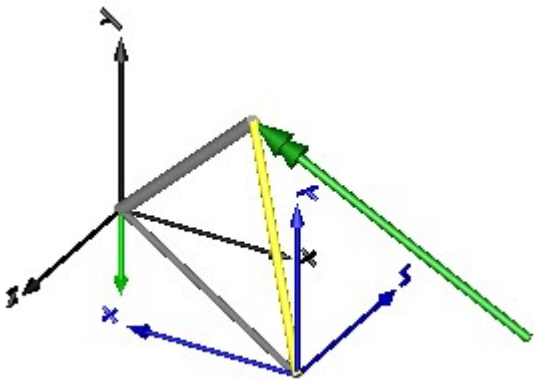
The **3** signals of the **torque** connector are interpreted as the x-, y- and z-coordinates of a **torque** acting at the frame connector to which frame_b of this component is attached. If connector **frame_resolve** is **not** connected, the torque coordinates of the inPort connector are with respect to **frame_b**. If connector **frame_resolve** is connected, the torque coordinates of the inPort connector are with respect to **frame_resolve**. In this case the torque in connector frame_resolve is set to zero, i.e., this connector is solely used to provide the information of the coordinate system, in which the torque coordinates are defined.

Note, the cut-forces in frame_a and frame_b (frame_a.f, frame_b.f) are set to zero and the cut-torque at frame_a (frame_a.t) is the same as the cut-torque at frame_b (frame_b.t) but with opposite sign.

An example how to use this model is given in the following figure:



This leads to the following animation (the yellow cylinder characterizes the line between frame_a and frame_b of the Torque component, i.e., the torque acts with negative sign also on the opposite side of this cylinder, but for clarity this is not shown in the animation):



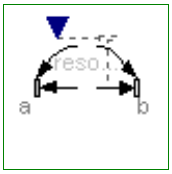
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Real	Nm_to_m	world.defaultNm_to_m	Torque arrow scaling (length = torque/Nm_to_m) [N.m/m]
Diameter	torqueDiameter	world.defaultArrowDiameter	Diameter of torque arrow [m]
Diameter	connectionLineDiameter	torqueDiameter	Diameter of line connecting frame_a and frame_b [m]
Color	torqueColor	Modelica.Mechanics.MultiBody...	Color of torque arrow
Color	connectionLineColor	Modelica.Mechanics.MultiBody...	Color of line connecting frame_a and frame_b
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the input signals are resolved in this frame
input RealInput	torque[3]	x-, y-, z-coordiantes of torque resolved in frame_b or frame_resolved (if connected)

Modelica.Mechanics.MultiBody.Forces.ForceAndTorque



Force and torque acting between two frames, defined by 6 input signals and resolved in frame_b or in frame_resolve

Information

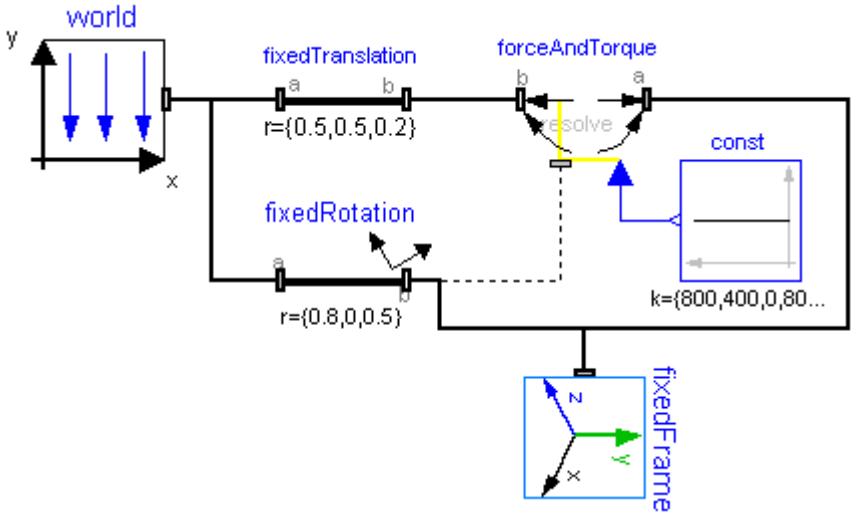
The **6** signals of the **load** connector are interpreted as the x-, y- and z-coordinates of a **force** and as the x-, y-, and z-coordinates of a **torque** acting at the frame connector to which frame_b of this component is attached. If connector **frame_resolve** is **not** connected, the force and torque coordinates are with respect to

frame_b. If connector **frame_resolve** is connected, the force and torque coordinates are with respect to **frame_resolve**. In this case the force and torque in connector frame_resolve are set to zero, i.e., this connector is solely used to provide the information of the coordinate system, in which the force/torque coordinates are defined. The input signals are mapped to the force and torque in the following way:

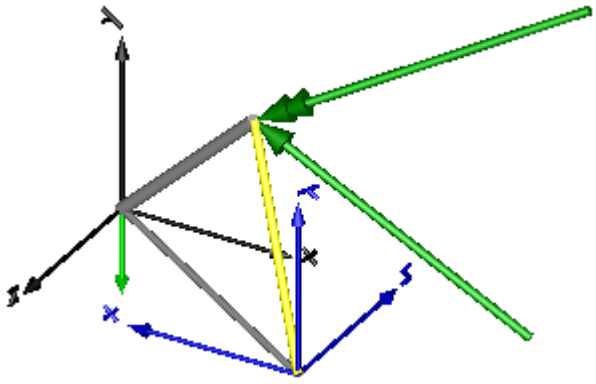
```
force  = load[1:3]
torque = load[4:6]
```

Additionally, a force and torque acts on frame_a in such a way that the force and torque balance between frame_a and frame_b is fulfilled.

An example how to use this model is given in the following figure:



This leads to the following animation (the yellow cylinder characterizes the line between frame_a and frame_b of the ForceAndTorque component, i.e., the force and torque acts with negative sign also on the opposite side of this cylinder, but for clarity this is not shown in the animation):



Parameters

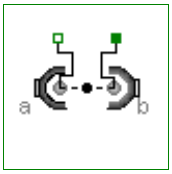
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Real	N_to_m	world.defaultN_to_m	Force arrow scaling (length = force/N_to_m) [N/m]
Real	Nm_to_m	world.defaultNm_to_m	Torque arrow scaling (length

			= torque/Nm_to_m) [N.m/m]
Diameter	forceDiameter	world.defaultArrowDiameter	Diameter of force arrow [m]
Diameter	torqueDiameter	forceDiameter	Diameter of torque arrow [m]
Diameter	connectionLineDiameter	forceDiameter	Diameter of line connecting frame_a and frame_b [m]
Color	forceColor	Modelica.Mechanics.MultiBody...	Color of force arrow
Color	torqueColor	Modelica.Mechanics.MultiBody...	Color of torque arrow
Color	connectionLineColor	Modelica.Mechanics.MultiBody...	Color of line connecting frame_a and frame_b
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the input signals are resolved in this frame
input RealInput	load[6]	[1:6] = x-, y-, z-coordinates of force and x-, y-, z-coordiantes of torque resolved in frame_b or frame_resolved (if connected)

Modelica.Mechanics.MultiBody.Forces.LineForceWithMass



General line force component with an optional point mass on the connection line

Information

This component is used to exert a **line force** between the origin of frame_a and the origin of frame_b by attaching components of the **1-dimensional translational** mechanical library of Modelica (Modelica.Mechanics.Translational) between the two flange connectors **flange_a** and **flange_b**. Optionally, there is a **point mass** on the line connecting the origin of frame_a and the origin of frame_b. This point mass approximates the **mass** of the **force element**. The distance of the point mass from frame_a as a fraction of the distance between frame_a and frame_b is defined via parameter **lengthFraction** (default is 0.5, i.e., the point mass is in the middle of the line).

In the translational library there is the implicit assumption that forces of components that have only one flange connector act with opposite sign on the bearings of the component. This assumption is also used in the LineForceWithMass component: If a connection is present to only one of the flange connectors, then the force in this flange connector acts implicitly with opposite sign also in the other flange connector.

Parameters

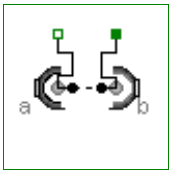
Type	Name	Default	Description
Boolean	animateLine	true	= true, if a line shape between frame_a and frame_b shall be visualized
Boolean	animateMass	true	= true, if point mass shall be visualized as sphere provided m > 0

Mass	m	0	Mass of point mass on the connetion line between the origin of frame_a and the origin of frame_b [kg]
Real	lengthFraction	0.5	Location of point mass with respect to frame_a as a fraction of the distance from frame_a to frame_b
Animation			
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
if animateLine = true			
ShapeType	lineShapeType	"cylinder"	Type of shape visualizing the line from frame_a to frame_b
Length	lineShapeWidth	world.defaultArrowDiameter	Width of shape [m]
Length	lineShapeHeight	lineShapeWidth	Height of shape [m]
ShapeExtra	lineShapeExtra	0.0	Extra parameter for shape
Color	lineShapeColor	Modelica.Mechanics.MultiBody...	Color of line shape
if animateMass = true			
Real	massDiameter	world.defaultBodyDiameter	Diameter of point mass sphere
Color	massColor	Modelica.Mechanics.MultiBody...	Color of point mass
Advanced			
Position	s_small	1.E-10	Prevent zero-division if distance between frame_a and frame_b is zero [m]

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Flange_a	flange_b	1-dim. translational flange (connect force of Translational library between flange_a and flange_b)
Flange_b	flange_a	1-dim. translational flange (connect force of Translational library between flange_a and flange_b)

Modelica.Mechanics.MultiBody.Forces.LineForceWithTwoMasses



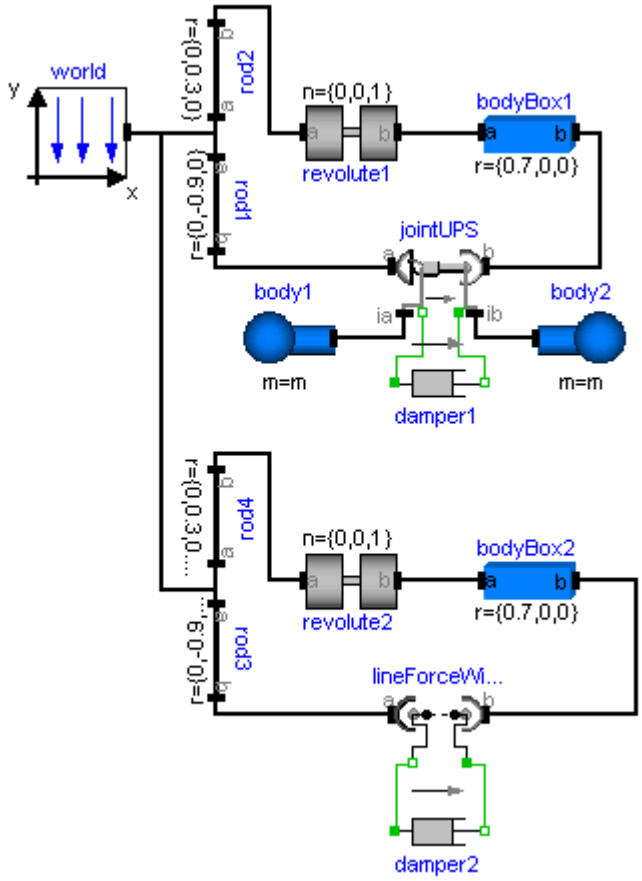
General line force component with two optional point masses on the connection line

Information

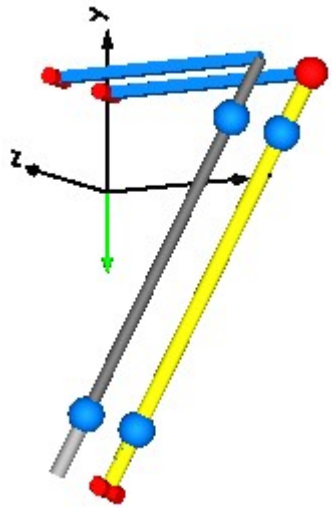
This component is used to exert a **line force** between the origin of frame_a and the origin of frame_b by attaching components of the **1-dimensional translational** mechanical library of Modelica (Modelica.Mechanics.Translational) between the two flange connectors **flange_a** and **flange_b**. Optionally, there are **two point masses** on the line connecting the origin of frame_a and the origin of frame_b. These point masses approximate the **masses** of the **force element**. The locations of the two point masses are defined by their (fixed) distances of L_a relative to frame_a and of L_b relative to frame_b, respectively.

In example MultiBody.Examples.Elementary.LineForceWithTwoMasses the usage of this line force element

is shown and is compared with an alternative implementation using a MultiBody.Joints.Assemblies.JointUPS component. The composition diagram of this example is displayed in the figure below.



The animation view at time = 0 is shown in the next figure. The system on the left side in the front is the animation with the LineForceWithTwoMasses component whereas the system on the right side in the back is the animation with the JointUPS component. Both implementations yield the same result. However, the implementation with the LineForceWithTwoMasses component is simpler.



In the translational library there is the implicit assumption that forces of components that have only one flange connector act with opposite sign on the bearings of the component. This assumption is also used in the LineForceWithTwoMasses component: If a connection is present to only one of the flange connectors, then the force in this flange connector acts implicitly with opposite sign also in the other flange connector.

Parameters

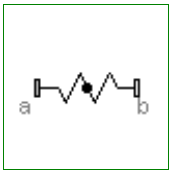
Type	Name	Default	Description
Boolean	animate	true	= true, if animation shall be enabled
Boolean	animateMasses	true	= true, if point masses shall be visualized provided animate=true and m_a, m_b > 0
Mass	m_a	0	Mass of point mass a on the connetion line between the origin of frame_a and the origin of frame_b [kg]
Mass	m_b	0	Mass of point mass b on the connetion line between the origin of frame_a and the origin of frame_b [kg]
Position	L_a	0	Distance between point mass a and frame_a (positive, if in direction of frame_b) [m]
Position	L_b	L_a	Distance between point mass b and frame_b (positive, if in direction of frame_a) [m]
Animation			
Cylinder at frame_a if animation = true			
Diameter	cylinderDiameter_a	world.defaultForceWidth	Diameter of cylinder at frame_a [m]
Length	cylinderLength_a	2*L_a	Length of cylinder at frame_a [m]
Color	color_a	{155,155,155}	Color of cylinder at frame_a
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Cylinder at frame_b if animation = true			
Real	diameterFraction	0.8	Diameter of cylinder at frame_b with respect to diameter of cylinder at frame_a
Length	cylinderLength_b	2*L_b	Length of cylinder at frame_b [m]
Color	color_b	{100,100,100}	Color of cylinder at frame_b
if animation = true and animateMasses = true			
Real	massDiameterFaction	1.7	Diameter of point mass spheres with respect to cylinderDiameter_a
Color	massColor	Modelica.Mechanics.MultiBody...	Color of point masses
Advanced			
Position	s_small	1.E-10	Prevent zero-division if distance between frame_a and frame_b is zero [m]

Connectors

Type	Name	Description

Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Flange_a	flange_b	1-dim. translational flange (connect force of Translational library between flange_a and flange_b)
Flange_b	flange_a	1-dim. translational flange (connect force of Translational library between flange_a and flange_b)

Modelica.Mechanics.MultiBody.Forces.Spring



Linear translational spring with optional mass

Information

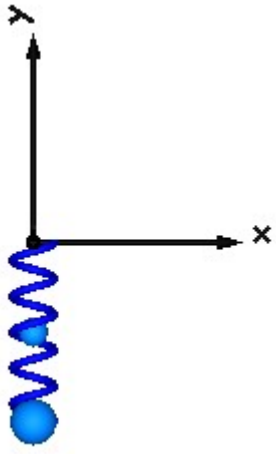
Linear spring acting as line force between frame_a and frame_b. A **force f** is exerted on the origin of frame_b and with opposite sign on the origin of frame_a along the line from the origin of frame_a to the origin of frame_b according to the equation:

```
f = c*(s - s_unstretched);
```

where "c" and "s_unstretched" are parameters and "s" is the distance between the origin of frame_a and the origin of frame_b.

Optionally, the mass of the spring is taken into account by a point mass located on the line between frame_a and frame_b (default: middle of the line). If the spring mass is zero, the additional equations to handle the mass are removed.

In the following figure a typical animation of the spring is shown. The blue sphere in the middle of the spring characterizes the the location of the point mass.



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Boolean	showMass	true	= true, if point mass shall be visualized as sphere if animation=true and m>0
Real	c		Spring constant [N/m]
Length	s_unstretched	0	Unstretched spring length [m]

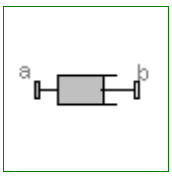
Mass	m	0	Spring mass located on the connetion line between the origin of frame_a and the origin of frame_b [kg]
Real	lengthFraction	0.5	Location of spring mass with respect to frame_a as a fraction of the distance from frame_a to frame_b (=0: at frame_a; =1: at frame_b)
Animation			
if animation = true			
Distance	width	world.defaultForceWidth	Width of spring [m]
Distance	coilWidth	width/10	Width of spring coil [m]
Integer	numberOfWindings	5	Number of spring windings
Color	color	Modelica.Mechanics.MultiBody...	Color of spring
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
if animation = true and showMass = true			
Real	massDiameter	max(0, (width - 2*coilWidth)...	Diameter of mass point sphere
Color	massColor	Modelica.Mechanics.MultiBody...	Color of mass point

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Forces.Damper



Linear (velocity dependent) damper

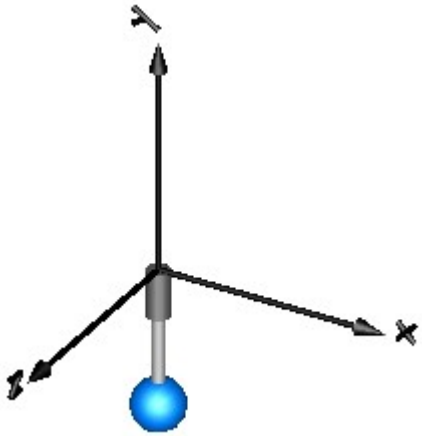
Information

Linear damper acting as line force between frame_a and frame_b. A **force f** is exerted on the origin of frame_b and with opposite sign on the origin of frame_a along the line from the origin of frame_a to the origin of frame_b according to the equation:

```
f = d*der(s);
```

where "d" is a parameter, "s" is the distance between the origin of frame_a and the origin of frame_b and der(s) is the time derivative of "s".

In the following figure a typical animation is shown where a mass is hanging on a damper.



Parameters

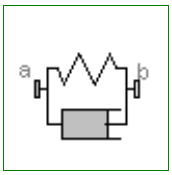
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Real	d	0	Damping constant [N.s/m]
Animation			
if animation = true			
Distance	length_a	world.defaultForceLength	Length of cylinder at frame_a side [m]
Distance	diameter_a	world.defaultForceWidth	Diameter of cylinder at frame_a side [m]
Real	diameter_b	0.6*diameter_a	Diameter of cylinder at frame_b side
Color	color_a	{100,100,100}	Color at frame_a
Color	color_b	{155,155,155}	Color at frame_b
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic..	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Position	s_small	1.E-6	Prevent zero-division if relative distance s=0 [m]

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the force element with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the force element with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Forces.SpringDamperParallel



Linear spring and linear damper in parallel

Information

Linear spring and **dinear damper** in parallel acting as line force between frame_a and frame_b. A **force f** is exerted on the origin of frame_b and with opposite sign on the origin of frame_a along the line from the origin of frame_a to the origin of frame_b according to the equation:

```
f = c*(s - s_unstretched) + d*der(s);
```

where "c", "s_unstretched" and "d" are parameters, "s" is the distance between the origin of frame_a and the origin of frame_b and der(s) is the time derivative of s.

Parameters

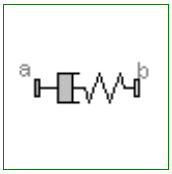
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Real	c		Spring constant [N/m]
Length	s_unstretched	0	Unstretched spring length [m]
Real	d	0	Damping constant [N.s/m]
Animation			
if animation = true			
Distance	width	world.defaultForceWidth	Width of spring [m]
Distance	coilWidth	width/10	Width of spring coil [m]
Integer	numberOfWindings	5	Number of spring windings
Color	color	Modelica.Mechanics.MultiBody...	Color of spring
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Position	s_small	1.E-6	Prevent zero-division if relative distance s=0 [m]

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the force element with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the force element with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Forces.SpringDamperSeries



Linear spring and linear damper in series connection

Information

Linear spring and **linear damper** in series connection acting as line force between frame_a and frame_b:

```
frame_a --> damper -----> spring --> frame_b
        |                |
        |-- s_damper --|   (s_damper is the state variable of this system)
```

A **force f** is exerted on the origin of frame_b and with opposite sign on the origin of frame_a along the line from the origin of frame_a to the origin of frame_b according to the equations:

```
f = c*(s - s_unstretched - s_damper);
f = d*der(s_damper);
```

where "c", "s_unstretched" and "d" are parameters, "s" is the distance between the origin of frame_a and the origin of frame_b. "s_damper" is the length of the damper (= an internal state of this force element) and der(s_damper) is the time derivative of s_damper.

Parameters

Type	Name	Default	Description
Real	c		Spring constant [N/m]
Length	s_unstretched	0	Unstretched spring length [m]
Real	d	0	Damping constant [N.s/m]
Length	s_damper_start	0	Initial length of damper [m]
Advanced			
Position	s_small	1.E-6	Prevent zero-division if relative distance s=0 [m]

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the force element with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the force element with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Frames

Functions to transform rotational frame quantities

Information

Package **Frames** contains type definitions and functions to transform rotational frame quantities. The basic idea is to hide the actual definition of an **orientation** in this package by providing essentially type **Orientation** together with **functions** operating on instances of this type.

Content

In the table below an example is given for every function definition. The used variables have the following declaration:

```
Frames.Orientation R, R1, R2, R_rel, R_inv;
Real[3,3]    T, T_inv;
Real[3]      v1, v2, w1, w2, n_x, n_y, n_z, e, e_x, res_ori, phi;
Real[6]      res_equal;
Real         L, angle;
```

Function/type	*Description*
Orientation R;	New type defining an orientation object that describes the rotation of frame 1 into frame 2.
res_ori = **orientationConstraint**(R);	Return the constraints between the variables of an orientation object (shall be zero).
w1 = **angularVelocity1**(R);	Return angular velocity resolved in frame 1 from orientation object R.
w2 = **angularVelocity2**(R);	Return angular velocity resolved in frame 2 from orientation object R.
v1 = **resolve1**(R,v2);	Transform vector v2 from frame 2 to frame 1.
v2 = **resolve2**(R,v1);	Transform vector v1 from frame 1 to frame 2.
v2 = **resolveRelative**(v1,R1,R2);	Transform vector v1 from frame 1 to frame 2 using absolute orientation objects R1 of frame 1 and R2 of frame 2.
D1 = **resolveDyade1**(R,D2);	Transform second order tensor D2 from frame 2 to frame 1.
D2 = **resolveDyade2**(R,D1);	Transform second order tensor D1 from frame 1 to frame 2.
R = **nullRotation**()	Return orientation object R that does not rotate a frame.
R_inv = **inverseRotation**(R);	Return inverse orientation object.

R_rel = **relativeRotation**(R1,R2);	Return relative orientation object from two absolute orientation objects.
R2 = **absoluteRotation**(R1,R_rel);	Return absolute orientation object from another absolute and a relative orientation object.
R = **planarRotation**(e, angle, der_angle);	Return orientation object of a planar rotation.
angle = **planarRotationAngle**(e, v1, v2);	Return angle of a planar rotation, given the rotation axis and the representations of a vector in frame 1 and frame 2.
R = **axisRotation**(axis, angle, der_angle);	Return orientation object R to rotate around angle along axis of frame 1.
R = **axesRotations**(sequence, angles, der_angles);	Return rotation object to rotate in sequence around 3 axes. Example: R = axesRotations({1,2,3},{pi/2,pi/4,-pi}, zeros(3));
angles = **axesRotationsAngles**(R, sequence);	Return the 3 angles to rotate in sequence around 3 axes to construct the given orientation object.
phi = **smallRotation**(R);	Return rotation angles phi valid for a small rotation R.
R = **from_nxy**(n_x, n_y);	Return orientation object from n_x and n_y vectors.
R = **from_nxz**(n_x, n_z);	Return orientation object from n_x and n_z vectors.
R = **from_T**(T,w);	Return orientation object R from transformation matrix T and its angular velocity w.
R = **from_T2**(T,der(T));	Return orientation object R from transformation matrix T and its derivative der(T).
R = **from_T_inv**(T_inv,w);	Return orientation object R from inverse transformation matrix T_inv and its angular velocity w.
R = **from_Q**(Q,w);	Return orientation object R from quaternion orientation object Q and its angular velocity w.
T = **to_T**(R);	Return transformation matrix T from orientation object R.
T_inv = **to_T_inv**(R);	Return inverse transformation matrix T_inv from orientation object R.
Q = **to_Q**(R);	Return quaternione orientation object Q from orientation object R.
exy = **to_exy**(R);	Return [e_x, e_y] matrix of an orientation object R, with e_x and e_y vectors of frame 2, resolved in frame 1.
L = **length**(n_x);	Return length L of a vector n_x.
e_x = **normalize**(n_x);	Return normalized vector e_x of n_x such that length of e_x is one.
e = **axis**(i);	Return unit vector e directed along axis i
Quaternions	**Package** with functions to transform rotational frame quantities based on quaternions (also called Euler parameters).
TransformationMatrices	**Package** with functions to transform rotational frame quantities based on transformation matrices.

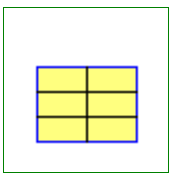
Package Content

Name	Description
Orientation	Orientation object defining rotation from a frame 1 into a frame 2
orientationConstraint	Return residues of orientation constraints (shall be zero)
angularVelocity1	Return angular velocity resolved in frame 1 from orientation object
angularVelocity2	Return angular velocity resolved in frame 2 from orientation object
resolve1	Transform vector from frame 2 to frame 1
resolve2	Transform vector from frame 1 to frame 2
resolveRelative	Transform vector from frame 1 to frame 2 using absolute orientation objects of

	frame 1 and of frame 2
resolveDyade1	Transform second order tensor from frame 2 to frame 1
resolveDyade2	Transform second order tensor from frame 1 to frame 2
nullRotation	Return orientation object that does not rotate a frame
inverseRotation	Return inverse orientation object
relativeRotation	Return relative orientation object
absoluteRotation	Return absolute orientation object from another absolute and a relative orientation object
planarRotation	Return orientation object of a planar rotation
planarRotationAngle	Return angle of a planar rotation, given the rotation axis and the representations of a vector in frame 1 and frame 2
axisRotation	Return rotation object to rotate around an angle along one frame axis
axesRotations	Return fixed rotation object to rotate in sequence around fixed angles along 3 axes
axesRotationsAngles	Return the 3 angles to rotate in sequence around 3 axes to construct the given orientation object
smallRotation	Return rotation angles valid for a small rotation and optionally residues that should be zero
from_nxy	Return fixed orientation object from n_x and n_y vectors
from_nxz	Return fixed orientation object from n_x and n_z vectors
from_T	Return orientation object R from transformation matrix T
from_T2	Return orientation object R from transformation matrix T and its derivative der(T)
from_T_inv	Return orientation object R from inverse transformation matrix T_inv
from_Q	Return orientation object R from quaternion orientation object Q
to_T	Return transformation matrix T from orientation object R
to_T_inv	Return inverse transformation matrix T_inv from orientation object R
to_Q	Return quaternion orientation object Q from orientation object R
to_vector	Map rotation object into vector
to_exy	Map rotation object into e_x and e_y vectors of frame 2, resolved in frame 1
length	Return length of a vector
normalize	Return normalized vector such that length = 1
axis	Return unit vector for x-, y-, or z-axis
Quaternions	Functions to transform rotational frame quantities based on quaternions (also called Euler parameters)
TransformationMatrices	Functions for transformation matrices

Modelica.Mechanics.MultiBody.Frames.Orientation

Orientation object defining rotation from a frame 1 into a frame 2



Information

This object describes the **rotation** from a **frame 1** into a **frame 2**. An instance of this type should never be directly accessed but only with the access functions provided in package Modelica.Mechanics.MultiBody.Frames. As a consequence, it is not necessary to know the internal representation of this object as described in the next paragraphs.

"Orientation" is defined to be a record consisting of two elements: "Real T[3,3]", the transformation matrix to rotate frame 1 into frame 2 and "Real w[3]", the angular velocity of frame 2 with respect to frame 1, resolved in frame 2. Element "T" has the following interpretation:

```
Orientation R;
R.T = [e_x, e_y, e_z];
    e.g., R.T = [1,0,0; 0,1,0; 0,0,1]
```

where $\mathbf{e}_x$,$\mathbf{e}_y$,$\mathbf{e}_z$ are unit vectors in the direction of the x-axis, y-axis, and z-axis of frame 1, resolved in frame 2, respectively. Therefore, if $\mathbf{v}_1$ is vector **v** resolved in frame 1 and $\mathbf{v}_2$ is vector **v** resolved in frame 2, the following relationship holds:

$$\mathbf{v}_2 = \mathbf{R.T} * \mathbf{v}_1$$

The **inverse** orientation **R_inv.T** = **R.T**^{T} describes the rotation from frame 2 into frame 1.

Since the orientation is described by 9 variables, there are 6 constraints between these variables. These constraints are defined in function **Frames.orientationConstraint**.

R.w is the angular velocity of frame 2 with respect to frame 1, resolved in frame 2. Formally, R.w is defined as:
skew(R.w) = R.T***der**(transpose(R.T)) with

```
             |   0   -w[3]   w[2] |
skew(w)  =   |  w[3]   0    -w[1] |
             | -w[2]  w[1]     0  |
```

Modelica definition

```
record Orientation
  "Orientation object defining rotation from a frame 1 into a frame 2"

  import SI = Modelica.SIunits;
  extends Modelica.Icons.Record;
  Real T[3, 3] "Transformation matrix from world frame to local frame";
  SI.AngularVelocity w[3]
    "Absolute angular velocity of local frame, resolved in local frame";

  encapsulated function equalityConstraint
    "Return the constraint residues to express that two frames have the same
orientation"

    import Modelica;
    import Modelica.Mechanics.MultiBody.Frames;
    extends Modelica.Icons.Function;
    input Frames.Orientation R1
      "Orientation object to rotate frame 0 into frame 1";
    input Frames.Orientation R2
      "Orientation object to rotate frame 0 into frame 2";
    output Real residue[3]
      "The rotation angles around x-, y-, and z-axis of frame 1 to rotate frame
```

```
1 into frame 2 for a small rotation (should be zero)";
  algorithm
    residue := {
       Modelica.Math.atan2(cross(R1.T[1, :], R1.T[2, :])*R2.T[2,
:],R1.T[1,:]*R2.T[1,:]),
       Modelica.Math.atan2(-cross(R1.T[1, :],R1.T[2, :])*R2.T[1,
:],R1.T[2,:]*R2.T[2,:]),
       Modelica.Math.atan2(R1.T[2, :]*R2.T[1, :],R1.T[3,:]*R2.T[3,:])};
  end equalityConstraint;


end Orientation;
```

Modelica.Mechanics.MultiBody.Frames.orientationConstraint

Return residues of orientation constraints (shall be zero)

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	residue[6]	Residues of constraints between elements of orientation object (shall be zero)

Modelica.Mechanics.MultiBody.Frames.angularVelocity1

Return angular velocity resolved in frame 1 from orientation object

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
AngularVelocity	w[3]	Angular velocity of frame 2 with respect to frame 1 resolved in frame 1 [rad/s]

Modelica.Mechanics.MultiBody.Frames.angularVelocity2

Return angular velocity resolved in frame 2 from orientation object

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
AngularVelocity	w[3]	Angular velocity of frame 2 with respect to frame 1 resolved in frame 2 [rad/s]

Modelica.Mechanics.MultiBody.Frames.resolve1

Transform vector from frame 2 to frame 1

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2
Real	v2[3]		Vector in frame 2

Outputs

Type	Name	Description
Real	v1[3]	Vector in frame 1

Modelica.Mechanics.MultiBody.Frames.resolve2

Transform vector from frame 1 to frame 2

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2
Real	v1[3]		Vector in frame 1

Outputs

Type	Name	Description
Real	v2[3]	Vector in frame 2

Modelica.Mechanics.MultiBody.Frames.resolveRelative

Transform vector from frame 1 to frame 2 using absolute orientation objects of frame 1 and of frame 2

Inputs

Type	Name	Default	Description
Real	v1[3]		Vector in frame 1
Orientation	R1		Orientation object to rotate frame 0 into frame 1
Orientation	R2		Orientation object to rotate frame 0 into frame 2

Outputs

Type	Name	Description
Real	v2[3]	Vector in frame 2

Modelica.Mechanics.MultiBody.Frames.resolveDyade1

Transform second order tensor from frame 2 to frame 1

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2
Real	D2[3, 3]		Second order tensor resolved in frame 2

Outputs

Type	Name	Description
Real	D1[3, 3]	Second order tensor resolved in frame 1

Modelica.Mechanics.MultiBody.Frames.resolveDyade2

Transform second order tensor from frame 1 to frame 2

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2
Real	D1[3, 3]		Second order tensor resolved in frame 1

Outputs

Type	Name	Description
Real	D2[3, 3]	Second order tensor resolved in frame 2

Modelica.Mechanics.MultiBody.Frames.nullRotation

Return orientation object that does not rotate a frame

Outputs

Type	Name	Description
Orientation	R	Orientation object such that frame 1 and frame 2 are identical

Modelica.Mechanics.MultiBody.Frames.inverseRotation

Return inverse orientation object

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Orientation	R_inv	Orientation object to rotate frame 2 into frame 1

Modelica.Mechanics.MultiBody.Frames.relativeRotation

Return relative orientation object

Inputs

Type	Name	Default	Description
Orientation	R1		Orientation object to rotate frame 0 into frame 1
Orientation	R2		Orientation object to rotate frame 0 into frame 2

Outputs

Type	Name	Description
Orientation	R_rel	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.absoluteRotation

Return absolute orientation object from another absolute and a relative orientation object

Inputs

Type	Name	Default	Description
Orientation	R1		Orientation object to rotate frame 0 into frame 1
Orientation	R_rel		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Orientation	R2	Orientation object to rotate frame 0 into frame 2

Modelica.Mechanics.MultiBody.Frames.planarRotation

Return orientation object of a planar rotation

Inputs

Type	Name	Default	Description
Real	e[3]		Normalized axis of rotation (must have length=1)
Angle	angle		Rotation angle to rotate frame 1 into frame 2 along axis e [rad]
AngularVelocity	der_angle		= der(angle) [rad/s]

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.planarRotationAngle

Return angle of a planar rotation, given the rotation axis and the representations of a vector in frame 1 and frame 2

Information

A call to this function of the form

```
  Real[3]                e, v1, v2;
  Modelica.SIunits.Angle angle;
equation
  angle = planarRotationAngle(e, v1, v2);
```

computes the rotation angle "**angle**" of a planar rotation along unit vector **e**, rotating frame 1 into frame 2, given the coordinate representations of a vector "v" in frame 1 (**v1**) and in frame 2 (**v2**). Therefore, the result of this function fulfills the following equation:

```
v2 = resolve2(planarRotation(e,angle), v1)
```

The rotation angle is returned in the range

```
-π <= angle <= π
```

This function makes the following assumptions on the input arguments

- Vector **e** has length 1, i.e., length(e) = 1
- Vector "v" is not parallel to **e**, i.e., length(cross(e,v1)) ≠ 0

The function does not check the above assumptions. If these assumptions are violated, a wrong result will be returned and/or a division by zero will occur.

Inputs

Type	Name	Default	Description
Real	e[3]		Normalized axis of rotation to rotate frame 1 around e into frame 2 (must have length=1)
Real	v1[3]		A vector v resolved in frame 1 (shall not be parallel to e)
Real	v2[3]		Vector v resolved in frame 2, i.e., v2 = resolve2(planarRotation(e,angle),v1)

Outputs

Type	Name	Description
Angle	angle	Rotation angle to rotate frame 1 into frame 2 along axis e in the range: -pi <= angle <= pi [rad]

Modelica.Mechanics.MultiBody.Frames.axisRotation

Return rotation object to rotate around an angle along one frame axis

Inputs

Type	Name	Default	Description
Integer	axis		Rotate around 'axis' of frame 1
Angle	angle		Rotation angle to rotate frame 1 into frame 2 along 'axis' of frame 1 [rad]
AngularVelocity	der_angle		= der(angle) [rad/s]

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.axesRotations

Return fixed rotation object to rotate in sequence around fixed angles along 3 axes

Inputs

Type	Name	Default	Description
Integer	sequence[3]	{1,2,3}	Sequence of rotations from frame 1 to frame 2 along axis sequence[i]
Angle	angles[3]		Rotation angles around the axes defined in 'sequence' [rad]
AngularVelocity	der_angles[3]		= der(angles) [rad/s]

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.axesRotationsAngles

Return the 3 angles to rotate in sequence around 3 axes to construct the given orientation object

Information

A call to this function of the form

```
    Frames.Orientation     R;
    parameter Integer      sequence[3] = {1,2,3};
    Modelica.SIunits.Angle angles[3];
  equation
    angle = axesRotationAngles(R, sequence);
```

computes the rotation angles "**angles**[1:3]" to rotate frame 1 into frame 2 along axes **sequence**[1:3], given the orientation object **R** from frame 1 to frame 2. Therefore, the result of this function fulfills the following equation:

```
R = axesRotation(sequence, angles)
```

The rotation angles are returned in the range

```
-π <= angles[i] <= π
```

There are **two solutions** for "angles[1]" in this range. Via the third argument **guessAngle1** (default = 0) the returned solution is selected such that |angles[1] - guessAngle1| is minimal. The orientation object R may be in a singular configuration, i.e., there is an infinite number of angle values leading to the same R. The returned solution is selected by setting angles[1] = guessAngle1. Then angles[2] and angles[3] can be uniquely determined in the above range.

Note, that input argument **sequence** has the restriction that only values 1,2,3 can be used and that sequence[1] ≠ sequence[2] and sequence[2] ≠ sequence[3]. Often used values are:

```
sequence = {1,2,3}  // Cardan angle sequence
         = {3,1,3}  // Euler angle sequence
         = {3,2,1}  // Tait-Bryan angle sequence
```

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2
Integer	sequence[3]	{1,2,3}	Sequence of rotations from frame 1 to frame 2 along axis sequence[i]
Angle	guessAngle1	0	Select angles[1] such that |angles[1] - guessAngle1| is a minimum [rad]

Outputs

Type	Name	Description
Angle	angles[3]	Rotation angles around the axes defined in 'sequence' such that R=Frames.axesRotation(sequence,angles); -pi < angles[i] <= pi [rad]

Modelica.Mechanics.MultiBody.Frames.smallRotation

Return rotation angles valid for a small rotation and optionally residues that should be zero

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2
Boolean	withResidues	false	= false/true, if 'angles'/'angles and residues' are returned in phi

Outputs

Type	Name	Description
Angle	phi[if withResidues then 6 else 3]	The rotation angles around x-, y-, and z-axis of frame 1 to rotate frame 1 into frame 2 for a small rotation + optionally 3 residues that should be zero [rad]

Modelica.Mechanics.MultiBody.Frames.from_nxy

Return fixed orientation object from n_x and n_y vectors

Information

It is assumed that the two input vectors n_x and n_y are resolved in frame 1 and are directed along the x and y axis of frame 2 (i.e., n_x and n_y are orthogonal to each other) The function returns the orientation object R to rotate from frame 1 to frame 2.

The function is robust in the sense that it returns always an orientation object R, even if n_y is not orthogonal to n_x. This is performed in the following way:

If n_x and n_y are not orthogonal to each other, first a unit vector e_y is determined that is orthogonal to n_x and is lying in the plane spanned by n_x and n_y. If n_x and n_y are parallel or nearly parallel to each other, a vector e_y is selected arbitrarily such that e_x and e_y are orthogonal to each other.

Inputs

Type	Name	Default	Description
Real	n_x[3]		Vector in direction of x-axis of frame 2, resolved in frame 1
Real	n_y[3]		Vector in direction of y-axis of frame 2, resolved in frame 1

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.from_nxz

Return fixed orientation object from n_x and n_z vectors

Information

It is assumed that the two input vectors n_x and n_z are resolved in frame 1 and are directed along the x and z axis of frame 2 (i.e., n_x and n_z are orthogonal to each other) The function returns the orientation object R to rotate from frame 1 to frame 2.

The function is robust in the sense that it returns always an orientation object R, even if n_z is not orthogonal to n_x. This is performed in the following way:

If n_x and n_z are not orthogonal to each other, first a unit vector e_z is determined that is orthogonal to n_x and is lying in the plane spanned by n_x and n_z. If n_x and n_z are parallel or nearly parallel to each other, a vector e_z is selected arbitrarily such that n_x and e_z are orthogonal to each other.

Inputs

Type	Name	Default	Description
Real	n_x[3]		Vector in direction of x-axis of frame 2, resolved in frame 1
Real	n_z[3]		Vector in direction of z-axis of frame 2, resolved in frame 1

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.from_T

Return orientation object R from transformation matrix T

Inputs

Type	Name	Default	Description
Real	T[3, 3]		Transformation matrix to transform vector from frame 1 to frame 2 (v2=T*v1)
AngularVelocity	w[3]		Angular velocity from frame 2 with respect to frame 1, resolved in frame 2 (skew(w)=T*der(transpose(T))) [rad/s]

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.from_T2

Return orientation object R from transformation matrix T and its derivative der(T)

Information

Computes the orientation object from a transformation matrix T and the derivative der(T) of the transformation matrix. Usually, it is more efficient to use function "from_T" instead, where the angular velocity has to be given as input argument. Only if this is not possible or too difficult to compute, use function from_T2(..).

Inputs

Type	Name	Default	Description
Real	T[3, 3]		Transformation matrix to transform vector from frame 1 to frame 2 (v2=T*v1)
Real	der_T[3, 3]		= der(T)

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.from_T_inv

Return orientation object R from inverse transformation matrix T_inv

Inputs

Type	Name	Default	Description
Real	T_inv[3, 3]		Inverse transformation matrix to transform vector from frame 2 to frame 1 (v1=T_inv*v2)
AngularVelocity	w[3]		Angular velocity from frame 1 with respect to frame 2, resolved in frame 1 (skew(w)=T_inv*der(transpose(T_inv))) [rad/s]

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.from_Q

Return orientation object R from quaternion orientation object Q

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2
AngularVelocity	w[3]		Angular velocity from frame 2 with respect to frame 1, resolved in frame 2 [rad/s]

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.to_T

Return transformation matrix T from orientation object R

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	T[3, 3]	Transformation matrix to transform vector from frame 1 to frame 2 (v2=T*v1)

Modelica.Mechanics.MultiBody.Frames.to_T_inv

Return inverse transformation matrix T_inv from orientation object R

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	T_inv[3, 3]	Inverse transformation matrix to transform vector from frame 2 into frame 1 (v1=T_inv*v2)

Modelica.Mechanics.MultiBody.Frames.to_Q

Return quaternion orientation object Q from orientation object R

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2
Orientation	Q_guess	Quaternions.nullRotation()	Guess value for output Q (there are 2 solutions; the one closer to Q_guess is used

Outputs

Type	Name	Description
Orientation	Q	Quaternions orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.to_vector

Map rotation object into vector

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	vec[9]	Elements of R in one vector

Modelica.Mechanics.MultiBody.Frames.to_exy

Map rotation object into e_x and e_y vectors of frame 2, resolved in frame 1

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	exy[3, 2]	= [e_x, e_y] where e_x and e_y are axes unit vectors of frame 2, resolved in frame 1

Modelica.Mechanics.MultiBody.Frames.length

Return length of a vector

Inputs

Type	Name	Default	Description
Real	r[:]		Vector

Outputs

Type	Name	Description
Real	r_length	Length of vector r

Modelica.Mechanics.MultiBody.Frames.normalize

Return normalized vector such that length = 1

Inputs

Type	Name	Default	Description
Real	r[:]		Vector

Outputs

Type	Name	Description
Real	r_unitLength[size(r, 1)]	Input vector r normalized to length=1

Modelica.Mechanics.MultiBody.Frames.axis

Return unit vector for x-, y-, or z-axis

Inputs

Type	Name	Default	Description
Integer	axis		Axis vector to be returned

Outputs

Type	Name	Description
Real	e[3]	Unit axis vector

Modelica.Mechanics.MultiBody.Frames.Quaternions

Functions to transform rotational frame quantities based on quaternions (also called Euler parameters)

Information

Package **Frames.Quaternions** contains type definitions and functions to transform rotational frame quantities with quaternions. Functions of this package are currently only utilized in MultiBody.Parts.Body components, when quaternions shall be used as parts of the body states. Some functions are also used in a new Modelica package for B-Spline interpolation that is able to interpolate paths consisting of position vectors and orientation objects.

Content

In the table below an example is given for every function definition. The used variables have the following declaration:

```
Quaternions.Orientation Q, Q1, Q2, Q_rel, Q_inv;
Real[3,3]   T, T_inv;
```

```
Real[3]      v1, v2, w1, w2, n_x, n_y, n_z, res_ori, phi;
Real[6]      res_equal;
Real         L, angle;
```

Function/type	*Description*
Orientation Q;	New type defining a quaternion object that describes the rotation of frame 1 into frame 2.
der_Orientation der_Q;	New type defining the first time derivative of Frames.Quaternions.Orientation.
res_ori = **orientationConstraint**(Q);	Return the constraints between the variables of a quaternion object (shall be zero).
w1 = **angularVelocity1**(Q, der_Q);	Return angular velocity resolved in frame 1 from quaternion object Q and its derivative der_Q.
w2 = **angularVelocity2**(Q, der_Q);	Return angular velocity resolved in frame 2 from quaternion object Q and its derivative der_Q.
v1 = **resolve1**(Q,v2);	Transform vector v2 from frame 2 to frame 1.
v2 = **resolve2**(Q,v1);	Transform vector v1 from frame 1 to frame 2.
[v1,w1] = **multipleResolve1**(Q, [v2,w2]);	Transform several vectors from frame 2 to frame 1.
[v2,w2] = **multipleResolve2**(Q, [v1,w1]);	Transform several vectors from frame 1 to frame 2.
Q = **nullRotation**()	Return quaternion object R that does not rotate a frame.
Q_inv = **inverseRotation**(Q);	Return inverse quaternion object.
Q_rel = **relativeRotation**(Q1,Q2);	Return relative quaternion object from two absolute quaternion objects.
Q2 = **absoluteRotation**(Q1,Q_rel);	Return absolute quaternion object from another absolute and a relative quaternion object.
Q = **planarRotation**(e, angle);	Return quaternion object of a planar rotation.
phi = **smallRotation**(Q);	Return rotation angles phi valid for a small rotation.
Q = **from_T**(T);	Return quaternion object Q from transformation matrix T.
Q = **from_T_inv**(T_inv);	Return quaternion object Q from inverse transformation matrix T_inv.
T = **to_T**(Q);	Return transformation matrix T from quaternion object Q.
T_inv = **to_T_inv**(Q);	Return inverse transformation matrix T_inv from quaternion object Q.

Package Content

Name	Description
Orientation	Orientation type defining rotation from a frame 1 into a frame 2 with quaternions {p1,p2,p3,p0}
der_Orientation	First time derivative of Quaternions.Orientation
f orientationConstraint	Return residues of orientation constraints (shall be zero)
f angularVelocity1	Compute angular velocity resolved in frame 1 from quaternion orientation object and its derivative
f angularVelocity2	Compute angular velocity resolved in frame 2 from quaternions orientation object and its derivative
f resolve1	Transform vector from frame 2 to frame 1
f resolve2	Transform vector from frame 1 to frame 2
f multipleResolve1	Transform several vectors from frame 2 to frame 1
f multipleResolve2	Transform several vectors from frame 1 to frame 2

nullRotation	Return quaternions orientation object that does not rotate a frame
inverseRotation	Return inverse quaternions orientation object
relativeRotation	Return relative quaternions orientation object
absoluteRotation	Return absolute quaternions orientation object from another absolute and a relative quaternions orientation object
planarRotation	Return quaternions orientation object of a planar rotation
smallRotation	Return rotation angles valid for a small rotation
from_T	Return quaternions orientation object Q from transformation matrix T
from_T_inv	Return quaternions orientation object Q from inverse transformation matrix T_inv
to_T	Return transformation matrix T from quaternion orientation object Q
to_T_inv	Return inverse transformation matrix T_inv from quaternion orientation object Q

Types and constants

```
  type Orientation
  "Orientation type defining rotation from a frame 1 into a frame 2 with
quaternions {p1,p2,p3,p0}"

    extends Internal.QuaternionBase;

    encapsulated function equalityConstraint
    "Return the constraint residues to express that two frames have the same
quaternion orientation"

      import Modelica;
      import Modelica.Mechanics.MultiBody.Frames.Quaternions;
      extends Modelica.Icons.Function;
      input Quaternions.Orientation Q1
      "Quaternions orientation object to rotate frame 0 into frame 1";
      input Quaternions.Orientation Q2
      "Quaternions orientation object to rotate frame 0 into frame 2";
      output Real residue[3]
      "The half of the rotation angles around x-, y-, and z-axis of frame 1 to
rotate frame 1 into frame 2 for a small rotation (shall be zero)";
    algorithm
      residue := [Q1[4], Q1[3], -Q1[2], -Q1[1]; -Q1[3], Q1[4], Q1[1], -Q1[2];
         Q1[2], -Q1[1], Q1[4], -Q1[3]]*Q2;
    end equalityConstraint;

  end Orientation;

  type der_Orientation = Real[4] (each unit="1/s")
  "First time derivative of Quaternions.Orientation";
```

Modelica.Mechanics.MultiBody.Frames.Quaternions.orientationConstraint

Return residues of orientation constraints (shall be zero)

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	residue[1]	Residue constraint (shall be zero)

Modelica.Mechanics.MultiBody.Frames.Quaternions.angularVelocity1

Compute angular velocity resolved in frame 1 from quaternion orientation object and its derivative

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2
der_Orientation	der_Q		Derivative of Q [1/s]

Outputs

Type	Name	Description
AngularVelocity	w[3]	Angular velocity resolved in frame 1 [rad/s]

Modelica.Mechanics.MultiBody.Frames.Quaternions.angularVelocity2

Compute angular velocity resolved in frame 2 from quaternions orientation object and its derivative

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2
der_Orientation	der_Q		Derivative of Q [1/s]

Outputs

Type	Name	Description
AngularVelocity	w[3]	Angular velocity of frame 2 with respect to frame 1 resolved in frame 2 [rad/s]

Modelica.Mechanics.MultiBody.Frames.Quaternions.resolve1

Transform vector from frame 2 to frame 1

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2
Real	v2[3]		Vector in frame 2

Outputs

Type	Name	Description
Real	v1[3]	Vector in frame 1

Modelica.Mechanics.MultiBody.Frames.Quaternions.resolve2

Transform vector from frame 1 to frame 2

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2
Real	v1[3]		Vector in frame 1

Outputs

Type	Name	Description
Real	v2[3]	Vector in frame 2

Modelica.Mechanics.MultiBody.Frames.Quaternions.multipleResolve1

Transform several vectors from frame 2 to frame 1

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2
Real	v2[3, :]		Vectors in frame 2

Outputs

Type	Name	Description
Real	v1[3, size(v2, 2)]	Vectors in frame 1

Modelica.Mechanics.MultiBody.Frames.Quaternions.multipleResolve2

Transform several vectors from frame 1 to frame 2

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2
Real	v1[3, :]		Vectors in frame 1

Outputs

Type	Name	Description
Real	v2[3, size(v1, 2)]	Vectors in frame 2

Modelica.Mechanics.MultiBody.Frames.Quaternions.nullRotation

Return quaternions orientation object that does not rotate a frame

Outputs

Type	Name	Description
Orientation	Q	Quaternions orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.Quaternions.inverseRotation

Return inverse quaternions orientation object

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Orientation	Q_inv	Quaternions orientation object to rotate frame 2 into frame 1

Modelica.Mechanics.MultiBody.Frames.Quaternions.relativeRotation

Return relative quaternions orientation object

Inputs

Type	Name	Default	Description
Orientation	Q1		Quaternions orientation object to rotate frame 0 into frame 1
Orientation	Q2		Quaternions orientation object to rotate frame 0 into frame 2

Outputs

Type	Name	Description
Orientation	Q_rel	Quaternions orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.Quaternions.absoluteRotation

Return absolute quaternions orientation object from another absolute and a relative quaternions orientation object

Inputs

Type	Name	Default	Description
Orientation	Q1		Quaternions orientation object to rotate frame 0 into frame 1
Orientation	Q_rel		Quaternions orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Orientation	Q2	Quaternions orientation object to rotate frame 0 into frame 2

Modelica.Mechanics.MultiBody.Frames.Quaternions.planarRotation

Return quaternions orientation object of a planar rotation

Inputs

Type	Name	Default	Description
Real	e[3]		Normalized axis of rotation (must have length=1)
Angle	angle		Rotation angle to rotate frame 1 into frame 2 along axis e [rad]

Outputs

Type	Name	Description
Orientation	Q	Quaternions orientation object to rotate frame 1 into frame 2 along axis e

---

Modelica.Mechanics.MultiBody.Frames.Quaternions.smallRotation

Return rotation angles valid for a small rotation

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Angle	phi[3]	The rotation angles around x-, y-, and z-axis of frame 1 to rotate frame 1 into frame 2 for a small relative rotation [rad]

Modelica.Mechanics.MultiBody.Frames.Quaternions.from_T

Return quaternions orientation object Q from transformation matrix T

Inputs

Type	Name	Default	Description
Real	T[3, 3]		Transformation matrix to transform vector from frame 1 to frame 2 (v2=T*v1)
Orientation	Q_guess	nullRotation()	Guess value for Q (there are 2 solutions; the one close to Q_guess is used

Outputs

Type	Name	Description
Orientation	Q	Quaternions orientation object to rotate frame 1 into frame 2 (Q and -Q have same

		transformation matrix)

Modelica.Mechanics.MultiBody.Frames.Quaternions.from_T_inv

Return quaternions orientation object Q from inverse transformation matrix T_inv

Inputs

Type	Name	Default	Description
Real	T_inv[3, 3]		Inverse transformation matrix to transform vector from frame 2 to frame 1 (v1=T_inv*v2)
Orientation	Q_guess	nullRotation()	Guess value for output Q (there are 2 solutions; the one closer to Q_guess is used

Outputs

Type	Name	Description
Orientation	Q	Quaternions orientation object to rotate frame 1 into frame 2 (Q and -Q have same transformation matrix)

Modelica.Mechanics.MultiBody.Frames.Quaternions.to_T

Return transformation matrix T from quaternion orientation object Q

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	T[3, 3]	Transformation matrix to transform vector from frame 1 to frame 2 (v2=T*v1)

Modelica.Mechanics.MultiBody.Frames.Quaternions.to_T_inv

Return inverse transformation matrix T_inv from quaternion orientation object Q

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	T_inv[3, 3]	Transformation matrix to transform vector from frame 2 to frame 1 (v1=T*v2)

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices

Functions for transformation matrices

Information

Package **Frames.TransformationMatrices** contains type definitions and functions to transform rotational frame quantities using transformation matrices.

Content

In the table below an example is given for every function definition. The used variables have the following declaration:

```
Orientation T, T1, T2, T_rel, T_inv;
Real[3]     v1, v2, w1, w2, n_x, n_y, n_z, e, e_x, res_ori, phi;
Real[6]     res_equal;
Real        L, angle;
```

Function/type	*Description*
Orientation T;	New type defining an orientation object that describes the rotation of frame 1 into frame 2.
der_Orientation der_T;	New type defining the first time derivative of Frames.Orientation.
res_ori = **orientationConstraint**(T);	Return the constraints between the variables of an orientation object (shall be zero).
w1 = **angularVelocity1**(T, der_T);	Return angular velocity resolved in frame 1 from orientation object T and its derivative der_T.
w2 = **angularVelocity2**(T, der_T);	Return angular velocity resolved in frame 2 from orientation object T and its derivative der_T.
v1 = **resolve1**(T,v2);	Transform vector v2 from frame 2 to frame 1.
v2 = **resolve2**(T,v1);	Transform vector v1 from frame 1 to frame 2.
[v1,w1] = **multipleResolve1**(T, [v2,w2]);	Transform several vectors from frame 2 to frame 1.
[v2,w2] = **multipleResolve2**(T, [v1,w1]);	Transform several vectors from frame 1 to frame 2.
D1 = **resolveDyade1**(T,D2);	Transform second order tensor D2 from frame 2 to frame 1.
D2 = **resolveDyade2**(T,D1);	Transform second order tensor D1 from frame 1 to frame 2.
T= **nullRotation**()	Return orientation object T that does not rotate a frame.
T_inv = **inverseRotation**(T);	Return inverse orientation object.
T_rel = **relativeRotation**(T1,T2);	Return relative orientation object from two absolute orientation objects.
T2 = **absoluteRotation**(T1,T_rel);	Return absolute orientation object from another absolute and a relative orientation object.
T = **planarRotation**(e, angle);	Return orientation object of a planar rotation.
angle = **planarRotationAngle**(e, v1, v2);	Return angle of a planar rotation, given the rotation axis and the representations of a vector in frame 1 and frame 2.
T = **axisRotation**(i, angle);	Return orientation object T for rotation around axis i of frame 1.
T = **axesRotations**(sequence, angles);	Return rotation object to rotate in sequence around 3 axes. Example: T = axesRotations({1,2,3},{90,45,-90});
angles = **axesRotationsAngles**(T,	Return the 3 angles to rotate in sequence around 3 axes to

sequence);	construct the given orientation object.
phi = **smallRotation**(T);	Return rotation angles phi valid for a small rotation.
T = **from_nxy**(n_x, n_y);	Return orientation object from n_x and n_y vectors.
T = **from_nxz**(n_x, n_z);	Return orientation object from n_x and n_z vectors.
R = **from_T**(T);	Return orientation object R from transformation matrix T.
R = **from_T_inv**(T_inv);	Return orientation object R from inverse transformation matrix T_inv.
T = **from_Q**(Q);	Return orientation object T from quaternion orientation object Q.
T = **to_T**(R);	Return transformation matrix T from orientation object R.
T_inv = **to_T_inv**(R);	Return inverse transformation matrix T_inv from orientation object R.
Q = **to_Q**(T);	Return quaternione orientation object Q from orientation object T.
exy = **to_exy**(T);	Return [e_x, e_y] matrix of an orientation object T, with e_x and e_y vectors of frame 2, resolved in frame 1.

Package Content

Name	Description
Orientation	Orientation type defining rotation from a frame 1 into a frame 2 with a transformation matrix
der_Orientation	New type defining the first time derivative of Orientation
orientationConstraint	Return residues of orientation constraints (shall be zero)
angularVelocity1	Return angular velocity resolved in frame 1 from orientation object and its derivative
angularVelocity2	Return angular velocity resolved in frame 2 from orientation object and its derivative
resolve1	Transform vector from frame 2 to frame 1
resolve2	Transform vector from frame 1 to frame 2
multipleResolve1	Transform several vectors from frame 2 to frame 1
multipleResolve2	Transform several vectors from frame 1 to frame 2
resolveDyade1	Transform second order tensor from frame 2 to frame 1
resolveDyade2	Transform second order tensor from frame 1 to frame 2
nullRotation	Return orientation object that does not rotate a frame
inverseRotation	Return inverse orientation object
relativeRotation	Return relative orientation object
absoluteRotation	Return absolute orientation object from another absolute and a relative orientation object
planarRotation	Return orientation object of a planar rotation
planarRotationAngle	Return angle of a planar rotation, given the rotation axis and the representations of a vector in frame 1 and frame 2
axisRotation	Return rotation object to rotate around one frame axis
axesRotations	Return rotation object to rotate in sequence around 3 axes
axesRotationsAngles	Return the 3 angles to rotate in sequence around 3 axes to construct the given orientation object

smallRotation	Return rotation angles valid for a small rotation and optionally residues that should be zero
from_nxy	Return orientation object from n_x and n_y vectors
from_nxz	Return orientation object from n_x and n_z vectors
from_T	Return orientation object R from transformation matrix T
from_T_inv	Return orientation object R from inverse transformation matrix T_inv
from_Q	Return orientation object T from quaternion orientation object Q
to_T	Return transformation matrix T from orientation object R
to_T_inv	Return inverse transformation matrix T_inv from orientation object R
to_Q	Return quaternion orientation object Q from orientation object T
to_vector	Map rotation object into vector
to_exy	Map rotation object into e_x and e_y vectors of frame 2, resolved in frame 1

Types and constants

```
  type Orientation
  "Orientation type defining rotation from a frame 1 into a frame 2 with a
transformation matrix"

    extends Internal.TransformationMatrix;

    encapsulated function equalityConstraint
    "Return the constraint residues to express that two frames have the same
orientation"

      import Modelica;
      import Modelica.Mechanics.MultiBody.Frames.TransformationMatrices;
      extends Modelica.Icons.Function;
      input TransformationMatrices.Orientation T1
      "Orientation object to rotate frame 0 into frame 1";
      input TransformationMatrices.Orientation T2
      "Orientation object to rotate frame 0 into frame 2";
      output Real residue[3]
      "The rotation angles around x-, y-, and z-axis of frame 1 to rotate frame
1 into frame 2 for a small rotation (should be zero)";
    algorithm
      residue := {cross(T1[1, :], T1[2, :])*T2[2, :],-cross(T1[1, :], T1[2, :])
        *T2[1, :],T1[2, :]*T2[1, :]};
    end equalityConstraint;
  end Orientation;

  type der_Orientation = Real[3, 3] (each unit="1/s")
  "New type defining the first time derivative of Orientation";
```

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.orientationConstraint

Return residues of orientation constraints (shall be zero)

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	residue[6]	Residues of constraints between elements of orientation object (shall be zero)

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.angularVelocity1

Return angular velocity resolved in frame 1 from orientation object and its derivative

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
der_Orientation	der_T		Derivative of T [1/s]

Outputs

Type	Name	Description
AngularVelocity	w[3]	Angular velocity of frame 2 with respect to frame 1 resolved in frame 1 [rad/s]

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.angularVelocity2

Return angular velocity resolved in frame 2 from orientation object and its derivative

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
der_Orientation	der_T		Derivative of T [1/s]

Outputs

Type	Name	Description
AngularVelocity	w[3]	Angular velocity of frame 2 with respect to frame 1 resolved in frame 2 [rad/s]

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.resolve1

Transform vector from frame 2 to frame 1

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Real	v2[3]		Vector in frame 2

Outputs

Type	Name	Description
Real	v1[3]	Vector in frame 1

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.resolve2

Transform vector from frame 1 to frame 2

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Real	v1[3]		Vector in frame 1

Outputs

Type	Name	Description
Real	v2[3]	Vector in frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.multipleResolve1

Transform several vectors from frame 2 to frame 1

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Real	v2[3, :]		Vectors in frame 2

Outputs

Type	Name	Description
Real	v1[3, size(v2, 2)]	Vectors in frame 1

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.multipleResolve2

Transform several vectors from frame 1 to frame 2

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Real	v1[3, :]		Vectors in frame 1

Outputs

Type	Name	Description
Real	v2[3, size(v1, 2)]	Vectors in frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.resolveDyade1

Transform second order tensor from frame 2 to frame 1

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Real	D2[3, 3]		Second order tensor resolved in frame 2

Outputs

Type	Name	Description
Real	D1[3, 3]	Second order tensor resolved in frame 1

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.resolveDyade2

Transform second order tensor from frame 1 to frame 2

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Real	D1[3, 3]		Second order tensor resolved in frame 1

Outputs

Type	Name	Description
Real	D2[3, 3]	Second order tensor resolved in frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.nullRotation

Return orientation object that does not rotate a frame

Outputs

Type	Name	Description
Orientation	T	Orientation object such that frame 1 and frame 2 are identical

---

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.inverseRotation

Return inverse orientation object

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description

Orientation	T_inv	Orientation object to rotate frame 2 into frame 1

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.relativeRotation

Return relative orientation object

Inputs

Type	Name	Default	Description
Orientation	T1		Orientation object to rotate frame 0 into frame 1
Orientation	T2		Orientation object to rotate frame 0 into frame 2

Outputs

Type	Name	Description
Orientation	T_rel	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.absoluteRotation

Return absolute orientation object from another absolute and a relative orientation object

Inputs

Type	Name	Default	Description
Orientation	T1		Orientation object to rotate frame 0 into frame 1
Orientation	T_rel		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Orientation	T2	Orientation object to rotate frame 0 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.planarRotation

Return orientation object of a planar rotation

Inputs

Type	Name	Default	Description
Real	e[3]		Normalized axis of rotation (must have length=1)
Angle	angle		Rotation angle to rotate frame 1 into frame 2 along axis e [rad]

Outputs

Type	Name	Description
Orientation	T	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.planarRotationAngle

Return angle of a planar rotation, given the rotation axis and the representations of a vector in frame 1 and frame 2

Information

A call to this function of the form

```
  Real[3]                e, v1, v2;
  Modelica.SIunits.Angle angle;
equation
  angle = planarRotationAngle(e, v1, v2);
```

computes the rotation angle "**angle**" of a planar rotation along unit vector **e**, rotating frame 1 into frame 2, given the coordinate representations of a vector "v" in frame 1 (**v1**) and in frame 2 (**v2**). Therefore, the result of this function fulfills the following equation:

```
v2 = resolve2(planarRotation(e,angle), v1)
```

The rotation angle is returned in the range

```
-π <= angle <= π
```

This function makes the following assumptions on the input arguments

- Vector **e** has length 1, i.e., length(e) = 1
- Vector "v" is not parallel to **e**, i.e., length(cross(e,v1)) ≠ 0

The function does not check the above assumptions. If these assumptions are violated, a wrong result will be returned and/or a division by zero will occur.

Inputs

Type	Name	Default	Description
Real	e[3]		Normalized axis of rotation to rotate frame 1 around e into frame 2 (must have length=1)
Real	v1[3]		A vector v resolved in frame 1 (shall not be parallel to e)
Real	v2[3]		Vector v resolved in frame 2, i.e., v2 = resolve2(planarRotation(e,angle),v1)

Outputs

Type	Name	Description
Angle	angle	Rotation angle to rotate frame 1 into frame 2 along axis e in the range: -pi <= angle <= pi [rad]

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.axisRotation

Return rotation object to rotate around one frame axis

Inputs

Type	Name	Default	Description
Integer	axis		Rotate around 'axis' of frame 1
Angle	angle		Rotation angle to rotate frame 1 into frame 2 along 'axis' of frame 1 [rad]

Outputs

Type	Name	Description
Orientation	T	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.axesRotations

Return rotation object to rotate in sequence around 3 axes

Inputs

Type	Name	Default	Description
Integer	sequence[3]	{1,2,3}	Sequence of rotations from frame 1 to frame 2 along axis sequence[i]
Angle	angles[3]	{0,0,0}	Rotation angles around the axes defined in 'sequence' [rad]

Outputs

Type	Name	Description
Orientation	T	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.axesRotationsAngles

Return the 3 angles to rotate in sequence around 3 axes to construct the given orientation object

Information

A call to this function of the form

```
    TransformationMatrices.Orientation     T;
    parameter Integer      sequence[3] = {1,2,3};
    Modelica.SIunits.Angle angles[3];
  equation
    angle = axesRotationAngles(T, sequence);
```

computes the rotation angles "**angles**[1:3]" to rotate frame 1 into frame 2 along axes **sequence**[1:3], given the orientation object **T** from frame 1 to frame 2. Therefore, the result of this function fulfills the following equation:

```
T = axesRotation(sequence, angles)
```

The rotation angles are returned in the range

```
-π <= angles[i] <= π
```

There are **two solutions** for "angles[1]" in this range. Via the third argument **guessAngle1** (default = 0) the returned solution is selected such that |angles[1] - guessAngle1| is minimal. The orientation object T may be in a singular configuration, i.e., there is an infinite number of angle values leading to the same T. The returned solution is selected by setting angles[1] = guessAngle1. Then angles[2] and angles[3] can be uniquely determined in the above range.

Note, that input argument **sequence** has the restriction that only values 1,2,3 can be used and that sequence[1] ≠ sequence[2] and sequence[2] ≠ sequence[3]. Often used values are:

```
sequence = {1,2,3}  // Cardan angle sequence
         = {3,1,3}  // Euler angle sequence
         = {3,2,1}  // Tait-Bryan angle sequence
```

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Integer	sequence[3]	{1,2,3}	Sequence of rotations from frame 1 to frame 2 along axis sequence[i]
Angle	guessAngle1	0	Select angles[1] such that |angles[1] - guessAngle1| is a minimum [rad]

Outputs

Type	Name	Description
Angle	angles[3]	Rotation angles around the axes defined in 'sequence' such that T=TransformationMatrices.axesRotation(sequence,angles); -pi < angles[i] <= pi [rad]

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.smallRotation

Return rotation angles valid for a small rotation and optionally residues that should be zero

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Boolean	withResidues	false	= false/true, if 'angles'/'angles and residues' are returned in phi

Outputs

Type	Name	Description
Angle	phi[if withResidues then 6 else 3]	The rotation angles around x-, y-, and z-axis of frame 1 to rotate frame 1 into frame 2 for a small rotation + optionally 3 residues that should be zero [rad]

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_nxy

Return orientation object from n_x and n_y vectors

Information

It is assumed that the two input vectors n_x and n_y are resolved in frame 1 and are directed along the x and y axis of frame 2 (i.e., n_x and n_y are orthogonal to each other) The function returns the orientation object T to rotate from frame 1 to frame 2.

The function is robust in the sense that it returns always an orientation object T, even if n_y is not orthogonal to n_x. This is performed in the following way:

If n_x and n_y are not orthogonal to each other, first a unit vector e_y is determined that is orthogonal to n_x and is lying in the plane spanned by n_x and n_y. If n_x and n_y are parallel or nearly parallel to each other, a vector e_y is selected arbitrarily such that e_x and e_y are orthogonal to each other.

Inputs

Type	Name	Default	Description
Real	n_x[3]		Vector in direction of x-axis of frame 2, resolved in frame 1
Real	n_y[3]		Vector in direction of y-axis of frame 2, resolved in frame 1

Outputs

Type	Name	Description
Orientation	T	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_nxz

Return orientation object from n_x and n_z vectors

Information

It is assumed that the two input vectors n_x and n_z are resolved in frame 1 and are directed along the x and z axis of frame 2 (i.e., n_x and n_z are orthogonal to each other) The function returns the orientation object T to rotate from frame 1 to frame 2.

The function is robust in the sense that it returns always an orientation object T, even if n_z is not orthogonal to n_x. This is performed in the following way:

If n_x and n_z are not orthogonal to each other, first a unit vector e_z is determined that is orthogonal to n_x and is lying in the plane spanned by n_x and n_z. If n_x and n_z are parallel or nearly parallel to each other, a vector e_z is selected arbitrarily such that n_x and e_z are orthogonal to each other.

Inputs

Type	Name	Default	Description
Real	n_x[3]		Vector in direction of x-axis of frame 2, resolved in frame 1
Real	n_z[3]		Vector in direction of z-axis of frame 2, resolved in frame 1

Outputs

Type	Name	Description
Orientation	T	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_T

Return orientation object R from transformation matrix T

Inputs

Type	Name	Default	Description
Real	T[3, 3]		Transformation matrix to transform vector from frame 1 to frame 2 (v2=T*v1)

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_T_inv

Return orientation object R from inverse transformation matrix T_inv

Inputs

Type	Name	Default	Description
Real	T_inv[3, 3]		Inverse transformation matrix to transform vector from frame 2 to frame 1 (v1=T_inv*v2)

Outputs

Type	Name	Description
Orientation	R	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.from_Q

Return orientation object T from quaternion orientation object Q

Inputs

Type	Name	Default	Description
Orientation	Q		Quaternions orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Orientation	T	Orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_T

Return transformation matrix T from orientation object R

Inputs

Type	Name	Default	Description
Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	T[3, 3]	Transformation matrix to transform vector from frame 1 to frame 2 (v2=T*v1)

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_T_inv

Return inverse transformation matrix T_inv from orientation object R

Inputs

Type	Name	Default	Description

Orientation	R		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	T_inv[3, 3]	Inverse transformation matrix to transform vector from frame 2 into frame 1 (v1=T_inv*v2)

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_Q

Return quaternion orientation object Q from orientation object T

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2
Orientation	Q_guess	Quaternions.nullRotation()	Guess value for output Q (there are 2 solutions; the one closer to Q_guess is used

Outputs

Type	Name	Description
Orientation	Q	Quaternions orientation object to rotate frame 1 into frame 2

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_vector

Map rotation object into vector

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description
Real	vec[9]	Elements of T in one vector

Modelica.Mechanics.MultiBody.Frames.TransformationMatrices.to_exy

Map rotation object into e_x and e_y vectors of frame 2, resolved in frame 1

Inputs

Type	Name	Default	Description
Orientation	T		Orientation object to rotate frame 1 into frame 2

Outputs

Type	Name	Description

Real	exy[3, 2]	= [e_x, e_y] where e_x and e_y are axes unit vectors of frame 2, resolved in frame 1

Modelica.Mechanics.MultiBody.Interfaces

Connectors and partial models for 3-dim. mechanical components

Information

This package contains connectors and partial models (i.e. models that are only used to build other models) of the MultiBody library.

Package Content

Name	Description
Frame	Coordinate system fixed to the component with one cut-force and cut-torque (no icon)
Frame_a	Coordinate system fixed to the component with one cut-force and cut-torque (filled rectangular icon)
Frame_b	Coordinate system fixed to the component with one cut-force and cut-torque (non-filled rectangular icon)
Frame_resolve	Coordinate system fixed to the component used to express in which coordinate system a vector is resolved (non-filled rectangular icon)
FlangeWithBearing	Connector consisting of 1-dim. rotational flange and its bearing frame
FlangeWithBearingAdaptor	Adaptor to allow direct connections to the sub-connectors of FlangeWithBearing
PartialTwoFrames	Base model for components providing two frame connectors + outer world + assert to guarantee that the component is connected
PartialTwoFramesDoubleSize	Base model for components providing two frame connectors + outer world + assert to guarantee that the component is connected (default icon size is factor 2 larger as usual)
PartialOneFrame_a	Base model for components providing one frame_a connector + outer world + assert to guarantee that the component is connected
PartialOneFrame_b	Base model for components providing one frame_b connector + outer world + assert to guarantee that the component is connected
PartialElementaryJoint	Base model for elementary joints (has two frames + outer world + assert to guarantee that the joint is connected)
PartialForce	Base model for force elements (provide frame_b.f and frame_b.t in subclasses)
PartialLineForce	Base model for line force elements
PartialAbsoluteSensor	Base model to measure an absolute frame variable
PartialRelativeSensor	Base model to measure a relative variable between two frames
PartialCutForceSensor	Base model to measure the cut force and/or torque between two frames
PartialVisualizer	Base model for visualizers (has a frame_a on the left side + outer world + assert to guarantee that the component is connected)

Modelica.Mechanics.MultiBody.Interfaces.Frame

Coordinate system fixed to the component with one cut-force and cut-torque (no icon)

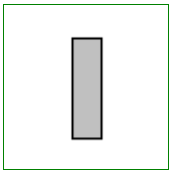
Information

Basic definition of a coordinate system that is fixed to a mechanical component. In the origin of the coordinate system the cut-force and the cut-torque is acting. This component has no icon definition and is only used by inheritance from frame connectors to define different icons.

Contents

Type	Name	Description
Position	r_0[3]	Position vector from world frame to the connector frame origin, resolved in world frame [m]
Orientation	R	Orientation object to rotate the world frame into the connector frame
flow Force	f[3]	Cut-force resolved in connector frame [N]
flow Torque	t[3]	Cut-torque resolved in connector frame [N.m]

Modelica.Mechanics.MultiBody.Interfaces.Frame_a



Coordinate system fixed to the component with one cut-force and cut-torque (filled rectangular icon)

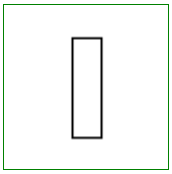
Information

Basic definition of a coordinate system that is fixed to a mechanical component. In the origin of the coordinate system the cut-force and the cut-torque is acting. This component has a filled rectangular icon.

Contents

Type	Name	Description
Position	r_0[3]	Position vector from world frame to the connector frame origin, resolved in world frame [m]
Orientation	R	Orientation object to rotate the world frame into the connector frame
flow Force	f[3]	Cut-force resolved in connector frame [N]
flow Torque	t[3]	Cut-torque resolved in connector frame [N.m]

Modelica.Mechanics.MultiBody.Interfaces.Frame_b



Coordinate system fixed to the component with one cut-force and cut-torque (non-filled rectangular icon)

Information

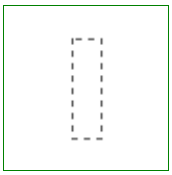
Basic definition of a coordinate system that is fixed to a mechanical component. In the origin of the coordinate system the cut-force and the cut-torque is acting. This component has a non-filled rectangular icon.

Contents

Type	Name	Description
Position	r_0[3]	Position vector from world frame to the connector frame origin, resolved in world frame [m]
Orientation	R	Orientation object to rotate the world frame into the connector frame

flow Force	f[3]	Cut-force resolved in connector frame [N]
flow Torque	t[3]	Cut-torque resolved in connector frame [N.m]

Modelica.Mechanics.MultiBody.Interfaces.Frame_resolve



Coordinate system fixed to the component used to express in which coordinate system a vector is resolved (non-filled rectangular icon)

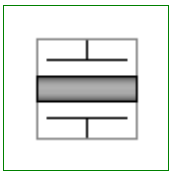
Information

Basic definition of a coordinate system that is fixed to a mechanical component. In the origin of the coordinate system the cut-force and the cut-torque is acting. This coordinate system is used to express in which coordinate system a vector is resolved. A component that uses a Frame_resolve connector has to set the cut-force and cut-torque of this frame to zero. When connecting from a Frame_resolve connector to another frame connector, by default the connecting line has line style "dotted". This component has a non-filled rectangular icon.

Contents

Type	Name	Description
Position	r_0[3]	Position vector from world frame to the connector frame origin, resolved in world frame [m]
Orientation	R	Orientation object to rotate the world frame into the connector frame
flow Force	f[3]	Cut-force resolved in connector frame [N]
flow Torque	t[3]	Cut-torque resolved in connector frame [N.m]

Modelica.Mechanics.MultiBody.Interfaces.FlangeWithBearing



Connector consisting of 1-dim. rotational flange and its bearing frame

Information

This hierarchical connector models a 1-dim. rotational flange connector and its optional bearing defined by a 3-dim. frame connector. If a connection to the subconnectors should be clearly visible, connect first an instance of FlangeWithBearingAdaptor to the FlangeWithBearing connector.

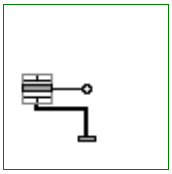
Parameters

Type	Name	Default	Description
Boolean	includeBearingConnector	false	= true, if bearing frame connector is present, otherwise not present

Contents

Type	Name	Description
Boolean	includeBearingConnector	= true, if bearing frame connector is present, otherwise not present
Flange_a	flange	1-dim. rotational flange
Frame	bearingFrame	3-dim. frame in which the 1-dim. shaft is mounted

Modelica.Mechanics.MultiBody.Interfaces.FlangeWithBearingAdaptor



Adaptor to allow direct connections to the sub-connectors of FlangeWithBearing

Information

Adaptor object to make a more visible connection to the flange and frame subconnectors of a FlangeWithBearing connector.

Parameters

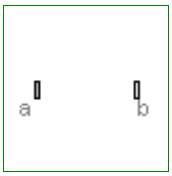
Type	Name	Default	Description
Boolean	includeBearingConnector	false	= true, if bearing frame connector is present, otherwise not present

Connectors

Type	Name	Description
FlangeWithBearing	flangeAndFrame	Compound connector consisting of 1-dim. rotational flange and 3-dim. frame mounting
Flange_b	flange	1-dim. rotational flange
Frame_a	frame	3-dim. frame in which the 1-dim. shaft is mounted

---

Modelica.Mechanics.MultiBody.Interfaces.PartialTwoFrames



Base model for components providing two frame connectors + outer world + assert to guarantee that the component is connected

Information

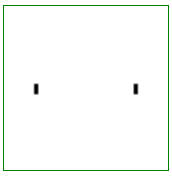
This partial model provides two frame connectors, access to the world object and an assert to check that both frame connectors are connected. Therefore, inherit from this partial model if the two frame connectors are needed and if the two frame connectors should be connected for a correct model.

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Interfaces.PartialTwoFramesDoubleSize



Base model for components providing two frame connectors + outer world + assert to guarantee that the component is connected (default icon size is factor 2 larger as usual)

Information

This partial model provides two frame connectors, access to the world object and an assert to check that both frame connectors are connected. Therefore, inherit from this partial model if the two frame connectors are needed and if the two frame connectors should be connected for a correct model.

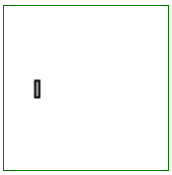
When dragging "PartialTwoFrames", the default size is a factor of two larger as usual. This partial model is used by the Joint.Assemblies joint aggregation models.

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Interfaces.PartialOneFrame_a



Base model for components providing one frame_a connector + outer world + assert to guarantee that the component is connected

Information

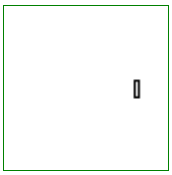
This partial model provides one frame_a connector, access to the world object and an assert to check that the frame_a connector is connected. Therefore, inherit from this partial model if the frame_a connector is needed and if this connector should be connected for a correct model.

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Interfaces.PartialOneFrame_b



Base model for components providing one frame_b connector + outer world + assert to guarantee that the component is connected

Information

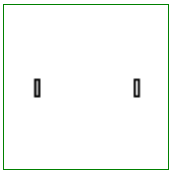
This partial model provides one frame_b connector, access to the world object and an assert to check that the frame_b connector is connected. Therefore, inherit from this partial model if the frame_b connector is needed and if this connector should be connected for a correct model.

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Interfaces.PartialElementaryJoint



Base model for elementary joints (has two frames + outer world + assert to guarantee that the joint is connected)

Information

All **elementary joints** should inherit from this base model, i.e., joints that are directly defined by equations, provided they compute either the rotation object of frame_b from the rotation object of frame_a and from relative quantities (or vice versa), or there is a constraint equation between the rotation objects of the two frames. In other cases, a joint object should inherit from **Interfaces.PartialTwoFrames** (e.g., joint Spherical, because there is no constraint between the rotation objects of frame_a and frame_b or joint Cylindrical because it is not an elementary joint).

This partial model provides two frame connectors, a "defineBranch" between frame_a and frame_b, access

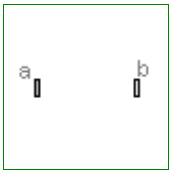
to the world object and an assert to check that both frame connectors are connected.

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the joint with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the joint with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Interfaces.PartialForce



Base model for force elements (provide frame_b.f and frame_b.t in subclasses)

Information

All **3-dimensional force** and **torque elements** should be based on this superclass. This model defines frame_a and frame_b, computes the relative translation and rotation between the two frames and calculates the cut-force and cut-torque at frame_a by a force and torque balance from the cut-force and cut-torque at frame_b. As a result, in a subclass, only the relationship between the cut-force and cut-torque at frame_b has to be defined as a function of the following relative quantities:

```
r_rel_b[3]: Position vector from origin of frame_a to origin
            of frame_b, resolved in frame_b
R_rel     : Relative orientation object to rotate from frame_a to frame_b
```

Assume that force f = {100,0,0} should be applied on the body to which this force element is attached at frame_b, then the definition should be:

```
model Constant_x_Force
   extends Modelica.Mechanics.MultiBody.Interfaces.PartialForce;
equation
   frame_b.f = {-100, 0, 0};
   frame_b.t = zeros(3);
end Constant_x_Force;
```

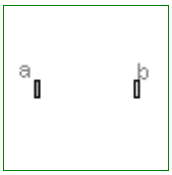
Note, that frame_b.f and frame_b.t are flow variables and therefore the negative value of frame_b.f and frame_b.t is acting at the part to which this force element is connected.

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the joint with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the joint with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Interfaces.PartialLineForce



Base model for line force elements

Information

All **line force** elements should be based on this base model. This model defines frame_a and frame_b, computes the relative distance **s** and provides the force and torque balance of the cut-forces and cut-torques at frame_a and frame_b, respectively. In sub-models, only the line force **f**, acting at frame_b on the line from

frame_a to frame_b, as a function of the relative distance **s** and its derivative **der**(**s**) has to be defined. Example:

```
model Spring
   parameter Real c "spring constant",
   parameter Real s_unstretched "unstretched spring length";
   extends Modelica.Mechanics.MultiBody.Interfaces.PartialLineForce;
equation
   f = c*(s-s_unstretched);
end Spring;
```

Parameters

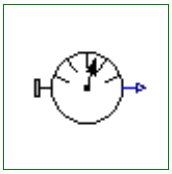
Type	Name	Default	Description
Advanced			
Position	s_small	1.E-6	Prevent zero-division if relative distance s=0 [m]

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the force element with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the force element with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Interfaces.PartialAbsoluteSensor



Base model to measure an absolute frame variable

Information

This is the base class of a 3-dim. mechanics component with one frame and one output port in order to measure an absolute quantity in the frame connector and to provide the measured signal as output for further processing with the blocks of package Modelica.Blocks.

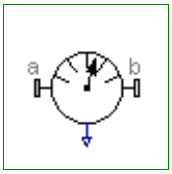
Parameters

Type	Name	Default	Description
Integer	n_out	1	Number of output signals

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system from which absolute quantities are provided as output signals
output RealOutput	y[n_out]	Measured data as signal vector

Modelica.Mechanics.MultiBody.Interfaces.PartialRelativeSensor



Base model to measure a relative variable between two frames

Information

This is a base class for 3-dim. mechanical components with two frames and one output port in order to measure relative quantities between the two frames or the cut-forces/torques in the frame and to provide the

measured signals as output for further processing with the blocks of package Modelica.Blocks.

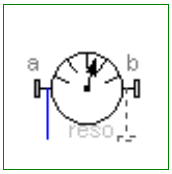
Parameters

Type	Name	Default	Description
Integer	n_out	1	Number of output signals

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system a
Frame_b	frame_b	Coordinate system b
output RealOutput	y[n_out]	Measured data as signal vector

Modelica.Mechanics.MultiBody.Interfaces.PartialCutForceSensor



Base model to measure the cut force and/or torque between two frames

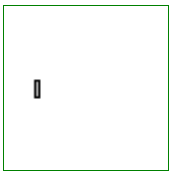
Information

This is a base class for 3-dim. mechanical components with two frames and one output port in order to measure the cut-force and/or cut-torque acting between the two frames and to provide the measured signals as output for further processing with the blocks of package Modelica.Blocks.

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the output signals are resolved in this frame (cut-force/-torque are set to zero)

Modelica.Mechanics.MultiBody.Interfaces.PartialVisualizer



Base model for visualizers (has a frame_a on the left side + outer world + assert to guarantee that the component is connected)

Information

This partial model provides one frame_a connector, access to the world object and an assert to check that the frame_a connector is connected. It is used by inheritance from all visualizer objects.

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system in which visualization data is resolved

---

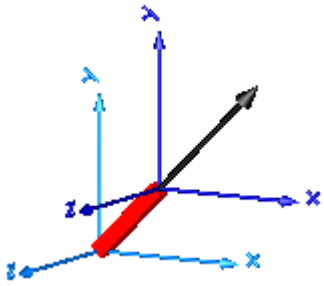
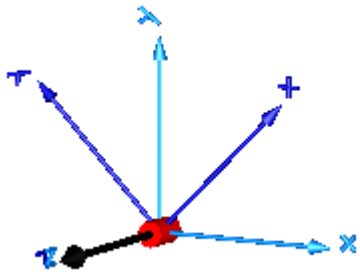
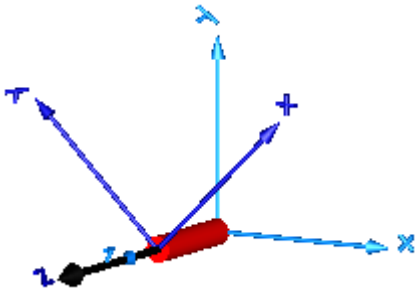
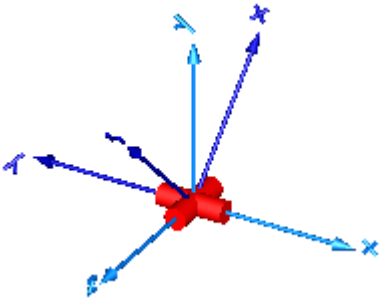
Modelica.Mechanics.MultiBody.Joints

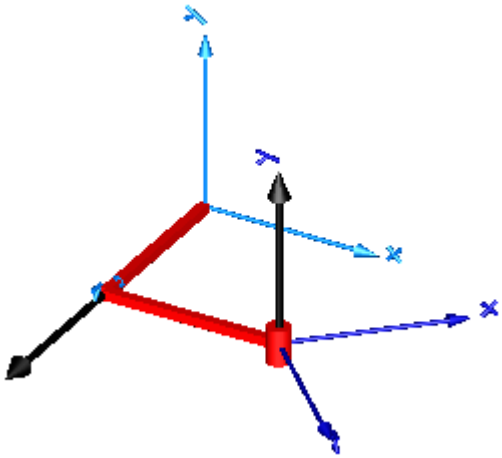
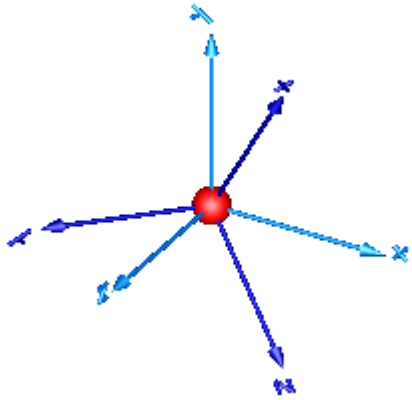
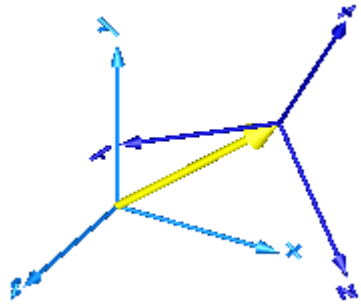
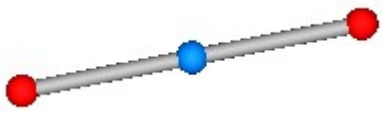
Components that constrain the motion between two frames

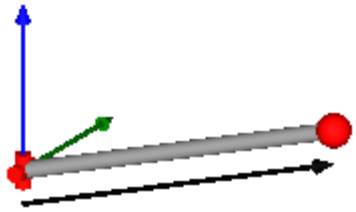
Information

This package contains **joint components**, that is, idealized, massless elements that constrain the motion between frames. In subpackage **Assemblies** aggregation joint components are provided to handle kinematic loops analytically (this means that non-linear systems of equations occuring in these joint aggregations are analytically solved, i.e., robustly and efficiently).

Content

Model	*Description*
Prismatic ActuatedPrismatic	Prismatic joint and actuated prismatic joint (1 translational degree-of-freedom, 2 potential states) 
Revolute ActuatedRevolute	Revolute and actuated revolute joint (1 rotational degree-of-freedom, 2 potential states) 
Cylindrical	Cylindrical joint (2 degrees-of-freedom, 4 potential states) 
Universal	Universal joint (2 degrees-of-freedom, 4 potential states) 
Planar	Planar joint (3 degrees-of-freedom, 6 potential states)

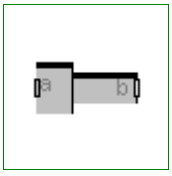
	
Spherical	Spherical joint (3 constraints and no potential states, or 3 degrees-of-freedom and 3 states) 
FreeMotion	Free motion joint (6 degrees-of-freedom, 12 potential states) 
SphericalSpherical	Spherical - spherical joint aggregation (1 constraint, no potential states) with an optional point mass in the middle 
UniversalSpherical	Universal - spherical joint aggregation (1 constraint, no potential states)

	
GearConstraint	Ideal 3-dim. gearbox (arbitrary shaft directions)
MultiBody.Joints.Assemblies	**Package** of joint aggregations for analytic loop handling.

Package Content

Name	Description
Prismatic	Prismatic joint (1 translational degree-of-freedom, 2 potential states)
ActuatedPrismatic	Actuated prismatic joint (1 translational degree-of-freedom, 2 potential states)
Revolute	Revolute joint (1 rotational degree-of-freedom, 2 potential states)
ActuatedRevolute	Actuated revolute joint (1 rotational degree-of-freedom, 2 potential states)
Cylindrical	Cylindrical joint (2 degrees-of-freedom, 4 potential states)
Universal	Universal joint (2 degrees-of-freedom, 4 potential states)
Planar	Planar joint (3 degrees-of-freedom, 6 potential states)
Spherical	Spherical joint (3 constraints and no potential states, or 3 degrees-of-freedom and 3 states)
FreeMotion	Free motion joint (6 degrees-of-freedom, 12 potential states)
SphericalSpherical	Spherical - spherical joint aggregation (1 constraint, no potential states) with an optional point mass in the middle
UniversalSpherical	Universal - spherical joint aggregation (1 constraint, no potential states)
GearConstraint	Ideal 3-dim. gearbox (arbitrary shaft directions)
Assemblies	Joint aggregations for analytic loop handling

Modelica.Mechanics.MultiBody.Joints.Prismatic



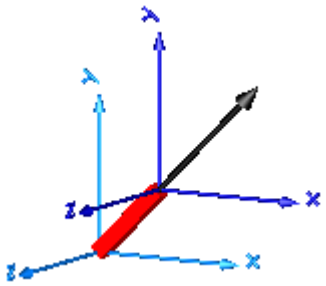
Prismatic joint (1 translational degree-of-freedom, 2 potential states)

Information

Joint where frame_b is translated along axis n which is fixed in frame_a. The two frames coincide when "s + s_offset = 0", where "s_offset" is a parameter with a zero default and "s" is the relative distance.

In the "Advanced" menu it can be defined via parameter **enforceStates** that the relative distance "s" and its derivative shall be definitely used as states (this means that the Modelica attributes stateSelect=StateSelect.always are set on these variables). The states are usually selected automatically. In certain situations, especially when closed kinematic loops are present, it might be slightly more efficient, when using the "enforceStates" setting.

In the following figure the animation of a prismatic joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint. The black arrow is parameter vector "n" defining the translation axis (here: n = {1,1,0}).



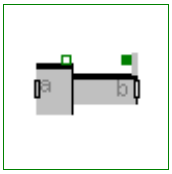
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Axis	n	{1,0,0}	Axis of translation resolved in frame_a (= same as in frame_b)
Position	s_offset	0	Relative distance offset (distance between frame_a and frame_b = s_offset + s) [m]
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
Position	s_start	0	Initial value of distance (fixed or guess value) [m]
Velocity	v_start	0	Initial value of relative velocity v = der(s) [m/s]
Acceleration	a_start	0	Initial value of relative acceleration a = der(v) [m/s2]
Animation			
if animation = true			
Axis	boxWidthDirection	{0,1,0}	Vector in width direction of box, resolved in frame_a
Distance	boxWidth	world.defaultJointWidth	Width of prismatic joint box [m]
Distance	boxHeight	boxWidth	Height of prismatic joint box [m]
Color	boxColor	Modelica.Mechanics.MultiBody...	Color of prismatic joint box
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if generalized variables (s,v) shall be used as states (StateSelect.always)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the joint with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the joint with one cut-force and cut-torque

Modelica.Mechanics.MultiBody.Joints.ActuatedPrismatic



Actuated prismatic joint (1 translational degree-of-freedom, 2 potential states)

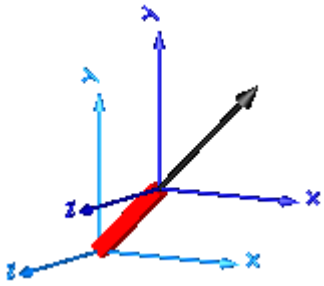
Information

Joint where frame_b is translated along axis n which is fixed in frame_a. The two frames coincide when "s + s_offset = 0", where "s_offset" is a parameter with a zero default and "s" is the relative distance.

The prismatic joint has two additional 1-dimensional mechanical flanges (flange "axis" represents the driving flange and flange "bearing" represents the bearing) where it can be driven with elements of the Modelica.Mechanics.Translational library.

In the "Advanced" menu it can be defined via parameter **enforceStates** that the relative distance "s" and its derivative shall be definitely used as states (this means that the Modelica attributes stateSelect=StateSelect.always are set on these variables). The states are usually selected automatically. In certain situations, especially when closed kinematic loops are present, it might be slightly more efficient, when using the "enforceStates" setting.

In the following figure the animation of an actuated prismatic joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint. The black arrow is parameter vector "n" defining the translation axis (here: n = {1,1,0}).



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Axis	n	{1,0,0}	Axis of translation resolved in frame_a (= same as in frame_b)
Position	s_offset	0	Relative distance offset (distance between frame_a and frame_b = s_offset + s) [m]
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody..	Type of initialization (defines usage of start values below)
Position	s_start	0	Initial value of distance (fixed or guess value) [m]
Velocity	v_start	0	Initial value of relative velocity v = der(s) [m/s]
Acceleration	a_start	0	Initial value of relative acceleration a = der(v) [m/s2]
Animation			
if animation = true			
Axis	boxWidthDirection	{0,1,0}	Vector in width direction of box, resolved in frame_a

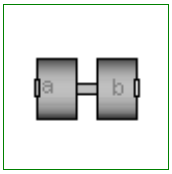
Distance	boxWidth	world.defaultJointWidth	Width of prismatic joint box [m]
Distance	boxHeight	boxWidth	Height of prismatic joint box [m]
Color	boxColor	Modelica.Mechanics.MultiBody...	Color of prismatic joint box
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if generalized variables (s,v) shall be used as states (StateSelect.always)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the joint with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the joint with one cut-force and cut-torque
Flange_a	axis	1-dim. translational flange that drives the joint
Flange_b	bearing	1-dim. translational flange of the drive bearing

---

Modelica.Mechanics.MultiBody.Joints.Revolute



Revolute joint (1 rotational degree-of-freedom, 2 potential states)

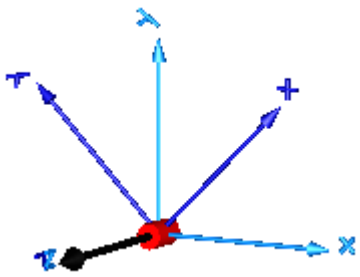
Information

Joint where frame_b rotates around axis n which is fixed in frame_a. The two frames coincide when "phi + phi_offset = 0", where "phi_offset" is a parameter with a zero default and "phi" is the rotation angle.

In the "Advanced" menu it can be defined via parameter **enforceStates** that the rotation angle "phi" and its derivative shall be definitely used as states (this means that the Modelica attributes stateSelect=StateSelect.always are set on these variables). The states are usually selected automatically. In certain situations, especially when closed kinematic loops are present, it might be slightly more efficient, when using the "enforceStates" setting.

If a **planar loop** is present, e.g., consisting of 4 revolute joints where the joint axes are all parallel to each other, then there is no longer a unique mathematical solution and the symbolic algorithms will fail. Usually, an error message will be printed pointing out this situation. In this case, parameter **planarCutJoint** in the "Advanced" menu of one of the revolute joints has to be set to **true**. The effect is that from the 5 constraints of a usual revolute joint, 3 constraints are removed and replaced by appropriate known variables (e.g., the force in the direction of the axis of rotation is treated as known with value equal to zero; for standard revolute joints, this force is an unknown quantity).

In the following figure the animation of a revolute joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint. The black arrow is parameter vector "n" defining the translation axis (here: n = {0,0,1}, phi_start = 45^o).



Parameters

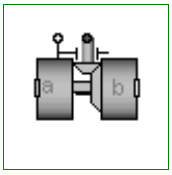
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show axis as cylinder)
Axis	n	{0,0,1}	Axis of rotation resolved in frame_a (= same as in frame_b)
Angle_deg	phi_offset	0	Relative angle offset (angle = phi + from_deg(phi_offset)) [deg]
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
Angle_deg	phi_start	0	Initial value of rotation angle phi (fixed or guess value) [deg]
AngularVelocity_degs	w_start	0	Initial value of relative angular velocity w = der(phi) [deg/s]
AngularAcceleration_degs2	a_start	0	Initial value of relative angular acceleration a = der(w) [deg/s2]
Animation			
if animation = true			
Distance	cylinderLength	world.defaultJointLength	Length of cylinder representing the joint axis [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinder representing the joint axis [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinder representing the joint axis
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if generalized variables (phi,w) shall be used as states (StateSelect.always)
Boolean	planarCutJoint	false	= true, if joint shall be used as cut-joint in a planar loop

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the joint with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the joint with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Joints.ActuatedRevolute



Actuated revolute joint (1 rotational degree-of-freedom, 2 potential states)

Information

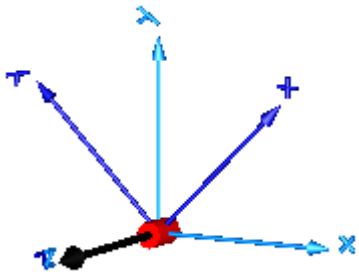
Joint where frame_b rotates around axis n which is fixed in frame_a. The two frames coincide when "phi + phi_offset = 0", where "phi_offset" is a parameter with a zero default and "phi" is the rotation angle.

The revolute joint has two additional 1-dimensional mechanical flanges (flange "axis" represents the driving flange and flange "bearing" represents the bearing) where it can be driven with elements of the Modelica.Mechanics.Rotational library.

In the "Advanced" menu it can be defined via parameter **enforceStates** that the rotation angle "phi" and its derivative shall be definitely used as states (this means that the Modelica attributes stateSelect=StateSelect.always are set on these variables). The states are usually selected automatically. In certain situations, especially when closed kinematic loops are present, it might be slightly more efficient, when using the "enforceStates" setting.

If a **planar loop** is present, e.g., consisting of 4 revolute joints where the joint axes are all parallel to each other, then there is no longer a unique mathematical solution and the symbolic algorithms will fail. Usually, an error message will be printed pointing out this situation. In this case, parameter **planarCutJoint** in the "Advanced" menu of one of the revolute joints has to be set to **true**. The effect is that from the 5 constraints of a usual revolute joint, 3 constraints are removed and replaced by appropriate known variables (e.g., the force in the direction of the axis of rotation is treated as known with value equal to zero; for standard revolute joints, this force is an unknown quantity).

In the following figure the animation of an actuated revolute joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint. The black arrow is parameter vector "n" defining the translation axis (here: n = {0,0,1}, phi_start = 45^o).



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show axis as cylinder)
Axis	n	{0,0,1}	Axis of rotation resolved in frame_a (= same as in frame_b)
Angle_deg	phi_offset	0	Relative angle offset (angle = phi + from_deg(phi_offset)) [deg]
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
Angle_deg	phi_start	0	Initial value of rotation angle phi (fixed or guess value) [deg]
AngularVelocity_degs	w_start	0	Initial value of relative angular velocity w = der(phi) [deg/s]
AngularAcceleration_degs2	a_start	0	Initial value of relative angular acceleration a = der(w) [deg/s2]
Animation			
if animation = true			
Distance	cylinderLength	world.defaultJointLength	Length of cylinder representing

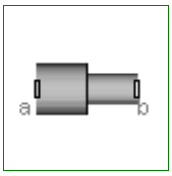
			the joint axis [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinder representing the joint axis [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinder representing the joint axis
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if generalized variables (phi,w) shall be used as states (StateSelect.always)
Boolean	planarCutJoint	false	= true, if joint shall be used as cut-joint in a planar loop

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the joint with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the joint with one cut-force and cut-torque
Flange_a	axis	1-dim. rotational flange that drives the joint
Flange_b	bearing	1-dim. rotational flange of the drive bearing

---

Modelica.Mechanics.MultiBody.Joints.Cylindrical



Cylindrical joint (2 degrees-of-freedom, 4 potential states)

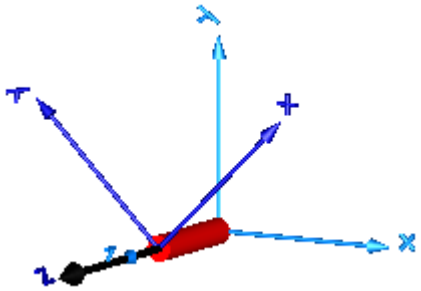
Information

Joint where frame_b rotates around and translates along axis n which is fixed in frame_a. The two frames coincide when "revolute.phi=0" and "prismatic.s=0". This joint has the following potential states;

- The relative angle revolute.phi [rad] around axis n,
- the relative distance prismatic.s [m] along axis n,
- the relative angular velocity revolute.w [rad/s] (= der(revolute.phi)) and
- the relative velocity prismatic.v [m/s] (= der(prismatic.s)).

They are used as candidates for automatic selection of states from the tool. This may be enforced by setting "enforceStates=**true**" in the **Advanced** menu (this means that the Modelica attributes stateSelect=StateSelect.always are set on these variables). The states are usually selected automatically. In certain situations, especially when closed kinematic loops are present, it might be slightly more efficient, when using the "enforceStates" setting.

In the following figure the animation of a cylindrical joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint. The black arrow is parameter vector "n" defining the cylinder axis (here: n = {0,0,1}).



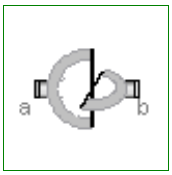
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show cylinder)
Axis	n	{1,0,0}	Cylinder axis resolved in frame_a (= same as in frame_b)
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
Angle_deg	phi_start	0	Initial value of rotation angle phi (fixed or guess value) [deg]
Position	s_start	0	Initial value of relative distance (fixed or guess value) [m]
AngularVelocity_degs	w_start	0	Initial value of relative angular velocity w = der(phi) [deg/s]
Velocity	v_start	0	Initial value of relative velocity v = der(s) [m/s]
Acceleration	a_start	0	Initial value of relative acceleration a = der(v) [m/s2]
AngularAcceleration_degs2	wd_start	0	Initial value of relative angular acceleration wd = der(w) [deg/s2]
Animation			
if animation = true			
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinder [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinder
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if generalized variables shall be used as states (StateSelect.always)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

Modelica.Mechanics.MultiBody.Joints.Universal



Universal joint (2 degrees-of-freedom, 4 potential states)

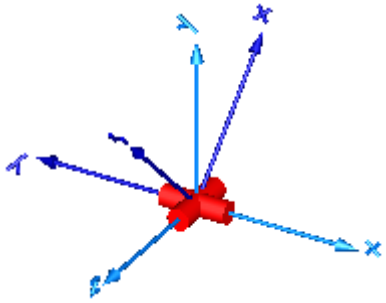
Information

Joint where frame_a rotates around axis n_a which is fixed in frame_a and frame_b rotates around axis n_b which is fixed in frame_b. The two frames coincide when "revolute_a.phi=0" and "revolute_b.phi=0". This joint has the following potential states;

- The relative angle revolute_a.phi [rad] around axis n_a,
- the relative angle revolute_b.phi [rad] around axis n_b,
- the relative angular velocity revolute_a.w [rad/s] (= der(revolute_a.phi)) and
- the relative angular velocity revolute_b.w [rad/s] (= der(revolute_b.phi)).

They are used as candidates for automatic selection of states from the tool. This may be enforced by setting "enforceStates=**true**" in the **Advanced** menu (this means that the Modelica attributes stateSelect=StateSelect.always are set on these variables). The states are usually selected automatically. In certain situations, especially when closed kinematic loops are present, it might be slightly more efficient, when using the "enforceStates" setting.

In the following figure the animation of a universal joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint (here: n_a = {0,0,1}, n_b = {0,1,0}, phi_start_a = 90^o, phi_start_b = 45^o).



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Axis	n_a	{1,0,0}	Axis of revolute joint 1 resolved in frame_a
Axis	n_b	{0,1,0}	Axis of revolute joint 2 resolved in frame_b
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
Angle_deg	phi_start_a	0	Initial value of rotation angle at frame_a (fixed or guess value) [deg]
Angle_deg	phi_start_b	0	Initial value of rotation angle at frame_b (fixed or guess value) [deg]
AngularVelocity_degs	w_start_a	0	Initial value of derivative of

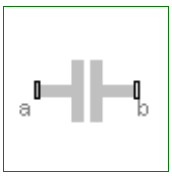
			rotation angle at frame_a [deg/s]
AngularVelocity_degs	w_start_b	0	Initial value of derivative of rotation angle at frame_b [deg/s]
AngularAcceleration_degs2	a_start_a	0	Initial value of second derivative of rotation angle at frame_a [deg/s2]
AngularAcceleration_degs2	a_start_b	0	Initial value of second derivative of rotation angle at frame_b [deg/s2]
Animation			
if animation = true			
Distance	cylinderLength	world.defaultJointLength	Length of cylinders representing the joint axes [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinders representing the joint axes [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinders representing the joint axes
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if generalized variables shall be used as states (StateSelect.always)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Joints.Planar



Planar joint (3 degrees-of-freedom, 6 potential states)

Information

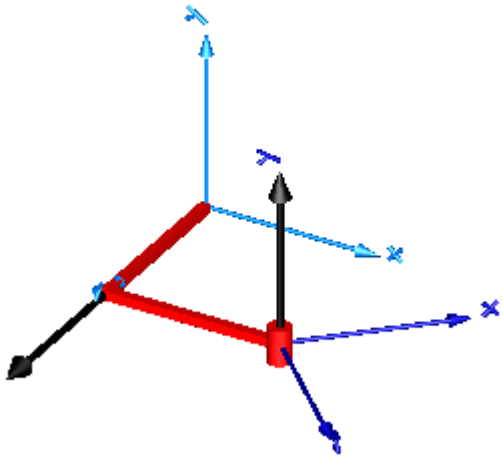
Joint where frame_b can move in a plane and can rotate around an axis orthogonal to the plane. The plane is defined by vector n which is perpendicular to the plane and by vector n_x, which points in the direction of the x-axis of the plane. frame_a and frame_b coincide when prismatic_a.s=0, prismatic_b=0 and revolute.phi=0. This joint has the following potential states:

- the relative distance prismatic_x.s [m] along axis n_x,
- the relative distance prismatic_y.s [m] along axis n_y = cross(n,n_x),
- the relative angle revolute.phi [rad] around axis n,
- the relative velocity prismatic_x.v [m/s] (= der(prismatic_x.s)).
- the relative velocity prismatic_y.v [m/s] (= der(prismatic_y.s)).
- the relative angular velocity revolute.w [rad/s] (= der(revolute.phi))

The potential states are used as candidates for automatic selection of states from the tool. This may be enforced by setting "enforceStates=**true**" in the **Advanced** menu (this means that the Modelica attributes stateSelect=StateSelect.always are set on these variables). The states are usually selected automatically. In certain situations, especially when closed kinematic loops are present, it might be slightly more efficient,

when using the "enforceStates" setting.

In the following figure the animation of a planar joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint. The black arrows are parameter vectors "n" and "n_x" (here: n = {0,1,0}, n_x = {0,0,1}, s_start_x = 0.5, s_start_y = 0.5, phi_start = 45^o).



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Axis	n	{0,0,1}	Axis orthogonal to unconstrained plane, resolved in frame_a (= same as in frame_b)
Axis	n_x	{1,0,0}	Vector in direction of x-axis of plane, resolved in frame_a (n_x shall be orthogonal to n)
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
Position	s_start_x	0	Initial value of x-distance (along n_x; fixed or guess value) [m]
Position	s_start_y	0	Initial value of y-distance (along cross(n,n_x); fixed or guess value) [m]
Angle_deg	phi_start	0	Initial value of rotation angle along n (fixed or guess value) [deg]
Velocity	v_start_x	0	Initial value of derivative of x-distance [m/s]
Velocity	v_start_y	0	Initial value of derivative of y-distance [m/s]
AngularVelocity_degs	w_start	0	Initial value of derivative of rotation angle [deg/s]
Acceleration	a_start_x	0	Initial value of second derivative of x-distance [m/s2]
Acceleration	a_start_y	0	Initial value of second derivative of y-distance [m/s2]

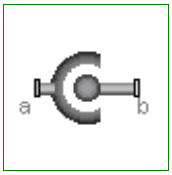
AngularAcceleration_degs2	wd_start	0	Initial value of second derivative of rotation angle [deg/s2]
Animation			
if animation = true			
Distance	cylinderLength	world.defaultJointLength	Length of revolute cylinder [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of revolute cylinder [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of revolute cylinder
Distance	boxWidth	0.3*cylinderDiameter	Width of prismatic joint boxes [m]
Distance	boxHeight	boxWidth	Height of prismatic joint boxes [m]
Color	boxColor	Modelica.Mechanics.MultiBody...	Color of prismatic joint boxes
Advanced			
Boolean	enforceStates	false	= true, if generalized variables (s,phi,v,w) shall be used as states (StateSelect.always)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Joints.Spherical



Spherical joint (3 constraints and no potential states, or 3 degrees-of-freedom and 3 states)

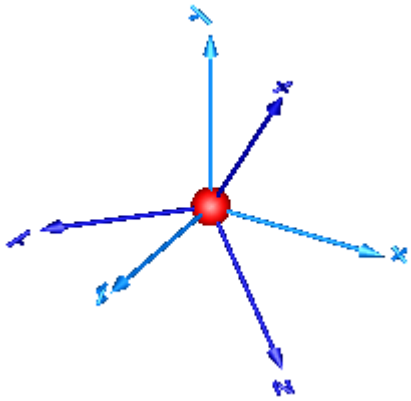
Information

Joint with **3 constraints** that define that the origin of frame_a and the origin of frame_b coincide. By default this joint defines only the 3 constraints without any potential states. If parameter **enforceStates** is set to **true** in the "Advanced" menu, three states are introduced. Depending on parameter **useQuaternions** these are either quaternions and the relative angular velocity or 3 angles and the angle derivatves. In the latter case the orientation of frame_b is computed by rotating frame_a along the axes defined in parameter vector "sequence_angleStates" (default = {1,2,3}, i.e., the Cardan angle sequence) around the angles used as states. For example, the default is to rotate the x-axis of frame_a around angles[1], the new y-axis around angles[2] and the new z-axis around angles[3], arriving at frame_b. If angles are used as states there is the slight disadvantage that a singular configuration is present leading to a divison by zero.

If this joint is used in a **chain** structure, a Modelica translator has to select orientation coordinates of a body as states, if the default setting is used. It is usually better to use relative coordinates in the spherical joint as states, and therefore in this situation parameter enforceStates might be set to **true**.

If this joint is used in a **loop** structure, the default setting results in a **cut-joint** that breaks the loop in independent kinematic pieces, hold together by the constraints of this joint. As a result, a Modelica translator will first try to select 3 generalized coordinates in the joints of the remaining parts of the loop and their first derivative as states and if this is not possible, e.g., because there are only spherical joints in the loop, will select coordinates from a body of the loop as states.

In the following figure the animation of a spherical joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint. (here: angles_start = {45, 45, 45}^{o}).



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show sphere)
if animation = true			
Distance	sphereDiameter	world.defaultJointLength	Diameter of sphere representing the spherical joint [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of sphere representing the spherical joint
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
RotationSequence	sequence_start	{1,2,3}	Sequence of rotations to rotate frame_a into frame_b at initial time
Angle_deg	angles_start[3]	{0,0,0}	Initial values of angles to rotate frame_a around 'sequence_start' axes into frame_b [deg]
AngularVelocity_degs	w_rel_a_start[3]	{0,0,0}	Initial values of angular velocity of frame_b with respect to frame_a, resolved in frame_a [deg/s]
AngularAcceleration_degs2	z_rel_a_start[3]	{0,0,0}	Initial values of angular acceleration z_rel = der(w_rel) [deg/s2]
Advanced			
Boolean	enforceStates	false	= true, if relative variables of spherical joint shall be used as states (StateSelect.always)
Boolean	useQuaternions	true	= true, if quaternions shall be used as states otherwise use 3 angles as states (provided enforceStates=true)
RotationSequence	sequence_angleSta	{1,2,3}	Sequence of rotations to rotate

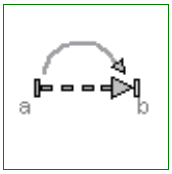
	tes		frame_a into frame_b around the 3 angles used as states

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Joints.FreeMotion



Free motion joint (6 degrees-of-freedom, 12 potential states)

Information

Joint which does not constrain the motion between frame_a and frame_b. Such a joint is only meaningful if the **relative** distance and orientation between frame_a and frame_b, and their derivatives, shall be used as **states**.

Note, that **bodies** such as Parts.Body, Parts.BodyShape, have potential states describing the distance and orientation, and their derivatives, between the **world frame** and a **body fixed frame**. Therefore, if these potential state variables are suited, a FreeMotion joint is not needed.

The states of the FreeMotion object are:

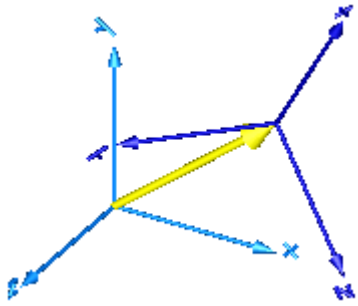
- The **relative position vector** r_rel_a from the origin of frame_a to the origin of frame_b, resolved in frame_a and the **relative velocity** v_rel_a of the origin of frame_b with respect to the origin of frame_a, resolved in frame_a (= der(r_rel_a)).
- If parameter **useQuaternions** in the "Advanced" menu is **true** (this is the default), then **4 quaternions** are states. Additionally, the coordinates of the relative angular velocity vector are 3 potential states.
  If **useQuaternions** in the "Advanced" menu is **false**, then **3 angles** and the derivatives of these angles are potential states. The orientation of frame_b is computed by rotating frame_a along the axes defined in parameter vector "sequence_angleStates" (default = {1,2,3}, i.e., the Cardan angle sequence) around the angles used as states. For example, the default is to rotate the x-axis of frame_a around angles[1], the new y-axis around angles[2] and the new z-axis around angles[3], arriving at frame_b.

The quaternions have the slight disadvantage that there is a non-linear constraint equation between the 4 quaternions. Therefore, at least one non-linear equation has to be solved during simulation. A tool might, however, analytically solve this simple constraint equation. Using the 3 angles as states has the disadvantage that there is a singular configuration in which a division by zero will occur. If it is possible to determine in advance for an application class that this singular configuration is outside of the operating region, the 3 angles might be used as states by setting **useQuaternions** = **false**.

In text books about 3-dimensional mechanics often 3 angles and the angular velocity are used as states. This is not the case here, since 3 angles and their derivatives are used as states (if useQuaternions = false). The reason is that for real-time simulation the discretization formula of the integrator might be "inlined" and solved together with the model equations. By appropriate symbolic transformation the performance is drastically increased if angles and their derivatives are used as states, instead of angles and the angular velocity.

If parameter **enforceStates** is set to **true** (= the default) in the "Advanced" menu, then FreeMotion variables are forced to be used as states according to the setting of parameters "useQuaternions" and "sequence_angleStates".

In the following figure the animation of a FreeMotion joint is shown. The light blue coordinate system is frame_a and the dark blue coordinate system is frame_b of the joint. (here: r_rel_a_start = {0.5, 0, 0.5}, angles_start = {45, 45, 45}^{o}).



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show arrow from frame_a to frame_b)
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
Position	r_rel_a_start[3]	{0,0,0}	Initial values of r_rel_a (vector from origin of frame_a to origin of frame_b resolved in frame_a) [m]
RotationSequence	sequence_start	{1,2,3}	Sequence of rotations to rotate frame_a into frame_b at initial time
Angle_deg	angles_start[3]	{0,0,0}	Initial values of angles to rotate frame_a around 'sequence_start' axes into frame_b [deg]
Velocity	v_rel_a_start[3]	{0,0,0}	Initial values of velocity v_rel_a = der(r_rel_a) [m/s]
AngularVelocity_degs	w_rel_a_start[3]	{0,0,0}	Initial values of angular velocity of frame_b with respect to frame_a resolved in frame_a [deg/s]
Acceleration	a_rel_a_start[3]	{0,0,0}	Initial values of acceleration a_rel_a = der(v_rel_a) [m/s2]
AngularAcceleration_degs2	z_rel_a_start[3]	{0,0,0}	Initial values of angular acceleration z_rel_a = der(w_rel_a) [deg/s2]
Animation			
if animation = true			
Length	arrowDiameter	world.defaultArrowDiameter	Diameter of arrow from frame_a to frame_b [m]
Color	arrowColor	Modelica.Mechanics.MultiBody...	Color of arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	true	= true, if relative variables

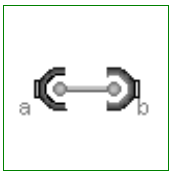
			between frame_a and frame_b shall be used as states
Boolean	useQuaternions	true	= true, if quaternions shall be used as states otherwise use 3 angles as states
RotationSequence	sequence_angleStates	{1,2,3}	Sequence of rotations to rotate frame_a into frame_b around the 3 angles used as states

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

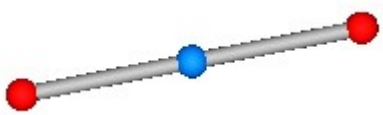
Modelica.Mechanics.MultiBody.Joints.SphericalSpherical



Spherical - spherical joint aggregation (1 constraint, no potential states) with an optional point mass in the middle

Information

Joint that has a spherical joint on each of its two ends. The rod connecting the two spherical joints is approximated by a point mass that is located in the middle of the rod. When the mass is set to zero (default), special code for a massless body is generated. In the following default animation figure, the two spherical joints are represented by two red spheres, the connecting rod by a grey cylinder and the point mass in the middle of the rod by a light blue sphere:



This joint introduces **one constraint** defining that the distance between the origin of frame_a and the origin of frame_b is constant (= rodLength). It is highly recommended to use this joint in loops whenever possible, because this enhances the efficiency considerably due to smaller systems of non-linear algebraic equations.

It is sometimes desirable to **compute** the **rodLength** of the connecting rod during initialization. For this, parameter **computeLength** has to be set to **true** and instead **one** other, easier to determine, position variable in the same loop needs to have a fixed attribute of **true**. For example, if a loop consists of one Revolute joint, one Prismatic joint and a SphericalSpherical joint, one may fix the start values of the revolute joint angle and of the relative distance of the prismatic joint in order to compute the rodLength of the rod.

It is not possible to connect other components, such as a body with mass properties or a special visual shape object to the rod connecting the two spherical joints. If this is needed, use instead joint Joints.**UniversalSpherical** that has this property.

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Boolean	showMass	true	= true, if mass shall be shown (provided animation = true and m

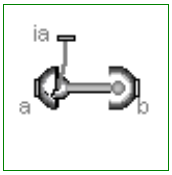
			> 0)
Boolean	computeRodLength	false	= true, if rodLength shall be computed during initialization (see info)
Length	rodLength	1	Distance between the origins of frame_a and frame_b (if computeRodLength=true, guess value) [m]
Mass	m	0	Mass of rod (= point mass located in middle of rod) [kg]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultJointLength	Diameter of spheres respresenting the spherical joints [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of spheres respresenting the spherical joints
Diameter	rodDiameter	sphereDiameter/Types.Default...	Diameter of rod connecting the two spherical joint [m]
Color	rodColor	Modelica.Mechanics.MultiBody...	Color of rod connecting the two spherical joints
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
if animation = true and showMass = true and m > 0			
Diameter	massDiameter	sphereDiameter	Diameter of sphere representing the mass point [m]
Color	massColor	Modelica.Mechanics.MultiBody...	Color of sphere representing the mass point
Advanced			
Boolean	kinematicConstraint	true	= false, if no constraint shall be defined, due to analytically solving a kinematic loop
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

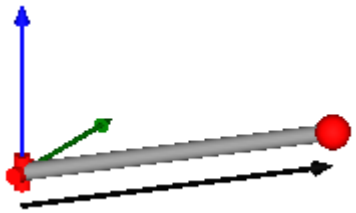
Modelica.Mechanics.MultiBody.Joints.UniversalSpherical



Universal - spherical joint aggregation (1 constraint, no potential states)

Information

This component consists of a **universal joint** at frame_a and a **spherical joint** at frame_b that are connected together with a **rigid rod**, see default aimation figure (the arrows are not part of the default animation):



This joint aggregation has no mass and no inertia and introduces the constraint that the distance between the origin of frame_a and the origin of frame_b is constant (= Frames.length(rRod_ia)). The universal joint is defined in the following way:

- The rotation **axis** of revolute joint **1** is along parameter vector n1_a which is fixed in frame_a.
- The rotation **axis** of revolute joint **2** is perpendicular to axis 1 and to the line connecting the universal and the spherical joint.

The definition of axis 2 of the universal joint is performed according to the most often occuring case. In a future release, axis 2 might be explicitly definable via a parameter. However, the treatment is much more complicated and the number of operations is considerably higher, if axis 2 is not orthogonal to axis 1 and to the connecting rod.

Note, there is a **singularity** when axis 1 and the connecting rod are parallel to other. Therefore, if possible n1_a should be selected in such a way that it is perpendicular to rRod_ia in the initial configuration (i.e., the distance to the singularity is as large as possible).

An additional **frame_ia** is present. It is **fixed** in the connecting **rod** at the origin of **frame_a**. The placement of frame_ia on the rod is implicitly defined by the universal joint (frame_a and frame_ia coincide when the angles of the two revolute joints of the universal joint are zero) and by parameter vector **rRod_ia**, the position vector from the origin of frame_a to the origin of frame_b, resolved in frame_**ia**.

The easiest way to define the parameters of this joint is by moving the MultiBody system in a **reference configuration** where **all frames** of all components are **parallel** to other (alternatively, at least frame_a and frame_ia of the UniversalSpherical joint should be parallel to other when defining an instance of this component). Since frame_a and frame_ia are parallel to other, vector **rRod_ia** from frame_a to frame_b resolved in frame_**ia** can be resolved in frame_**a** (or the **world frame**, if all frames are parallel to other).

This joint aggregation can be used in cases where in reality a rod with spherical joints at end are present. Such a system has an additional degree of freedom to rotate the rod along its axis. In practice this rotation is usually of no interested and is mathematically removed by replacing one of the spherical joints by a universal joint. Still, in most cases the Joints.SphericalSpherical joint aggregation can be used instead of the UniversalSpherical joint since the rod is animated and its mass properties are approximated by a point mass in the middle of the rod. The SphericalSpherical joint has the advantage that it does not have a singular configuration.

In the public interface of the UniversalSpherical joint, the following (final) **parameters** are provided:

```
  parameter Real rodLength(unit="m")  "Length of rod";
  parameter Real eRod_ia[3] "Unit vector along rod, resolved in frame_ia";
  parameter Real e2_ia  [3] "Unit vector along axis 2, resolved in frame_ia";
```

This allows a more convenient definition of data which is related to the rod. For example, if a box shall be connected at frame_ia directing from the origin of frame_a to the middle of the rod, this might be defined as:

```
    Modelica.Mechanics.MultiBody.Joints.UniversalSpherical jointUS(rRod_ia={1.2,
1, 0.2});
    Modelica.Mechanics.MultiBody.Visualizers.FixedShape    shape(shapeType
= "box",
                                                lengthDirection = jointUS.eRod_ia,
                                                widthDirection  = jointUS.e2_ia,
                                                length          =
jointUS.rodLength/2,
                                                width           =
```

```
jointUS.rodLength/10);
  equation
    connect(jointUS.frame_ia, shape.frame_a);
```

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Boolean	showUniversalAxes	true	= true, if universal joint shall be visualized with two cylinders, otherwise with a sphere (provided animation=true)
Boolean	computeRodLength	false	= true, if distance between frame_a and frame_b shall be computed during initialization (see info)
Axis	n1_a	{0,0,1}	Axis 1 of universal joint resolved in frame_a (axis 2 is orthogonal to axis 1 and to rod)
Position	rRod_ia[3]	{1,0,0}	Vector from origin of frame_a to origin of frame_b, resolved in frame_ia (if computeRodLength=true, rRod_ia is only an axis vector along the connecting rod) [m]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultJointLength	Diameter of spheres representing the universal and the spherical joint [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of spheres representing the universal and the spherical joint
ShapeType	rodShapeType	"cylinder"	Shape type of rod connecting the universal and the spherical joint
Distance	rodWidth	sphereDiameter/Types.Default...	Width of rod shape in direction of axis 2 of universal joint. [m]
Distance	rodHeight	rodWidth	Height of rod shape in direction that is orthogonal to rod and to axis 2 [m]
ShapeExtra	rodExtra	0.0	Additional parameter depending on rodShapeType
Color	rodColor	Modelica.Mechanics.MultiBody...	Color of rod shape connecting the universal and the spherical joints
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
if animation = true and showUniversalAxes			
Distance	cylinderLength	world.defaultJointLength	Length of cylinders representing the two universal joint axes [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinders representing the two universal

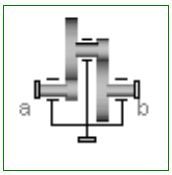
			joint axes [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody. ..	Color of cylinders representing the two universal joint axes
Advanced			
Boolean	kinematicConstraint	true	= false, if no constraint shall be defined, due to analytically solving a kinematic loop
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_a	frame_ia	Coordinate system at the origin of frame_a, fixed at the rod connecting the universal with the spherical joint

---

Modelica.Mechanics.MultiBody.Joints.GearConstraint



Ideal 3-dim. gearbox (arbitrary shaft directions)

Information

This ideal massless joint provides a gear constraint between frames `frame_a` and `frame_b`. The axes of rotation of `frame_a` and `frame_b` may be arbitrary.

Reference

Schweiger, Christian ; Otter, Martin: Modelling 3D Mechanical Effects of 1-dim. Powertrains. In: *Proceedings of the 3rd International Modelica Conference*. Linköping : The Modelica Association and Linköping University, November 3-4, 2003, pp. 149-158

Parameters

Type	Name	Default	Description
Real	ratio	2	Gear speed ratio
Axis	n_a	{1,0,0}	Axis of rotation of shaft a (same coordinates in frame_a, frame_b, bearing)
Axis	n_b	{1,0,0}	Axis of rotation of shaft b (same coordinates in frame_a, frame_b, bearing)
Position	r_a[3]	{0,0,0}	Vector from frame bearing to frame_a resolved in bearing [m]
Position	r_b[3]	{0,0,0}	Vector from frame bearing to frame_b resolved in bearing [m]

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_a	bearing	Coordinate system fixed in the bearing

Modelica.Mechanics.MultiBody.Joints.Assemblies

Joint aggregations for analytic loop handling

Information

The joints in this package are mainly designed to be used in **kinematic loop** structures. Every component consists of **3 elementary joints**. These joints are combined in such a way that the kinematics of the 3 joints between frame_a and frame_b are computed from the movement of frame_a and frame_b, i.e., there are **no constraints** between frame_a and frame_b. This requires to solve a **non-linear system of equations** which is performed **analytically** (i.e., when a mathematical solution exists, it is computed efficiently and reliably). A detailed description how to use these joints is provided in MultiBody.UsersGuide.Tutorial.LoopStructures.AnalyticLoopHandling.

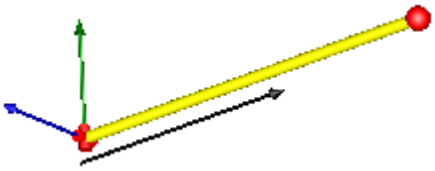
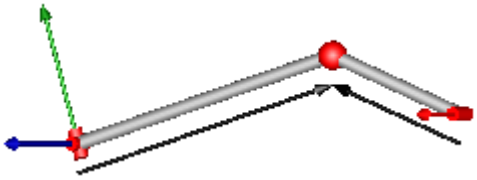
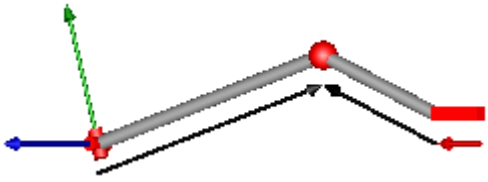
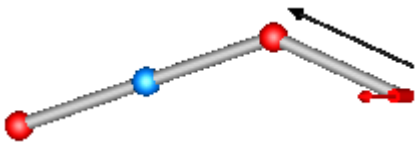
The assembly joints in this package are named **JointXYZ** where **XYZ** are the first letters of the elementary joints used in the component, in particular:

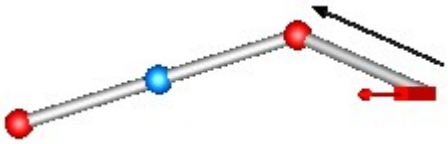
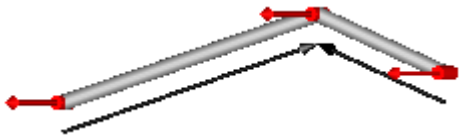
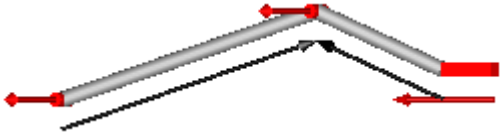
P	Prismatic joint
R	Revolute joint
S	Spherical joint
U	Universal joint

For example, JointUSR is an assembly joint consisting of a universal, a spherical and a revolute joint.

This package contains the following models:

Content

Model	*Description*
JointUPS	Universal - prismatic - spherical joint aggregation 
JointUSR	Universal - spherical - revolute joint aggregation 
JointUSP	Universal - spherical - prismatic joint aggregation 
JointSSR	Spherical - spherical - revolute joint aggregation with an optional mass point at the rod connecting the two spherical joints 

JointSSP	Spherical - spherical - prismatic joint aggregation with an optional mass point at the rod connecting the two spherical joints 
JointRRR	Revolute - revolute - revolute joint aggregation for planar loops 
JointRRP	Revolute - revolute - prismatic joint aggregation for planar loops 

Note, no component of this package has potential states, since the components are designed in such a way that the generalized coordinates of the used elementary joints are computed from the frame_a and frame_b coordinates. Still, it is possible to use the components in a tree structure. In this case states are selected from bodies that are connected to the frame_a or frame_b side of the component. In most cases this gives a less efficient solution, as if elementary joints of package Modelica.Mechanics.MultiBody.Joints would be used directly.

The analytic handling of kinematic loops by using joint aggregations with 6 degrees of freedom as provided in this package, is a **new** methodology. It is based on a more general method for solving non-linear equations of kinematic loops developed by Woernle and Hiller. An automatic application of this more general method is difficult, and a manual application is only suited for specialists in this field. The method introduced here is a compromize: It can be quite easily applied by an end user, but for a smaller class of kinematic loops. The method of the "characteristic pair of joints" from Woernle and Hiller is described in:

Woernle C.:
Ein systematisches Verfahren zur Aufstellung der geometrischen Schliessbedingungen in kinematischen Schleifen mit Anwendung bei der Rückwärtstransformation für Industrieroboter.
Fortschritt-Berichte VDI, Reihe 18, Nr. 59, Duesseldorf: VDI-Verlag 1988, ISBN 3-18-145918-6.

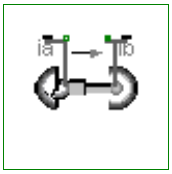
Hiller M., and Woernle C.:**A Systematic Approach for Solving the Inverse Kinematic Problem of Robot Manipulators**.
Proceedings 7th World Congress Th. Mach. Mech., Sevilla 1987.

Package Content

Name	Description
JointUPS	Universal - prismatic - spherical joint aggregation (no constraints, no potential states)
JointUSR	Universal - spherical - revolute joint aggregation (no constraints, no potential states)
JointUSP	Universal - spherical - prismatic joint aggregation (no constraints, no potential states)
JointSSR	Spherical - spherical - revolute joint aggregation with mass (no constraints, no potential states)
JointSSP	Spherical - spherical - prismatic joint aggregation with mass (no constraints, no potential states)
JointRRR	Planar revolute - revolute - revolute joint aggregation (no constraints, no potential states)

JointRRP	Planar revolute - revolute - prismatic joint aggregation (no constraints, no potential states)

Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUPS



Universal - prismatic - spherical joint aggregation (no constraints, no potential states)

Information

This component consists of a **universal** joint at frame_a, a **spherical** joint at frame_b and a **prismatic** joint along the line connecting the origin of frame_a and the origin of frame_b, see the default animation in the following figure (the axes vectors are not part of the default animation):

This joint aggregation has no mass and no inertia and introduces neither constraints nor potential state variables. It is especially useful to build up more complicated force elements where the mass and/or inertia of the force element shall be taken into account.

The universal joint is defined in the following way:

- The rotation **axis** of revolute joint **1** is along parameter vector n1_a which is fixed in frame_a.
- The rotation **axis** of revolute joint **2** is perpendicular to axis 1 and to the line connecting the universal and the spherical joint.

The definition of axis 2 of the universal joint is performed according to the most often occuring case. In a future release, axis 2 might be explicitly definable via a parameter. However, the treatment is much more complicated and the number of operations is considerably higher, if axis 2 is not orthogonal to axis 1 and to the connecting rod.

Note, there is a **singularity** when axis 1 and the connecting line are parallel to each other. Therefore, if possible n1_a should be selected in such a way that it is perpendicular to nAxis_ia in the initial configuration (i.e., the distance to the singularity is as large as possible).

An additional **frame_ia** is present. It is **fixed** on the line connecting the universal and the spherical joint at the origin of **frame_a**. The placement of frame_ia on this line is implicitly defined by the universal joint (frame_a and frame_ia coincide when the angles of the two revolute joints of the universal joint are zero) and by parameter vector **nAxis_ia**, an axis vector directed along the line from the origin of frame_a to the spherical joint, resolved in frame_**ia**.

An additional **frame_ib** is present. It is **fixed** in the line connecting the prismatic and the spherical joint at the origin of **frame_b**. It is always parallel to **frame_ia**.

Note, this joint aggregation can be used in cases where in reality a rod with spherical joints at each end are present. Such a system has an additional degree of freedom to rotate the rod along its axis. In practice this rotation is usually of no interested and is mathematically removed by replacing one of the spherical joints by a universal joint.

The easiest way to define the parameters of this joint is by moving the MultiBody system in a **reference configuration** where **all frames** of all components are **parallel** to each other (alternatively, at least frame_a, frame_ia and frame_ib of the JointUSP joint should be parallel to each other when defining an instance of this component).

Parameters

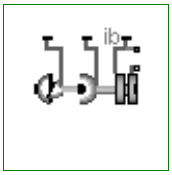
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Boolean	showUniversalAxes	true	= true, if universal joint shall be visualized with two cylinders, otherwise with a sphere (provided animation=true)
Axis	n1_a	{0,0,1}	Axis 1 of universal joint resolved in frame_a (axis 2 is orthogonal to axis 1 and to line from universal to spherical joint)
Position	nAxis_ia[3]	{1,0,0}	Axis vector along line from origin of frame_a to origin of frame_b, resolved in frame_ia [m]
Position	s_offset	0	Relative distance offset (distance between frame_a and frame_b = s(t) + s_offset) [m]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultJointLength	Diameter of spheres representing the spherical joints [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of spheres representing the spherical joints
Diameter	axisDiameter	sphereDiameter/Types.Default...	Diameter of cylinder on the connecting line from frame_a to frame_b [m]
Color	axisColor	Modelica.Mechanics.MultiBody...	Color of cylinder on the connecting line from frame_a to frame_b
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
if animation = true and showUniversalAxes			
Distance	cylinderLength	world.defaultJointLength	Length of cylinders representing the two universal joint axes [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinders representing the two universal joint axes [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinders representing the two universal joint axes
Advanced			
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_a	frame_ia	Coordinate system at origin of frame_a fixed at prismatic joint

Frame_b	frame_ib	Coordinate system at origin of frame_b fixed at prismatic joint
Flange_a	axis	1-dim. translational flange that drives the prismatic joint
Flange_b	bearing	1-dim. translational flange of the drive bearing of the prismatic joint

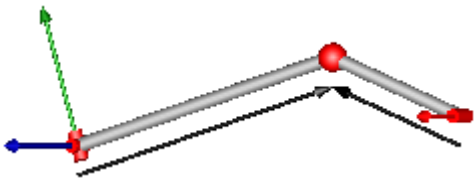
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUSR



Universal - spherical - revolute joint aggregation (no constraints, no potential states)

Information

This component consists of a **universal** joint at frame_a, a **revolute** joint at frame_b and a **spherical** joint which is connected via **rod1** to the universal and via **rod2** to the revolute joint, see the default animation in the following figure (the axes vectors are not part of the default animation):



This joint aggregation has no mass and no inertia and introduces neither constraints nor potential state variables. It should be used in kinematic loops whenever possible since the non-linear system of equations introduced by this joint aggregation is solved **analytically** (i.e., a solution is always computed, if a unique solution exists).

The universal joint is defined in the following way:

- The rotation **axis** of revolute joint **1** is along parameter vector n1_a which is fixed in frame_a.
- The rotation **axis** of revolute joint **2** is perpendicular to axis 1 and to the line connecting the universal and the spherical joint (= rod 1).

The definition of axis 2 of the universal joint is performed according to the most often occuring case. In a future release, axis 2 might be explicitly definable via a parameter. However, the treatment is much more complicated and the number of operations is considerably higher, if axis 2 is not orthogonal to axis 1 and to the connecting rod.

Note, there is a **singularity** when axis 1 and the connecting rod are parallel to each other. Therefore, if possible n1_a should be selected in such a way that it is perpendicular to rRod1_ia in the initial configuration (i.e., the distance to the singularity is as large as possible).

The rest of this joint aggregation is defined by the following parameters:

- The position of the spherical joint with respect to the universal joint is defined by vector **rRod1_ia**. This vector is directed from frame_a to the spherical joint and is resolved in frame_ia (it is most simple to select frame_ia such that it is parallel to frame_a in the reference or initial configuration).
- The position of the spherical joint with respect to the revolute joint is defined by vector **rRod2_ib**. This vector is directed from the inner frame of the revolute joint (frame_ib or revolute.frame_a) to the spherical joint and is resolved in frame_ib (note, that frame_ib and frame_b are parallel to each other).
- The axis of rotation of the revolute joint is defined by axis vector **n_b**. It is fixed and resolved in frame_b.
- When specifying this joint aggregation with the definitions above, **two** different **configurations** are possible. Via parameter **phi_guess** a guess value for revolute.phi(t0) at the initial time t0 is given. The configuration is selected that is closest to phi_guess (|revolute.phi - phi_guess|is minimal).

An additional **frame_ia** is present. It is **fixed** in the rod connecting the universal and the spherical joint at the origin of **frame_a**. The placement of frame_ia on the rod is implicitly defined by the universal joint (frame_a and frame_ia coincide when the angles of the two revolute joints of the universal joint are zero) and by parameter vector **rRod1_ia**, the position vector from the origin of frame_a to the spherical joint, resolved in

frame_**ia**.

An additional **frame_ib** is present. It is **fixed** in the rod connecting the revolute and the spherical joint at the side of the revolute joint that is connected to this rod (= rod2.frame_a = revolute.frame_a).

An additional **frame_im** is present. It is **fixed** in the rod connecting the revolute and the spherical joint at the side of the spherical joint that is connected to this rod (= rod2.frame_b). It is always parallel to **frame_ib**.

The easiest way to define the parameters of this joint is by moving the MultiBody system in a **reference configuration** where **all frames** of all components are **parallel** to each other (alternatively, at least frame_a and frame_ia of the JointUSR joint should be parallel to each other when defining an instance of this component).

In the public interface of the JointUSR joint, the following (final) **parameters** are provided:

```
  parameter Real rod1Length(unit="m")  "Length of rod 1";
  parameter Real eRod1_ia[3] "Unit vector along rod 1, resolved in frame_ia";
  parameter Real e2_ia  [3]  "Unit vector along axis 2, resolved in frame_ia";
```

This allows a more convenient definition of data which is related to rod 1. For example, if a box shall be connected at frame_ia directing from the origin of frame_a to the middle of rod 1, this might be defined as:

```
    Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUSP
jointUSR(rRod1_ia={1.2, 1, 0.2});
    Modelica.Mechanics.MultiBody.Visualizers.FixedShape     shape(shapeType
= "box",
                                                lengthDirection =
jointUSR.eRod1_ia,
                                                widthDirection  = jointUSR.e2_ia,
                                                length          =
jointUSR.rod1Length/2,
                                                width           =
jointUSR.rod1Length/10);
  equation
    connect(jointUSP.frame_ia, shape.frame_a);
```

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Boolean	showUniversalAxes	true	= true, if universal joint shall be visualized with two cylinders, otherwise with a sphere (provided animation=true)
Axis	n1_a	{0,0,1}	Axis 1 of universal joint fixed and resolved in frame_a (axis 2 is orthogonal to axis 1 and to rod 1)
Axis	n_b	{0,0,1}	Axis of revolute joint fixed and resolved in frame_b
Position	rRod1_ia[3]	{1,0,0}	Vector from origin of frame_a to spherical joint, resolved in frame_ia [m]
Position	rRod2_ib[3]	{-1,0,0}	Vector from origin of frame_ib to spherical joint, resolved in frame_ib [m]

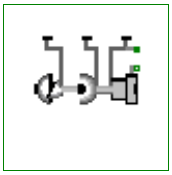
Angle_deg	phi_offset	0	Relative angle offset of revolute joint (angle = phi(t) + from_deg(phi_offset)) [deg]
Angle_deg	phi_guess	0	Select the configuration such that at initial time |phi(t0) - from_deg(phi_guess)|is minimal [deg]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultJointLength	Diameter of the spheres representing the universal and the spherical joint [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of the spheres representing the universal and the spherical joint
Diameter	rod1Diameter	sphereDiameter/Types.Default...	Diameter of rod 1 connecting the universal and the spherical joint [m]
Color	rod1Color	Modelica.Mechanics.MultiBody...	Color of rod 1 connecting the universal and the spherical joint
Diameter	rod2Diameter	rod1Diameter	Diameter of rod 2 connecting the revolute and the spherical joint [m]
Color	rod2Color	rod1Color	Color of rod 2 connecting the revolute and the spherical joint
Diameter	revoluteDiameter	world.defaultJointWidth	Diameter of cylinder representing the revolute joint [m]
Distance	revoluteLength	world.defaultJointLength	Length of cylinder representing the revolute joint [m]
Color	revoluteColor	Modelica.Mechanics.MultiBody...	Color of cylinder representing the revolute joint
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
if animation = true and showUniversalAxes			
Distance	cylinderLength	world.defaultJointLength	Length of cylinders representing the two universal joint axes [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinders representing the two universal joint axes [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinders representing the two universal joint axes
Advanced			
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_a	frame_ia	Coordinate system at origin of frame_a fixed at connecting rod of universal and

		spherical joint
Frame_b	frame_ib	Coordinate system at origin of frame_b fixed at connecting rod of spherical and revolute joint
Frame_b	frame_im	Coordinate system at origin of spherical joint fixed at connecting rod of spherical and revolute joint
Flange_a	axis	1-dim. rotational flange that drives the revolute joint
Flange_b	bearing	1-dim. rotational flange of the drive bearing of the revolute joint

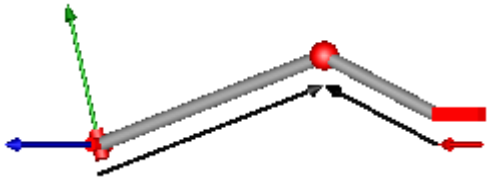
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUSP



Universal - spherical - prismatic joint aggregation (no constraints, no potential states)

Information

This component consists of a **universal** joint at frame_a, a **prismatic** joint at frame_b and a **spherical** joint which is connected via **rod1** to the universal and via **rod2** to the prismatic joint, see the default animation in the following figure (the axes vectors are not part of the default animation):



This joint aggregation has no mass and no inertia and introduces neither constraints nor potential state variables. It should be used in kinematic loops whenever possible since the non-linear system of equations introduced by this joint aggregation is solved **analytically** (i.e., a solution is always computed, if a unique solution exists).

The universal joint is defined in the following way:

- The rotation **axis** of revolute joint **1** is along parameter vector n1_a which is fixed in frame_a.
- The rotation **axis** of revolute joint **2** is perpendicular to axis 1 and to the line connecting the universal and the spherical joint (= rod 1).

The definition of axis 2 of the universal joint is performed according to the most often occuring case. In a future release, axis 2 might be explicitly definable via a parameter. However, the treatment is much more complicated and the number of operations is considerably higher, if axis 2 is not orthogonal to axis 1 and to the connecting rod.

Note, there is a **singularity** when axis 1 and the connecting rod are parallel to each other. Therefore, if possible n1_a should be selected in such a way that it is perpendicular to rRod1_ia in the initial configuration (i.e., the distance to the singularity is as large as possible).

The rest of this joint aggregation is defined by the following parameters:

- The position of the spherical joint with respect to the universal joint is defined by vector **rRod1_ia**. This vector is directed from frame_a to the spherical joint and is resolved in frame_ia (it is most simple to select frame_ia such that it is parallel to frame_a in the reference or initial configuration).
- The position of the spherical joint with respect to the prismatic joint is defined by vector **rRod2_ib**. This vector is directed from the inner frame of the prismatic joint (frame_ib or prismatic.frame_a) to the spherical joint and is resolved in frame_ib (note, that frame_ib and frame_b are parallel to each other).
- The axis of translation of the prismatic joint is defined by axis vector **n_b**. It is fixed and resolved in frame_b.
- The two frames of the prismatic joint, i.e., frame_b and frame_ib, are parallel to each other. The distance between the origins of these two frames along axis n_b is equal to "prismatic.s(t) + s_offset", where "prismatic.s(t)" is a time varying variable and "s_offset" is a fixed, constant offset

parameter.
- When specifying this joint aggregation with the definitions above, **two** different **configurations** are possible. Via parameter **s_guess** a guess value for prismatic.s(t0) at the initial time t0 is given. The configuration is selected that is closest to s_guess (|prismatic.s - s_guess|is minimal).

An additional **frame_ia** is present. It is **fixed** in the rod connecting the universal and the spherical joint at the origin of **frame_a**. The placement of frame_ia on the rod is implicitly defined by the universal joint (frame_a and frame_ia coincide when the angles of the two revolute joints of the universal joint are zero) and by parameter vector **rRod1_ia**, the position vector from the origin of frame_a to the spherical joint, resolved in frame_**ia**.

An additional **frame_ib** is present. It is **fixed** in the rod connecting the prismatic and the spherical joint at the side of the prismatic joint that is connected to this rod (= rod2.frame_a = prismatic.frame_a). It is always parallel to **frame_b**.

An additional **frame_im** is present. It is **fixed** in the rod connecting the prismatic and the spherical joint at the side of the spherical joint that is connected to this rod (= rod2.frame_b). It is always parallel to **frame_b**.

The easiest way to define the parameters of this joint is by moving the MultiBody system in a **reference configuration** where **all frames** of all components are **parallel** to each other (alternatively, at least frame_a and frame_ia of the JointUSP joint should be parallel to each other when defining an instance of this component).

In the public interface of the JointUSP joint, the following (final) **parameters** are provided:

```
  parameter Real rod1Length(unit="m")  "Length of rod 1";
  parameter Real eRod1_ia[3] "Unit vector along rod 1, resolved in frame_ia";
  parameter Real e2_ia  [3]  "Unit vector along axis 2, resolved in frame_ia";
```

This allows a more convenient definition of data which is related to rod 1. For example, if a box shall be connected at frame_ia directing from the origin of frame_a to the middle of rod 1, this might be defined as:

```
    Modelica.Mechanics.MultiBody.Joints.Assemblies.JointUSP
jointUSP(rRod1_ia={1.2, 1, 0.2});
    Modelica.Mechanics.MultiBody.Visualizers.FixedShape     shape(shapeType
= "box",
                                                  lengthDirection =
jointUSP.eRod1_ia,
                                                  widthDirection  = jointUSP.e2_ia,
                                                  length          =
jointUSP.rod1Length/2,
                                                  width           =
jointUSP.rod1Length/10);
  equation
    connect(jointUSP.frame_ia, shape.frame_a);
```

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Boolean	showUniversalAxes	true	= true, if universal joint shall be visualized with two cylinders, otherwise with a sphere (provided animation=true)
Axis	n1_a	{0,0,1}	Axis 1 of universal joint fixed and resolved in frame_a (axis 2 is

			orthogonal to axis 1 and to rod 1)
Axis	n_b	{-1,0,0}	Axis of prismatic joint fixed and resolved in frame_b
Position	rRod1_ia[3]	{1,0,0}	Vector from origin of frame_a to spherical joint, resolved in frame_ia [m]
Position	rRod2_ib[3]	{-1,0,0}	Vector from origin of frame_ib to spherical joint, resolved in frame_ib (frame_ib is parallel to frame_b) [m]
Position	s_offset	0	Relative distance offset of prismatic joint (distance between the prismatic joint frames = s(t) + s_offset) [m]
Position	s_guess	0	Select the configuration such that at initial time |s(t0)-s_guess|is minimal [m]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultJointLength	Diameter of the spheres representing the universal and the spherical joint [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of the spheres representing the universal and the spherical joint
Diameter	rod1Diameter	sphereDiameter/Types.Default...	Diameter of rod 1 connecting the universal and the spherical joint [m]
Color	rod1Color	Modelica.Mechanics.MultiBody...	Color of rod 1 connecting the universal and the spherical joint
Diameter	rod2Diameter	rod1Diameter	Diameter of rod 2 connecting the prismatic and the spherical joint [m]
Color	rod2Color	rod1Color	Color of rod 2 connecting the prismatic and the spherical joint
Axis	boxWidthDirection	{0,1,0}	Vector in width direction of prismatic joint, resolved in frame_b
Distance	boxWidth	world.defaultJointWidth	Width of prismatic joint box [m]
Distance	boxHeight	boxWidth	Height of prismatic joint box [m]
Color	boxColor	sphereColor	Color of prismatic joint box
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
if animation = true and showUniversalAxes			
Distance	cylinderLength	world.defaultJointLength	Length of cylinders representing the two universal joint axes [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinders representing the two universal joint axes [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinders representing the two universal joint axes
Advanced			

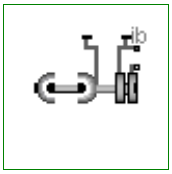
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_a	frame_ia	Coordinate system at origin of frame_a fixed at connecting rod of universal and spherical joint
Frame_b	frame_ib	Coordinate system at origin of frame_b fixed at connecting rod of spherical and prismatic joint
Frame_b	frame_im	Coordinate system at origin of spherical joint fixed at connecting rod of spherical and prismatic joint
Flange_a	axis	1-dim. translational flange that drives the prismatic joint
Flange_b	bearing	1-dim. translational flange of the drive bearing of the prismatic joint

---

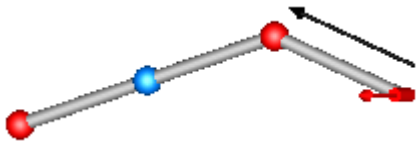
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointSSR



Spherical - spherical - revolute joint aggregation with mass (no constraints, no potential states)

Information

This component consists of a **spherical** joint 1 at frame_a, a **revolute** joint at frame_b and a **spherical** joint 2 which is connected via rod 1 to the spherical joint 1 and via rod 2 to the revolute joint, see the default animation in the following figure (the axes vectors are not part of the default animation):



Besides an optional point mass in the middle of rod 1, this joint aggregation has no mass and no inertia, and introduces neither constraints nor potential state variables. It should be used in kinematic loops whenever possible since the non-linear system of equations introduced by this joint aggregation is solved **analytically** (i.e., a solution is always computed, if a unique solution exists).

An additional **frame_ib** is present. It is **fixed** in rod 2 connecting the revolute and the spherical joint at the side of the revolute joint that is connected to this rod (= rod2.frame_a = revolute.frame_a).

An additional **frame_im** is present. It is **fixed** in rod 2 connecting the revolute and the spherical joint at the side of spherical joint 2 that is connected to this rod (= rod2.frame_b). It is always parallel to **frame_ib**.

The easiest way to define the parameters of this joint is by moving the MultiBody system in a **reference configuration** where **all frames** of all components are **parallel** to each other (alternatively, at least frame_b and frame_ib of the JointSSR joint should be parallel to each other when defining an instance of this component).

Parameters

Type	Name	Default	Description

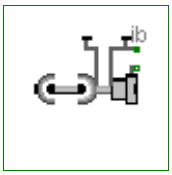
Boolean	animation	true	= true, if animation shall be enabled
Boolean	showMass	true	= true, if point mass on rod 1 shall be shown (provided animation = true and rod1Mass > 0)
Length	rod1Length	1	Distance between the origins of the two spherical joints [m]
Mass	rod1Mass	0	Mass of rod 1 (= point mass located in middle of rod connecting the two spherical joints) [kg]
Axis	n_b	{0,0,1}	Axis of revolute joint fixed and resolved in frame_b
Position	rRod2_ib[3]	{1,0,0}	Vector from origin of frame_ib to spherical joint in the middle, resolved in frame_ib [m]
Angle_deg	phi_offset	0	Relative angle offset of revolute joint (angle = phi(t) + from_deg(phi_offset)) [deg]
Angle_deg	phi_guess	0	Select the configuration such that at initial time |phi(t0) - from_deg(phi_guess)|is minimal [deg]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultJointLength	Diameter of the spheres representing the two spherical joints [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of the spheres representing the two spherical joints
Diameter	rod1Diameter	sphereDiameter/Types.Default...	Diameter of rod 1 connecting the two spherical joints [m]
Color	rod1Color	Modelica.Mechanics.MultiBody...	Color of rod 1 connecting the two spherical joint
Diameter	rod2Diameter	rod1Diameter	Diameter of rod 2 connecting the revolute joint and spherical joint 2 [m]
Color	rod2Color	rod1Color	Color of rod 2 connecting the revolute joint and spherical joint 2
Diameter	revoluteDiameter	world.defaultJointWidth	Diameter of cylinder representing the revolute joint [m]
Distance	revoluteLength	world.defaultJointLength	Length of cylinder representing the revolute joint [m]
Color	revoluteColor	Modelica.Mechanics.MultiBody...	Color of cylinder representing the revolute joint
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_ib	Coordinate system at origin of frame_b fixed at connecting rod of spherical and revolute joint
Frame_b	frame_im	Coordinate system at origin of spherical joint in the middle fixed at connecting rod of spherical and revolute joint
Flange_a	axis	1-dim. rotational flange that drives the revolute joint
Flange_b	bearing	1-dim. rotational flange of the drive bearing of the revolute joint

---

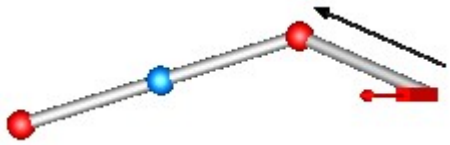
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointSSP



Spherical - spherical - prismatic joint aggregation with mass (no constraints, no potential states)

Information

This component consists of a **spherical** joint 1 at frame_a, a **prismatic** joint at frame_b and a **spherical** joint 2 which is connected via rod 1 to the spherical joint 1 and via rod 2 to the prismatic joint, see the default animation in the following figure (the axes vectors are not part of the default animation):



Besides an optional point mass in the middle of rod 1, this joint aggregation has no mass and no inertia, and introduces neither constraints nor potential state variables. It should be used in kinematic loops whenever possible since the non-linear system of equations introduced by this joint aggregation is solved **analytically** (i.e., a solution is always computed, if a unique solution exists).

An additional **frame_ib** is present. It is **fixed** in rod 2 connecting the prismatic and the spherical joint at the side of the prismatic joint that is connected to this rod (= rod2.frame_a = prismatic.frame_a).

An additional **frame_im** is present. It is **fixed** in rod 2 connecting the prismatic and the spherical joint at the side of spherical joint 2 that is connected to this rod (= rod2.frame_b). It is always parallel to **frame_ib**.

The easiest way to define the parameters of this joint is by moving the MultiBody system in a **reference configuration** where **all frames** of all components are **parallel** to each other (alternatively, at least frame_b and frame_ib of the JointSSP joint should be parallel to each other when defining an instance of this component).

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Boolean	showMass	true	= true, if point mass on rod 1 shall be shown (provided animation =

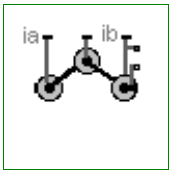
			true and rod1Mass > 0)
Length	rod1Length	1	Distance between the origins of the two spherical joints [m]
Mass	rod1Mass	0	Mass of rod 1 (= point mass located in middle of rod connecting the two spherical joints) [kg]
Axis	n_b	{0,0,1}	Axis of prismatic joint fixed and resolved in frame_b
Position	rRod2_ib[3]	{1,0,0}	Vector from origin of frame_ib to spherical joint in the middle, resolved in frame_ib [m]
Position	s_offset	0	Relative distance offset of prismatic joint (distance between frame_b and frame_ib = s(t) + s_offset) [m]
Position	s_guess	0	Select the configuration such that at initial time |s(t0)-s_guess|is minimal [m]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultJointLength	Diameter of the spheres representing the two spherical joints [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of the spheres representing the two spherical joints
Diameter	rod1Diameter	sphereDiameter/Types.Default...	Diameter of rod 1 connecting the two spherical joints [m]
Color	rod1Color	Modelica.Mechanics.MultiBody...	Color of rod 1 connecting the two spherical joint
Diameter	rod2Diameter	rod1Diameter	Diameter of rod 2 connecting the revolute joint and spherical joint 2 [m]
Color	rod2Color	rod1Color	Color of rod 2 connecting the revolute joint and spherical joint 2
Axis	boxWidthDirection	{0,1,0}	Vector in width direction of prismatic joint box, resolved in frame_b
Distance	boxWidth	world.defaultJointWidth	Width of prismatic joint box [m]
Distance	boxHeight	boxWidth	Height of prismatic joint box [m]
Color	boxColor	Modelica.Mechanics.MultiBody...	Color of prismatic joint box
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description

Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_ib	Coordinate system at origin of frame_b fixed at connecting rod of spherical and prismatic joint
Frame_b	frame_im	Coordinate system at origin of spherical joint in the middle fixed at connecting rod of spherical and prismatic joint
Flange_a	axis	1-dim. translational flange that drives the prismatic joint
Flange_b	bearing	1-dim. translational flange of the drive bearing of the prismatic joint

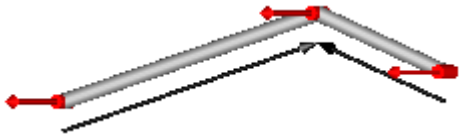
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointRRR



Planar revolute - revolute - revolute joint aggregation (no constraints, no potential states)

Information

This component consists of **3 revolute** joints with parallel axes of rotation that are connected together by two rods, see the default animation in the following figure (the axes vectors are not part of the default animation):



This joint aggregation introduces neither constraints nor state variables and should therefore be used in kinematic loops whenever possible to avoid non-linear systems of equations. It is only meaningful to use this component in **planar loops**. Basically, the position and orientation of the 3 revolute joints as well as of frame_ia, frame_ib, and frame_im are calculated by solving analytically a non-linear equation, given the position and orientation at frame_a and at frame_b.

Connector **frame_a** is the "left" side of the first revolute joint whereas **frame_ia** is the "right side of this revolute joint, fixed in rod 1. Connector **frame_b** is the "right" side of the third revolute joint whereas **frame_ib** is the "left" side of this revolute joint, fixed in rod 2. Finally, connector **frame_im** is the connector at the "right" side of the revolute joint in the middle, fixed in rod 2.

The easiest way to define the parameters of this joint is by moving the MultiBody system in a **reference configuration** where **all frames** of all components are **parallel** to each other (alternatively, at least frame_a, frame_ia, frame_im, frame_ib, frame_b of the JointRRR joint should be parallel to each other when defining an instance of this component).

Basically, the JointRRR model consists internally of a universal - spherical - revolute joint aggregation (= JointUSR). In a planar loop this will behave as if 3 revolute joints with parallel axes are connected by rigid rods.

Parameters

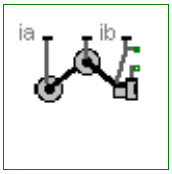
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Axis	n_a	{0,0,1}	Axes of revolute joints resolved in frame_a (all axes are parallel to

			each other)
Position	rRod1_ia[3]	{1,0,0}	Vector from origin of frame_a to revolute joint in the middle, resolved in frame_ia [m]
Position	rRod2_ib[3]	{-1,0,0}	Vector from origin of frame_ib to revolute joint in the middle, resolved in frame_ib [m]
Angle_deg	phi_offset	0	Relative angle offset of revolute joint at frame_b (angle = phi(t) + from_deg(phi_offset)) [deg]
Angle_deg	phi_guess	0	Select the configuration such that at initial time |phi(t0) - from_deg(phi_guess)|is minimal [deg]
Animation			
if animation = true			
Distance	cylinderLength	world.defaultJointLength	Length of cylinders representing the revolute joints [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinders representing the revolute joints [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinders representing the revolute joints
Diameter	rodDiameter	1.1*cylinderDiameter	Diameter of the two rods connecting the revolute joints [m]
Color	rodColor	Modelica.Mechanics.MultiBody...	Color of the two rods connecting the revolute joint
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_a	frame_ia	Coordinate system at origin of frame_a fixed at connecting rod of left and middle revolute joint
Frame_b	frame_ib	Coordinate system at origin of frame_b fixed at connecting rod of middle and right revolute joint
Frame_b	frame_im	Coordinate system at origin of revolute joint in the middle fixed at connecting rod of middle and right revolute joint
Flange_a	axis	1-dim. rotational flange that drives the right revolute joint at frame_b
Flange_b	bearing	1-dim. rotational flange of the drive bearing of the right revolute joint at frame_b

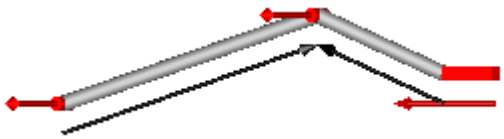
Modelica.Mechanics.MultiBody.Joints.Assemblies.JointRRP



Planar revolute - revolute - prismatic joint aggregation (no constraints, no potential states)

Information

This component consists of **2 revolute** joints with parallel axes of rotation that and a **prismatic** joint with a translational axis that is orthogonal to the revolute joint axes, see the default animation in the following figure (the axes vectors are not part of the default animation):



This joint aggregation introduces neither constraints nor state variables and should therefore be used in kinematic loops whenever possible to avoid non-linear systems of equations. It is only meaningful to use this component in **planar loops**. Basically, the position and orientation of the 3 joints as well as of frame_ia, frame_ib, and frame_im are calculated by solving analytically a non-linear equation, given the position and orientation at frame_a and at frame_b.

Connector **frame_a** is the "left" side of the first revolute joint whereas **frame_ia** is the "right side of this revolute joint, fixed in rod 1. Connector **frame_b** is the "right" side of the prismatic joint whereas **frame_ib** is the "left" side of this prismatic joint, fixed in rod 2. Finally, connector **frame_im** is the connector at the "right" side of the revolute joint in the middle, fixed in rod 2. The frames frame_b, frame_ib, frame_im are always parallel to each other.

The easiest way to define the parameters of this joint is by moving the MultiBody system in a **reference configuration** where **all frames** of all components are **parallel** to each other (alternatively, at least frame_a, frame_ia, frame_im, frame_ib, frame_b of the JointRRP joint should be parallel to each other when defining an instance of this component).

Basically, the JointRRP model consists internally of a universal - spherical - prismatic joint aggregation (= JointUSP). In a planar loop this will behave as if 2 revolute joints with parallel axes and 1 prismatic joint are connected by rigid rods.

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Axis	n_a	{0,0,1}	Axes of the two revolute joints resolved in frame_a (both axes are parallel to each other)
Axis	n_b	{-1,0,0}	Axis of prismatic joint fixed and resolved in frame_b (must be orthogonal to revolute joint axes)
Position	rRod1_ia[3]	{1,0,0}	Vector from origin of frame_a to revolute joint in the middle, resolved in frame_ia [m]
Position	rRod2_ib[3]	{-1,0,0}	Vector from origin of frame_ib to revolute joint in the middle, resolved in frame_ib (frame_ib is parallel to frame_b) [m]

Position	s_offset	0	Relative distance offset of prismatic joint (distance between the prismatic joint frames = s(t) + s_offset) [m]
Position	s_guess	0	Select the configuration such that at initial time |s(t0)-s_guess|is minimal [m]
Animation			
if animation = true			
Distance	cylinderLength	world.defaultJointLength	Length of cylinders representing the revolute joints [m]
Distance	cylinderDiameter	world.defaultJointWidth	Diameter of cylinders representing the revolute joints [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinders representing the revolute joints
Axis	boxWidthDirection	{0,1,0}	Vector in width direction of prismatic joint, resolved in frame_b
Distance	boxWidth	world.defaultJointWidth	Width of prismatic joint box [m]
Distance	boxHeight	boxWidth	Height of prismatic joint box [m]
Color	boxColor	cylinderColor	Color of prismatic joint box
Diameter	rodDiameter	1.1*cylinderDiameter	Diameter of the two rods connecting the joints [m]
Color	rodColor	Modelica.Mechanics.MultiBody...	Color of the two rods connecting the joints
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	checkTotalPower	false	= true, if total power flowing into this component shall be determined (must be zero)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_a	frame_ia	Coordinate system at origin of frame_a fixed at connecting rod of revolute joints
Frame_b	frame_ib	Coordinate system at origin of frame_b fixed at connecting rod of revolute and prismatic joint
Frame_b	frame_im	Coordinate system at origin of revolute joint in the middle fixed at connecting rod of revolute and prismatic joint
Flange_a	axis	1-dim. translational flange that drives the prismatic joint
Flange_b	bearing	1-dim. translational flange of the drive bearing of the prismatic joint

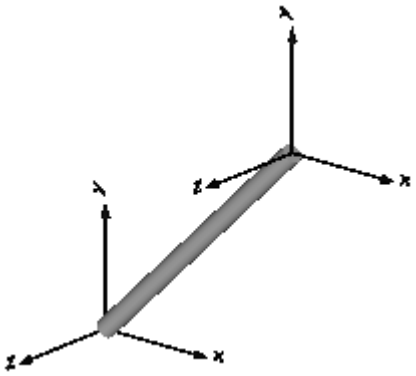
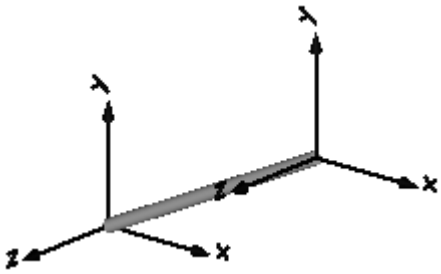
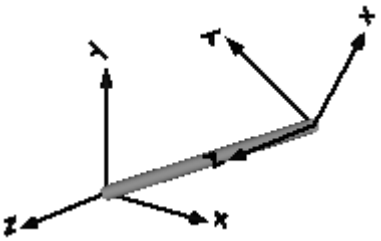
Modelica.Mechanics.MultiBody.Parts

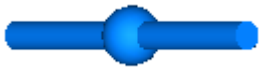
Rigid components such as bodies with mass and inertia and massless rods

Information

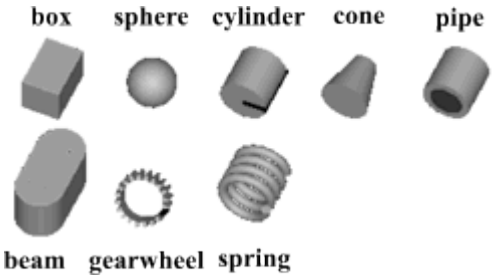
Package **Parts** contains **rigid components** of a multi-body system. These components may be used to build up more complicated structures. For example, a part may be built up of a "Body" and of several "FixedTranslation" components.

Content

Model	*Description*
Fixed	Frame fixed in world frame at a given position. It is visualized with a shape, see **shapeType** below (the frames on the two sides do not belong to the component): 
FixedTranslation	Fixed translation of frame_b with respect to frame_a. It is visualized with a shape, see **shapeType** below (the frames on the two sides do not belong to the component): 
FixedRotation	Fixed translation and fixed rotation of frame_b with respect to frame_a It is visualized with a shape, see **shapeType** below (the frames on the two sides do not belong to the component): 
Body	Rigid body with mass, inertia tensor and one frame connector. It is visualized with a cylinder and a sphere at the center of mass:

BodyShape	Rigid body with mass, inertia tensor, different shapes (see **shapeType** below) for animation, and two frame connectors: 
Fixed BodyBox	Rigid body with box shape (mass and animation properties are computed from box data and from density):
BodyCylinder	Rigid body with cylinder shape (mass and animation properties are computed from cylinder data and from density):
PointMass	Rigid body where inertia tensor and rotation is neglected:
Mounting1D	Propagate 1-dim. support torque to 3-dim. system
Rotor1D	1D inertia attachable on 3-dim. bodies (without neglecting dynamic effects)
BevelGear1D	1D gearbox with arbitrary shaft directions (3D bearing frame)

Components **Fixed**, **FixedTranslation**, **FixedRotation** and **BodyShape** are visualized according to parameter **shapeType**, that may have the following values (e.g., shapeType = "box"):



All the details of the visualization shape parameters are given in Visualizers.FixedShape

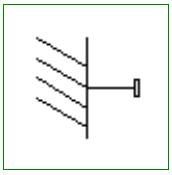
Colors in all animation parts are defined via parameter **color**. This is an Integer vector with 3 elements, {r, g, b}, and specifies the color of the shape. {r,g,b} are the "red", "green" and "blue" color parts, given in the ranges 0 .. 255, respectively. The predefined type **MultiBody.Types.Color** contains a menu definition of the colors used in the MultiBody library (this will be replaced by a color editor).

Package Content

Name	Description
Fixed	Frame fixed in the world frame at a given position

FixedTranslation	Fixed translation of frame_b with respect to frame_a
FixedRotation	Fixed translation followed by a fixed rotation of frame_b with respect to frame_a
Body	Rigid body with mass, inertia tensor and one frame connector (12 potential states)
BodyShape	Rigid body with mass, inertia tensor, different shapes for animation, and two frame connectors (12 potential states)
BodyBox	Rigid body with box shape. Mass and animation properties are computed from box data and density (12 potential states)
BodyCylinder	Rigid body with cylinder shape. Mass and animation properties are computed from cylinder data and density (12 potential states)
PointMass	Rigid body where body rotation and inertia tensor is neglected (6 potential states)
Mounting1D	Propagate 1-dim. support torque to 3-dim. system (provided world.driveTrainMechanics3D=true; default=false)
Rotor1D	1D inertia attachable on 3-dim. bodies (3D dynamic effects are taken into account if world.driveTrainMechanics3D=true; default=false)
BevelGear1D	1D gearbox with arbitrary shaft directions and 3-dim. bearing frame (3D dynamic effects are taken into account provided world.driveTrainMechanics3D=true; default=false)

Modelica.Mechanics.MultiBody.Parts.Fixed

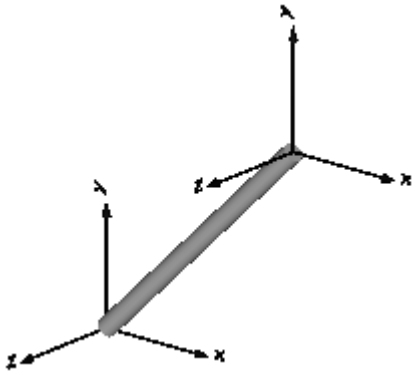


Frame fixed in the world frame at a given position

Information

Element consisting of a frame (frame_b) that is fixed in the world frame at a given position defined by parameter vector **r** (vector from origin of world frame to frame_b, resolved in the world frame).

By default, this component is visualized by a cylinder connecting the world frame and frame_b of this components, as shown in the figure below. Note, that the visualized world frame on the left side and Fixed.frame_b on the right side are not part of the component animation and that the animation may be switched off via parameter animation = **false**.



Parameters

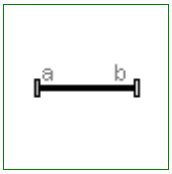
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Position	r[3]	{0,0,0}	Position vector from world frame to frame_b, resolved in world

			frame [m]
Animation			
if animation = true			
ShapeType	shapeType	"cylinder"	Type of shape
Position	r_shape[3]	{0,0,0}	Vector from world frame to shape origin, resolved in world frame [m]
Position	lengthDirection[3]	r - r_shape	Vector in length direction of shape, resolved in world frame [m]
Position	widthDirection[3]	{0,1,0}	Vector in width direction of shape, resolved in world frame [m]
Length	length	Frames.length(r - r_shape)	Length of shape [m]
Distance	width	length/world.defaultWidthFra...	Width of shape [m]
Distance	height	width	Height of shape [m]
ShapeExtra	extra	0.0	Additional parameter for cone, pipe etc. (see docu of Visualizers.Advanced.Shape)
Color	color	Modelica.Mechanics.MultiBody. ..	Color of shape
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_b	frame_b	Coordinate system fixed in the world frame

Modelica.Mechanics.MultiBody.Parts.FixedTranslation

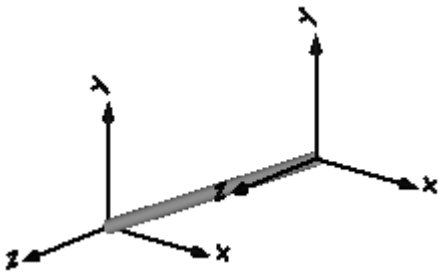


Fixed translation of frame_b with respect to frame_a

Information

Component for a **fixed translation** of frame_b with respect to frame_a, i.e., the relationship between connectors frame_a and frame_b remains constant and frame_a is always **parallel** to frame_b.

By default, this component is visualized by a cylinder connecting frame_a and frame_b, as shown in the figure below. Note, that the two visualized frames are not part of the component animation and that the animation may be switched off via parameter animation = **false**.



Parameters

Type	Name	Default	Description

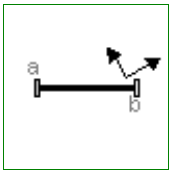
Boolean	animation	true	= true, if animation shall be enabled
Position	r[3]	{0,0,0}	Vector from frame_a to frame_b resolved in frame_a [m]
Animation			
if animation = true			
ShapeType	shapeType	"cylinder"	Type of shape
Position	r_shape[3]	{0,0,0}	Vector from frame_a to shape origin, resolved in frame_a [m]
Axis	lengthDirection	r - r_shape	Vector in length direction of shape, resolved in frame_a
Axis	widthDirection	{0,1,0}	Vector in width direction of shape, resolved in frame_a
Length	length	Frames.length(r - r_shape)	Length of shape [m]
Distance	width	length/world.defaultWidthFra...	Width of shape [m]
Distance	height	width	Height of shape. [m]
ShapeExtra	extra	0.0	Additional parameter depending on shapeType (see docu of Visualizers.Advanced.Shape).
Color	color	Modelica.Mechanics.MultiBody. ..	Color of shape
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Parts.FixedRotation



Fixed translation followed by a fixed rotation of frame_b with respect to frame_a

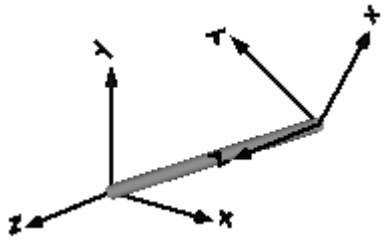
Information

Component for a **fixed translation** and **fixed rotation** of frame_b with respect to frame_a, i.e., the relationship between connectors frame_a and frame_b remains constant. There are several possibilities to define the orientation of frame_b with respect to frame_a:

- **Planar rotation** along axis 'n' (that is fixed and resolved in frame_a) with a fixed angle 'angle'.
- **Vectors n_x** and **n_y** that are directed along the corresponding axes direction of frame_b and are resolved in frame_a (if n_y is not orthogonal to n_x, the y-axis of frame_b is selected such that it is orthogonal to n_x and in the plane of n_x and n_y).
- **Sequence** of **three planar axes rotations**. For example, "sequence = {1,2,3}" and "angles = {90, 45, -90}" means to rotate frame_a around the x axis with 90 degrees, around the new y axis with 45 degrees and around the new z axis around -90 degrees to arrive at frame_b. Note, that sequence={1,2,3} is the Cardan angle sequence and sequence = {3,1,3} is the Euler angle sequence.

By default, this component is visualized by a cylinder connecting frame_a and frame_b, as shown in the figure below. In this figure frame_b is rotated along the z-axis of frame_a with 60 degree. Note, that the two visualized frames are not part of the component animation and that the animation may be switched off via

parameter animation = **false**.



Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
Position	r[3]	{0,0,0}	Vector from frame_a to frame_b resolved in frame_a [m]
Temp	rotationType	Modelica.Mechanics.MultiBody. ..	Type of rotation description
if rotationType = RotationAxis			
Axis	n	{1,0,0}	Axis of rotation in frame_a (= same as in frame_b)
Angle_deg	angle	0	Angle to rotate frame_a around axis n into frame_b [deg]
if rotationType = TwoAxesVectors			
Axis	n_x	{1,0,0}	Vector along x-axis of frame_b resolved in frame_a
Axis	n_y	{0,1,0}	Vector along y-axis of frame_b resolved in frame_a
if rotationType = PlanarRotationSequence			
RotationSequence	sequence	{1,2,3}	Sequence of rotations
Angle_deg	angles[3]	{0,0,0}	Rotation angles around the axes defined in 'sequence' [deg]
Animation			
if animation = true			
ShapeType	shapeType	"cylinder"	Type of shape
Position	r_shape[3]	{0,0,0}	Vector from frame_a to shape origin, resolved in frame_a [m]
Axis	lengthDirection	r - r_shape	Vector in length direction of shape, resolved in frame_a
Axis	widthDirection	{0,1,0}	Vector in width direction of shape, resolved in frame_a
Length	length	Frames.length(r - r_shape)	Length of shape [m]
Distance	width	length/world.defaultWidthFra...	Width of shape [m]
Distance	height	width	Height of shape. [m]
ShapeExtra	extra	0.0	Additional parameter depending on shapeType (see docu of Visualizers.Advanced.Shape).
Color	color	Modelica.Mechanics.MultiBody. ..	Color of shape

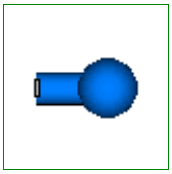
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Parts.Body



Rigid body with mass, inertia tensor and one frame connector (12 potential states)

Information

Rigid body with mass and inertia tensor. All parameter vectors have to be resolved in frame_a. The **inertia tensor** has to be defined with respect to a coordinate system that is parallel to frame_a with the origin at the center of mass of the body.

By default, this component is visualized by a **cylinder** located between frame_a and the center of mass and by a **sphere** that has its center at the center of mass. If the cylinder length is smaller as the radius of the sphere, e.g., since frame_a is located at the center of mass, the cylinder is not displayed. Note, that the animation may be switched off via parameter animation = **false**.

States of Body Components

Every body has potential states. If possible a tool will select the states of joints and not the states of bodies because this is usually the most efficient choice. In this case the position, orientation, velocity and angular velocity of frame_a of the body will be computed by the component that is connected to frame_a. However, if a body is moving freely in space, variables of the body have to be used as states. The potential states of the body are:

- The **position vector** frame_a.r_0 from the origin of the world frame to the origin of frame_a of the body, resolved in the world frame and the **absolute velocity** v_0 of the origin of frame_a, resolved in the world frame (= der(frame_a.r_0)).
- If parameter **useQuaternions** in the "Advanced" menu is **true** (this is the default), then **4 quaternions** are potential states. Additionally, the coordinates of the absolute angular velocity vector of the body are 3 potential states.
  If **useQuaternions** in the "Advanced" menu is **false**, then **3 angles** and the derivatives of these angles are potential states. The orientation of frame_a is computed by rotating the world frame along the axes defined in parameter vector "sequence_angleStates" (default = {1,2,3}, i.e., the Cardan angle sequence) around the angles used as potential states. For example, the default is to rotate the x-axis of the world frame around angles[1], the new y-axis around angles[2] and the new z-axis around angles[3], arriving at frame_a.

The quaternions have the slight disadvantage that there is a non-linear constraint equation between the 4 quaternions. Therefore, at least one non-linear equation has to be solved during simulation. A tool might, however, analytically solve this simple constraint equation. Using the 3 angles as states has the disadvantage that there is a singular configuration in which a division by zero will occur. If it is possible to determine in advance for an application class that this singular configuration is outside of the operating region, the 3 angles might be used as potential states by setting **useQuaternions** = **false**.

In text books about 3-dimensional mechanics often 3 angles and the angular velocity are used as states. This is not the case here, since 3 angles and their derivatives are used as potential states (if useQuaternions = false). The reason is that for real-time simulation the discretization formula of the integrator might be

"inlined" and solved together with the body equations. By appropriate symbolic transformation the performance is drastically increased if angles and their derivatives are used as states, instead of angles and the angular velocity.

Whether or not variables of the body are used as states is usually automatically selected by the Modelica translator. If parameter **enforceStates** is set to **true** in the "Advanced" menu, then body variables are forced to be used as states according to the setting of parameters "useQuaternions" and "sequence_angleStates".

Parameters

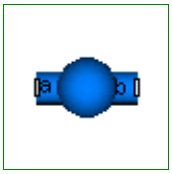
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show cylinder and sphere)
Position	r_CM[3]	{0,0,0}	Vector from frame_a to center of mass, resolved in frame_a [m]
Mass	m	1	Mass of rigid body [kg]
Inertia tensor (resolved in center of mass, parallel to frame_a)			
Inertia	I_11	0.001	(1,1) element of inertia tensor [kg.m2]
Inertia	I_22	0.001	(2,2) element of inertia tensor [kg.m2]
Inertia	I_33	0.001	(3,3) element of inertia tensor [kg.m2]
Inertia	I_21	0	(2,1) element of inertia tensor [kg.m2]
Inertia	I_31	0	(3,1) element of inertia tensor [kg.m2]
Inertia	I_32	0	(3,2) element of inertia tensor [kg.m2]
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody..	Type of initialization (defines usage of start values below)
Position	r_0_start[3]	{0,0,0}	Initial values of frame_a.r_0 (vector from origin of world frame to origin of frame_a resolved in world frame) [m]
RotationSequence	sequence_start	{1,2,3}	Sequence of rotations to rotate world frame into frame_a at initial time
Angle_deg	angles_start[3]	{0,0,0}	Initial values of angles to rotate world frame around 'sequence_start' axes into frame_a [deg]
Velocity	v_0_start[3]	{0,0,0}	Initial values of velocity v_0 = der(frame_a.r_0) [m/s]
AngularVelocity_degs	w_0_start[3]	{0,0,0}	Initial values of angular velocity of frame_a resolved in world frame [deg/s]
Acceleration	a_0_start[3]	{0,0,0}	Initial values of acceleration a_0 = der(v_0) [m/s2]
AngularAcceleration_	z_0_start[3]	{0,0,0}	Initial values of angular

degs2			acceleration z_0 = der(w_0) [deg/s2]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultBodyDiameter	Diameter of sphere [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of sphere
Diameter	cylinderDiameter	sphereDiameter/Types.Default...	Diameter of cylinder [m]
Color	cylinderColor	sphereColor	Color of cylinder
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if absolute variables of body object shall be used as states (StateSelect.always)
Boolean	useQuaternions	true	= true, if quaternions shall be used as potential states otherwise use 3 angles as potential states
RotationSequence	sequence_angleStates	{1,2,3}	Sequence of rotations to rotate world frame into frame_a around the 3 angles used as potential states

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed at body

Modelica.Mechanics.MultiBody.Parts.BodyShape

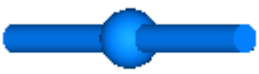


Rigid body with mass, inertia tensor, different shapes for animation, and two frame connectors (12 potential states)

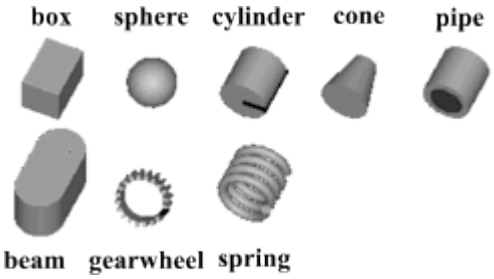
Information

Rigid body with mass and inertia tensor and **two frame connectors**. All parameter vectors have to be resolved in frame_a. The **inertia tensor** has to be defined with respect to a coordinate system that is parallel to frame_a with the origin at the center of mass of the body. The coordinate system **frame_b** is always parallel to **frame_a**.

By default, this component is visualized by any **shape** that can be defined with Modelica.Mechanics.MultiBody.Visualizers.FixedShape. This shape is placed between frame_a and frame_b (default: length(shape) = Frames.length(r)). Additionally a **sphere** may be visualized that has its center at the center of mass. Note, that the animation may be switched off via parameter animation = **false**.



The following shapes can be defined via parameter **shapeType**, e.g., shapeType="cone":



A BodyShape component has potential states. For details of these states and of the "Advanced" menu parameters, see model MultiBody.Parts.Body.

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show shape between frame_a and frame_b and optionally a sphere at the center of mass)
Boolean	animateSphere	true	= true, if mass shall be animated as sphere provided animation=true
Position	r[3]	{0,0,0}	Vector from frame_a to frame_b resolved in frame_a [m]
Position	r_CM[3]	{0,0,0}	Vector from frame_a to center of mass, resolved in frame_a [m]
Mass	m	1	Mass of rigid body [kg]
Inertia tensor (resolved in center of mass, parallel to frame_a)			
Inertia	I_11	0.001	(1,1) element of inertia tensor [kg.m2]
Inertia	I_22	0.001	(2,2) element of inertia tensor [kg.m2]
Inertia	I_33	0.001	(3,3) element of inertia tensor [kg.m2]
Inertia	I_21	0	(2,1) element of inertia tensor [kg.m2]
Inertia	I_31	0	(3,1) element of inertia tensor [kg.m2]
Inertia	I_32	0	(3,2) element of inertia tensor [kg.m2]
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody. ..	Type of initialization (defines usage of start values below)
Position	r_0_start[3]	{0,0,0}	Initial values of frame_a.r_0 (vector from origin of world frame to origin of frame_a resolved in world frame) [m]
RotationSequence	sequence_start	{1,2,3}	Sequence of rotations to rotate world frame into frame_a at initial time
Angle_deg	angles_start[3]	{0,0,0}	Initial values of angles to rotate world frame around

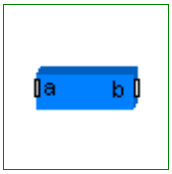
			'sequence_start' axes into frame_a [deg]
Velocity	v_0_start[3]	{0,0,0}	Initial values of velocity v_0 = der(frame_a.r_0) [m/s]
AngularVelocity_degs	w_0_start[3]	{0,0,0}	Initial values of angular velocity of frame_a resolved in world frame [deg/s]
Acceleration	a_0_start[3]	{0,0,0}	Initial values of acceleration a_0 = der(v_0) [m/s2]
AngularAcceleration_degs2	z_0_start[3]	{0,0,0}	Initial values of angular acceleration z_0 = der(w_0) [deg/s2]
Animation			
if animation = true			
ShapeType	shapeType	"cylinder"	Type of shape
Position	r_shape[3]	{0,0,0}	Vector from frame_a to shape origin, resolved in frame_a [m]
Axis	lengthDirection	r - r_shape	Vector in length direction of shape, resolved in frame_a
Axis	widthDirection	{0,1,0}	Vector in width direction of shape, resolved in frame_a
Length	length	Frames.length(r - r_shape)	Length of shape [m]
Distance	width	length/world.defaultWidthFra...	Width of shape [m]
Distance	height	width	Height of shape. [m]
ShapeExtra	extra	0.0	Additional parameter depending on shapeType (see docu of Visualizers.Advanced.Shape).
Color	color	Modelica.Mechanics.MultiBody. ..	Color of shape
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
if animation = true and animateSphere = true			
Diameter	sphereDiameter	2*width	Diameter of sphere [m]
Color	sphereColor	color	Color of sphere of mass
Advanced			
Boolean	enforceStates	false	= true, if absolute variables of body object shall be used as states (StateSelect.always)
Boolean	useQuaternions	true	= true, if quaternions shall be used as potential states otherwise use 3 angles as potential states
RotationSequence	sequence_angleStates	{1,2,3}	Sequence of rotations to rotate world frame into frame_a around the 3 angles used as potential states

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque

Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

Modelica.Mechanics.MultiBody.Parts.BodyBox



Rigid body with box shape. Mass and animation properties are computed from box data and density (12 potential states)

Information

Rigid body with **box** shape. The mass properties of the body (mass, center of mass, inertia tensor) are computed from the box data. Optionally, the box may be hollow. The (outer) box shape is by default used in the animation. The hollow part is not shown in the animation. The two connector frames **frame_a** and **frame_b** are always parallel to each other. Example of component animation (note, that the animation may be switched off via parameter animation = **false**):

A BodyBox component has potential states. For details of these states and of the "Advanced" menu parameters, see model MultiBody.Parts.Body.

Parameters

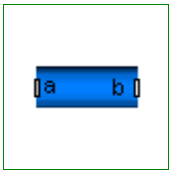
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show box between frame_a and frame_b)
Position	r[3]	{0.1,0,0}	Vector from frame_a to frame_b resolved in frame_a [m]
Position	r_shape[3]	{0,0,0}	Vector from frame_a to box origin, resolved in frame_a [m]
Axis	lengthDirection	r - r_shape	Vector in length direction of box, resolved in frame_a
Axis	widthDirection	{0,1,0}	Vector in width direction of box, resolved in frame_a
Length	length	Frames.length(r - r_shape)	Length of box [m]
Distance	width	length/world.defaultWidthFra...	Width of box [m]
Distance	height	width	Height of box [m]
Distance	innerWidth	0	Width of inner box surface (0 <= innerWidth <= width) [m]
Distance	innerHeight	innerWidth	Height of inner box surface (0 <= innerHeight <= height) [m]
Real	density	7.7	Density of box (e.g., steel: 7.7 .. 7.9, wood : 0.4 .. 0.8) [g/cm3]
Color	color	Modelica.Mechanics.MultiBody...	Color of box
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody..	Type of initialization (defines

		.	usage of start values below)
Position	r_0_start[3]	{0,0,0}	Initial values of frame_a.r_0 (vector from origin of world frame to origin of frame_a resolved in world frame) [m]
RotationSequence	sequence_start	{1,2,3}	Sequence of rotations to rotate world frame into frame_a at initial time
Angle_deg	angles_start[3]	{0,0,0}	Initial values of angles to rotate world frame around 'sequence_start' axes into frame_a [deg]
Velocity	v_0_start[3]	{0,0,0}	Initial values of velocity v_0 = der(frame_a.r_0) [m/s]
AngularVelocity_degs	w_0_start[3]	{0,0,0}	Initial values of angular velocity of frame_a resolved in world frame [deg/s]
Acceleration	a_0_start[3]	{0,0,0}	Initial values of acceleration a_0 = der(v_0) [m/s2]
AngularAcceleration_degs2	z_0_start[3]	{0,0,0}	Initial values of angular acceleration z_0 = der(w_0) [deg/s2]
Advanced			
Boolean	enforceStates	false	= true, if absolute variables of body object shall be used as states (StateSelect.always)
Boolean	useQuaternions	true	= true, if quaternions shall be used as potential states otherwise use 3 angles as potential states
RotationSequence	sequence_angleStates	{1,2,3}	Sequence of rotations to rotate world frame into frame_a around the 3 angles used as potential states

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

Modelica.Mechanics.MultiBody.Parts.BodyCylinder



Rigid body with cylinder shape. Mass and animation properties are computed from cylinder data and density (12 potential states)

Information

Rigid body with **cylinder** shape. The mass properties of the body (mass, center of mass, inertia tensor) are computed from the cylinder data. Optionally, the cylinder may be hollow. The cylinder shape is by default used in the animation. The two connector frames **frame_a** and **frame_b** are always parallel to each other. Example of component animation (note, that the animation may be switched off via parameter animation = **false**):

A BodyCylinder component has potential states. For details of these states and of the "Advanced" menu parameters, see model MultiBody.Parts.Body.

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show cylinder between frame_a and frame_b)
Position	r[3]	{0.1,0,0}	Vector from frame_a to frame_b, resolved in frame_a [m]
Position	r_shape[3]	{0,0,0}	Vector from frame_a to cylinder origin, resolved in frame_a [m]
Axis	lengthDirection	r - r_shape	Vector in length direction of cylinder, resolved in frame_a
Length	length	Frames.length(r - r_shape)	Length of cylinder [m]
Distance	diameter	length/world.defaultWidthFra...	Diameter of cylinder [m]
Distance	innerDiameter	0	Inner diameter of cylinder (0 <= innerDiameter <= Diameter) [m]
Real	density	7.7	Density of cylinder (e.g., steel: 7.7 .. 7.9, wood : 0.4 .. 0.8) [g/cm3]
Color	color	Modelica.Mechanics.MultiBody...	Color of cylinder
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)
Position	r_0_start[3]	{0,0,0}	Initial values of frame_a.r_0 (vector from origin of world frame to origin of frame_a resolved in world frame) [m]
RotationSequence	sequence_start	{1,2,3}	Sequence of rotations to rotate world frame into frame_a at initial time
Angle_deg	angles_start[3]	{0,0,0}	Initial values of angles to rotate world frame around 'sequence_start' axes into frame_a [deg]
Velocity	v_0_start[3]	{0,0,0}	Initial values of velocity v_0 = der(frame_a.r_0) [m/s]
AngularVelocity_degs	w_0_start[3]	{0,0,0}	Initial values of angular velocity of frame_a resolved in world frame [deg/s]
Acceleration	a_0_start[3]	{0,0,0}	Initial values of acceleration

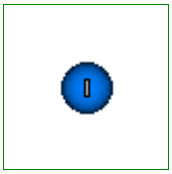
			a_0 = der(v_0) [m/s2]
AngularAcceleration_degs2	z_0_start[3]	{0,0,0}	Initial values of angular acceleration z_0 = der(w_0) [deg/s2]
Advanced			
Boolean	enforceStates	false	= true, if absolute variables of body object shall be used as states (StateSelect.always)
Boolean	useQuaternions	true	= true, if quaternions shall be used as potential states otherwise use 3 angles as potential states
RotationSequence	sequence_angleStates	{1,2,3}	Sequence of rotations to rotate world frame into frame_a around the 3 angles used as potential states

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque

---

Modelica.Mechanics.MultiBody.Parts.PointMass



Rigid body where body rotation and inertia tensor is neglected (6 potential states)

Information

Rigid body where the inertia tensor is neglected. This body is solely defined by its mass. By default, this component is visualized by a **sphere** that has its center at frame_a. Note, that the animation may be switched off via parameter animation = **false**.

Every PointMass has potential states. If possible a tool will select the states of joints and not the states of PointMasss because this is usually the most efficient choice. In this case the position and velocity of frame_a of the body will be computed by the component that is connected to frame_a. However, if a PointMass is moving freely in space, variables of the PointMass have to be used as states. The potential states are: The **position vector** frame_a.r_0 from the origin of the world frame to the origin of frame_a of the body, resolved in the world frame and the **absolute velocity** v_0 of the origin of frame_a, resolved in the world frame (= der(frame_a.r_0)).

Whether or not variables of the body are used as states is usually automatically selected by the Modelica translator. If parameter **enforceStates** is set to **true** in the "Advanced" menu, then PointMass variables frame_a.r_0 and der(frame_a.r_0) are forced to be used as states.

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show sphere)
Mass	m		Mass of mass point [kg]
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody...	Type of initialization (defines usage of start values below)

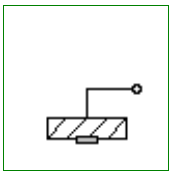
Position	r_0_start[3]	{0,0,0}	Initial values of frame_a.r_0 (vector from origin of world frame to origin of frame_a resolved in world frame) [m]
Velocity	v_0_start[3]	{0,0,0}	Initial values of velocity v_0 = der(frame_a.r_0) [m/s]
Acceleration	a_0_start[3]	{0,0,0}	Initial values of acceleration a_0 = der(v_0) [m/s2]
Animation			
if animation = true			
Diameter	sphereDiameter	world.defaultBodyDiameter	Diameter of sphere [m]
Color	sphereColor	Modelica.Mechanics.MultiBody...	Color of sphere
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if frame_a.r_0 and v_0 of body object shall be used as states (StateSelect.always)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed at center of mass point

---

Modelica.Mechanics.MultiBody.Parts.Mounting1D



Propagate 1-dim. support torque to 3-dim. system (provided world.driveTrainMechanics3D=true; default=false)

Information

This component is used to acquire support torques from a 1-dim.-rotational mechanical system (e.g., components from Modelica.Mechanics.Rotational) and to propagate them to a carrier body.

The 1-dim. support torque at `flange_b` is transformed into 3-dim. space under consideration of the rotation axis, parameter `n`, which has to be given in the local coordinate system of `frame_a`.

All components of a 1-dim.-rotational mechanical system that are connected to **a** common **Mounting1D** element need to have the same axis of rotation along parameter vector `n`. This means that, e.g., bevel gears where the axis of rotation of `flange_a` and `flange_b` are different cannot be described properly by connecting to the **Mounting1D** component. In this case, a combination of several **Mounting1D** components or the component **BevelGear1D** should be used.

Reference

Schweiger, Christian ; Otter, Martin: Modelling 3D Mechanical Effects of 1-dim. Powertrains. In: *Proceedings of the 3rd International Modelica Conference*. Linköping : The Modelica Association and Linköping University, November 3-4, 2003, pp. 149-158

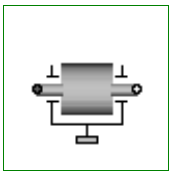
Parameters

Type	Name	Default	Description
Angle	phi0	0	Fixed offset angle of housing [rad]
Axis	n	{1,0,0}	Axis of rotation = axis of support torque (resolved in frame_a)

Connectors

Type	Name	Description
Flange_b	flange_b	(right) flange fixed in housing
Frame_a	frame_a	Frame in which housing is fixed (connector is removed, if world.driveTrainMechanics3D=false)

Modelica.Mechanics.MultiBody.Parts.Rotor1D



1D inertia attachable on 3-dim. bodies (3D dynamic effects are taken into account if world.driveTrainMechanics3D=true; default=false)

Information

This component is used to model the gyroscopic torques exerted by a 1-dim. inertia (so called *rotor*) on its 3-dim. carrier body. Gyroscopic torques appear, if the vector of the carrier body's angular velocity is not parallel to the vector of the rotor's. The axis of rotation of the rotor is defined by the parameter `n`, which has to be given in the local coordinate system of `frame_a`. The default animation of this component is shown in the figure below.

This component is a replacement for Modelica.Mechanics.Rotational.Inertia for the case, that a 1-dim.-rotational mechanical system should be attached with a 3-dim. carrier body.

The Boolean parameter `exact` was introduced due to performance reasons. If `exact` is set to true, the influence of the carrier body motion on the angular velocity of the rotor is neglected. This influence is usually negligible if the 1-dim.-rotational mechanical system accelerates much faster as the base body (this is, e.g., the case in vehicle powertrains). The essential advantage is that an algebraic loop is removed since then there is only an action on acceleration level from the powertrain to the base body but not vice versa.

Reference

Schweiger, Christian ; Otter, Martin: Modelling 3D Mechanical Effects of 1-dim. Powertrains. In: *Proceedings of the 3rd International Modelica Conference*. Linköping : The Modelica Association and Linköping University, November 3-4, 2003, pp. 149-158

Parameters

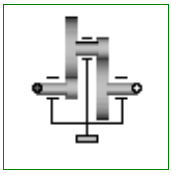
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show rotor as cylinder)
Inertia	J	1	Moment of inertia of rotor around its axis of rotation [kg.m2]
Axis	n	{1,0,0}	Axis of rotation resolved in frame_a
Initialization			
Temp	initType	Modelica.Mechanics.MultiBody..	Type of initialization (defines usage of start values below)
Angle_deg	phi_start	0	Initial value of rotor rotation angle phi (fixed or guess value) [deg]

AngularVelocity_degs	w_start	0	Initial value of relative angular velocity w = der(phi) [deg/s]
AngularAcceleration_degs2	a_start	0	Initial value of relative angular acceleration a = der(w) [deg/s2]
Animation			
if animation = true			
Position	r_center[3]	zeros(3)	Position vector from origin of frame_a to center of cylinder [m]
Distance	cylinderLength	2*world.defaultJointLength	Length of cylinder representing the rotor [m]
Distance	cylinderDiameter	2*world.defaultJointWidth	Diameter of cylinder representing the rotor [m]
Color	cylinderColor	Modelica.Mechanics.MultiBody...	Color of cylinder representing the rotor
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Boolean	enforceStates	false	= true, if rotor angle (phi) and rotor speed (w) shall be used as states
Boolean	exact	true	= true, if exact calculations; false if influence of bearing on rotor acceleration is neglected to avoid an algebraic loop

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
Frame_a	frame_a	Frame in which rotor housing is fixed (connector is removed, if world.driveTrainMechanics3D=false)

Modelica.Mechanics.MultiBody.Parts.BevelGear1D



1D gearbox with arbitrary shaft directions and 3-dim. bearing frame (3D dynamic effects are taken into account provided world.driveTrainMechanics3D=true; default=false)

Information

This component is used to model a 1-dim. gearbox with non-parallel axes (defined by parameters `n_a`, `n_b`). A 3-dim. `bearing` frame is necessary to reflect the correct support torque, as the axes of rotation of `flange_a` and `flange_b` and the direction of the support torque vector are different in general.

Note: The name BevelGear1D is kept only for simplicity. Regardless, this component could be used to model any kind of gearbox with non-parallel axes.

Reference

Schweiger, Christian ; Otter, Martin: Modelling 3D Mechanical Effects of 1-dim. Powertrains. In: *Proceedings of the 3rd International Modelica Conference*. Linköping : The Modelica Association and Linköping University, November 3-4, 2003, pp. 149-158

Parameters

Type	Name	Default	Description
Real	ratio	1	Gear speed ratio
Axis	n_a	{1,0,0}	Axis of rotation of flange_a, resolved in frame_a
Axis	n_b	{1,0,0}	Axis of rotation of flange_b, resolved in frame_a

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Frame_a	frame_a	Bearing frame

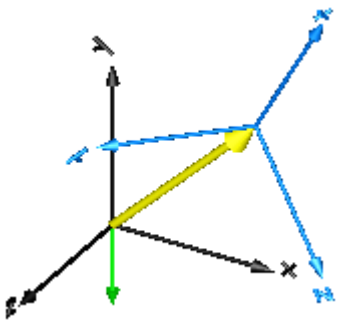
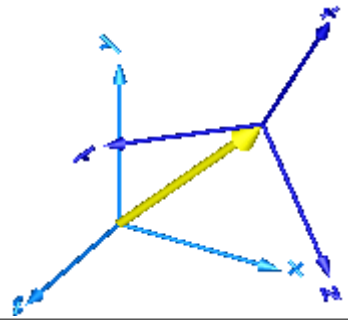
Modelica.Mechanics.MultiBody.Sensors

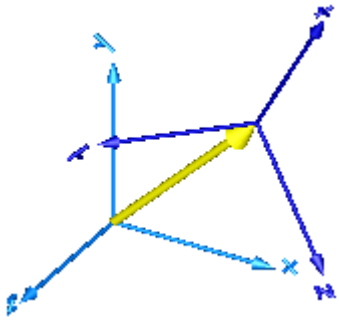
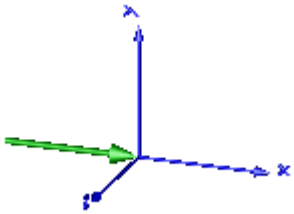
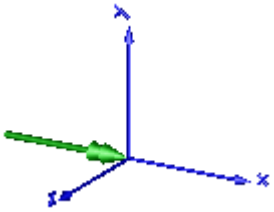
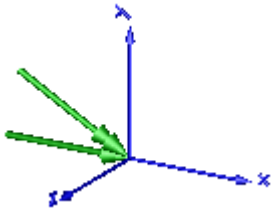
Sensors to measure variables

Information

Package **Sensors** contains **ideal measurement** components to determine absolute and relative kinematic quantities, as well as cut-forces and cut-torques. All measured quantities can be provided in every desired coordinate system.

Content

Model	*Description*
AbsoluteSensor	Measure absolute kinematic quantities of a frame connector 
RelativeSensor	Measure relative kinematic quantities between two frame connectors 
Distance	Measure distance between the origins of two frame connectors

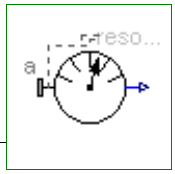
	
CutForce	Measure cut force vector 
CutTorque	Measure cut torque vector 
CutForceAndTorque	Measure cut force and cut torque vector 
Power	Measure power flowing from frame_a to frame_b

Package Content

Name	Description
AbsoluteSensor	Measure absolute kinematic quantities of a frame connector
RelativeSensor	Measure relative kinematic quantities between two frame connectors
Distance	Measure the distance between the origins of two frame connectors
CutForce	Measure cut force vector
CutTorque	Measure cut torque vector
CutForceAndTorque	Measure cut force and cut torque vector
Power	Measure power flowing from frame_a to frame_b

Modelica.Mechanics.MultiBody.Sensors.AbsoluteSensor

Measure absolute kinematic quantities of a frame connector



Information

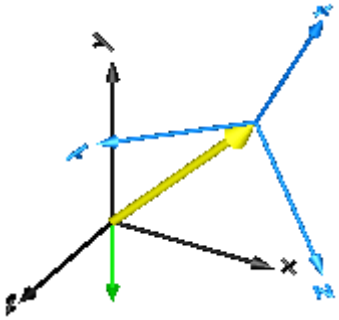
Absolute kinematic quantities of frame_a are computed and provided at the output signal connector **y** in packed format in the order

1. absolute position vector (= r_abs)
2. absolute velocity vectory (= v_abs)
3. absolute acceleration vector (= a_abs)
4. 3 angles to rotate the world frame into frame_a (= angles)
5. absolute angular velocity vector (= w_abs)
6. absolute angular acceleration vector (= z_abs)

For example, if parameters **get_v** and **get_w** are **true** and all other get_XXX parameters are **false**, then y contains 6 elements:

```
y[1:3] = absolute velocity
y[4:6] = absolute angular velocity
```

In the following figure the animation of an AbsoluteSensor component is shown. The light blue coordinate system is frame_a and the yellow arrow is the animated sensor.



If **frame_resolve** is connected to another frame, then the provided absolute kinematic vectors are resolved in this frame. If **frame_resolve** is **not** connected then the coordinate system in which the relative quantities are resolved is defined by parameter **resolveInFrame_a**. If this parameter is **true**, then the provided kinematic vectors are resolved in frame_a of this component. Otherwise, the kinematic vectors are resolved in the world frame. For example, if frame_resolve is not connected and if resolveInFrame_a = **false**, and get_v = **true**, then

```
y = der(frame_a.r) // resolved in world frame
```

is returned, i.e., the derivative of the distance frame_a.r_0 from the origin of the world frame to the origin of frame_a, resolved in the world frame.

Note, the cut-force and the cut-torque in frame_resolve are always zero, whether frame_resolve is connected or not.

If **get_angles** = **true**, the 3 angles to rotate the world frame into frame_a along the axes defined by parameter **sequence** are returned. For example, if sequence = {3,1,2} then the world frame is rotated around angles[1] along the z-axis, afterwards it is rotated around angles[2] along the x-axis, and finally it is rotated around angles[3] along the y-axis and is then identical to frame_a. The 3 angles are returned in the range

```
-π <= angles[i] <= π
```

There are **two solutions** for "angles[1]" in this range. Via parameter **guessAngle1** (default = 0) the returned solution is selected such that |angles[1] - guessAngle1| is minimal. The transformation matrix between the world frame and frame_a may be in a singular configuration with respect to "sequence", i.e., there is an infinite number of angle values leading to the same transformation matrix. In this case, the returned solution is selected by setting angles[1] = guessAngle1. Then angles[2] and angles[3] can be uniquely determined in the above range.

Note, that parameter **sequence** has the restriction that only values 1,2,3 can be used and that sequence[1] ≠ sequence[2] and sequence[2] ≠ sequence[3]. Often used values are:

```
sequence = {1,2,3}  // Cardan angle sequence
         = {3,1,3}  // Euler angle sequence
         = {3,2,1}  // Tait-Bryan angle sequence
```

Exact definition of the returned quantities:

1. r_abs is vector frame_a.r_0, resolved according to table below.
2. v_abs is vector **der**(frame_a.r_0), resolved according to table below.
3. a_abs is vector **der**(**der**(frame_a.r_0)), resolved according to table below.
4. angles is a vector of 3 angles such that frame_a.R = Frames.axesRotations(sequence, angles).
5. w_abs is vector Modelica.Mechanics.MultiBody.Frames.angularVelocity1(frame_a.R, **der**(frame_a.R)), resolved according to table below.
6. z_abs is vector **der**(w_abs) (= derivative of absolute angular velocity of frame_a with respect to the world frame, resolved according to table below).

frame_resolve is	***resolveInFrame_a =***	***vector is resolved in***
connected	true	**frame_resolve**
connected	false	**frame_resolve**
not connected	true	**frame_a**
not connected	false	**world frame**

Parameters

Type	Name	Default	Description
Integer	n_out	3*((if get_r_abs then 1 else...	Number of output signals
Boolean	animation	true	= true, if animation shall be enabled (show arrow)
Boolean	resolveInFrame_a	false	= true, if vectors are resolved in frame_a, otherwise in the world frame (if connector frame_resolve is connected, vectors are resolved in frame_resolve)
Boolean	get_r_abs	true	= true, to measure the position vector from the origin of the world frame to the origin of frame_a in [m]
Boolean	get_v_abs	false	= true, to measure the absolute velocity of the origin of frame_a in [m/s]
Boolean	get_a_abs	false	= true, to measure the absolute acceleration of the origin of frame_a in [m/s^2]
Boolean	get_angles	false	= true, to measure the 3 rotation angles to rotate the world frame into frame_a along the axes defined in 'sequence' below in [rad]
Boolean	get_w_abs	false	= true, to measure the absolute angular velocity of frame_a in [rad/s]
Boolean	get_z_abs	false	= true, to measure the absolute

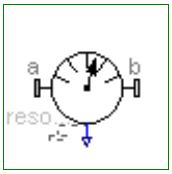
			angular acceleration to frame_a in [rad/s^2]
if get_angles = true			
RotationSequence	sequence	{1,2,3}	Angles are returned to rotate world frame around axes sequence[1], sequence[2] and finally sequence[3] into frame_a
Angle	guessAngle1	0	Select angles[1] such that abs(angles[1] - guessAngle1) is a minimum [rad]
Animation			
if animation = true			
Diameter	arrowDiameter	world.defaultArrowDiameter	Diameter of arrow from world frame to frame_a [m]
Color	arrowColor	Modelica.Mechanics.MultiBody...	Color of arrow from world frame to frame_a
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system from which absolute quantities are provided as output signals
output RealOutput	y[n_out]	Measured data as signal vector
Frame_resolve	frame_resolve	If connected, the output signals are resolved in this frame

---

Modelica.Mechanics.MultiBody.Sensors.RelativeSensor



Measure relative kinematic quantities between two frame connectors

Information

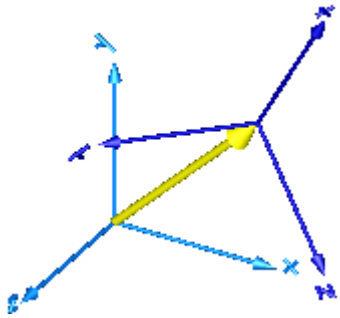
Relative kinematic quantities between frame_a and frame_b are determined and provided at the output signal connector **y** in packed format in the order

1. relative position vector (= r_rel)
2. relative velocity vectory (= v_rel)
3. relative acceleration vector (= a_rel))
4. 3 angles to rotate frame_a into frame_b (= angles)
5. relative angular velocity vector (= w_rel)
6. relative angular acceleration vector (= z_rel)

For example, if parameters **get_v_rel** and **get_w_rel** are **true** and all other get_XXX parameters are **false**, then y contains 6 elements:

```
y = relative velocity
y = relative angular velocity
```

In the following figure the animation of a RelativeSensor component is shown. The light blue coordinate system is frame_a, the dark blue coordinate system is frame_b, and the yellow arrow is the animated sensor.



If parameter **resolveInFrame_a** = **true**, then the provided relative kinematic vectors of frame_b with respect to frame_a are resolved before differentiation in frame_a. If this parameter is **false**, the relative kinematic vectors are resolved before differentiation in frame_b. If **frame_resolve** is connected to another frame, then the kinematic vector as defined above and/or its required derivatives are resolved in frame_resolve. Note, derivatives of relative kinematic quantities are always performed with respect to frame_a (**resolveInFrame_a** = **true**) or with respect to frame_b (**resolveInFrame_a** = **false**). The resulting vector is then resolved in frame_resolve, if this connector is connected.

For example, if frame_resolve is not connected and if resolveInFrame_a = **false**, and get_v = **true**, then

```
y = v_rel
  = der(r_rel)
```

is returned (r_rel = resolve2(frame_b.R, frame_b.r_0 - frame_a.r0)), i.e., the derivative of the relative distance from frame_a to frame_b, resolved in frame_b. If frame_resolve is connected, then

```
y = v_rel
  = resolve2(frame_resolve.R, der(r_rel))
```

is returned, i.e., the previous relative velocity vector is additionally resolved in frame_resolve.

Note, the cut-force and the cut-torque in frame_resolve are always zero, whether frame_resolve is connected or not.

If **get_angles** = **true**, the 3 angles to rotate frame_a into frame_b along the axes defined by parameter **sequence** are returned. For example, if sequence = {3,1,2} then frame_a is rotated around angles[1] along the z-axis, afterwards it is rotated around angles[2] along the x-axis, and finally it is rotated around angles[3] along the y-axis and is then identical to frame_b. The 3 angles are returned in the range

```
-π <= angles[i] <= π
```

There are **two solutions** for "angles[1]" in this range. Via parameter **guessAngle1** (default = 0) the returned solution is selected such that |angles[1] - guessAngle1| is minimal. The relative transformation matrix between frame_a and frame_b may be in a singular configuration with respect to "sequence", i.e., there is an infinite number of angle values leading to the same relative transformation matrix. In this case, the returned solution is selected by setting angles[1] = guessAngle1. Then angles[2] and angles[3] can be uniquely determined in the above range.

Note, that parameter **sequence** has the restriction that only values 1,2,3 can be used and that sequence[1] ≠ sequence[2] and sequence[2] ≠ sequence[3]. Often used values are:

```
sequence = {1,2,3}  // Cardan angle sequence
         = {3,1,3}  // Euler angle sequence
         = {3,2,1}  // Tait-Bryan angle sequence
```

Exact definition of the returned quantities (r_rel_ab, R_rel_ab, w_rel_ab are defined below the enumeration):

1. r_rel is vector r_rel_ab, resolved according to table below.
2. v_rel is vector **der**(r_rel_ab), resolved according to table below.
3. a_rel is vector **der**(**der**(r_rel_ab)), resolved according to table below.

4. angles is a vector of 3 angles such that R_rel_ab = Frames.axesRotations(sequence, angles).
5. w_rel is vector w_rel_ab, resolved according to table below.
6. z_rel is vector **der**(w_rel_ab), resolved according to table below.

using the auxiliary quantities

1. r_rel_ab is vector frame_b.r_0 - frame_a.r_0, resolved either in frame_a or frame_b according to parameter resolveInFrame_a.
2. R_rel_ab is orientation object Frames.relativeRotation(frame_a.R, frame_b.R).
3. w_rel_ab is vector Frames.angularVelocity1(R_rel_ab, der(R_rel_ab)), resolved either in frame_a or frame_b according to parameter resolveInFrame_a.

and resolved in the following frame

frame_resolve is	***resolveInFrame_a =***	***vector is resolved in***
connected	true	**frame_resolve**
connected	false	**frame_resolve**
not connected	true	**frame_a**
not connected	false	**frame_b**

Parameters

Type	Name	Default	Description
Integer	n_out	3*((if get_r_rel then 1 else...	Number of output signals
Boolean	animation	true	= true, if animation shall be enabled (show arrow)
Boolean	resolveInFrame_a	true	= true, if relative vectors from frame_a to frame_b are resolved before differentiation in frame_a, otherwise in frame_b. If frame_resolve is connected, the vector and its derivatives are resolved in frame_resolve
Boolean	get_r_rel	true	= true, to measure the relative position vector from the origin of frame_a to the origin of frame_b in [m]
Boolean	get_v_rel	false	= true, to measure the relative velocity of the origin of frame_b with respect to frame_a in [m/s]
Boolean	get_a_rel	false	= true, to measure the relative acceleration of the origin of frame_b with respect to frame_a in [m/s^2]
Boolean	get_angles	false	= true, to measure the 3 rotation angles to rotate frame_a into frame_b along the axes defined in 'sequence' below in [rad]
Boolean	get_w_rel	false	= true, to measure the relative angular velocity of frame_b with respect to frame_a in [rad/s]
Boolean	get_z_rel	false	= true, to measure the relative angular acceleration of frame_b with respect to frame_a in [rad/s^2]

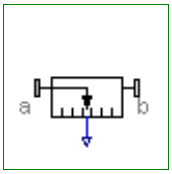
if get_angles = true			
RotationSequence	sequence	{1,2,3}	Angles are returned to rotate frame_a around axes sequence[1], sequence[2] and finally sequence[3] into frame_b
Angle	guessAngle1	0	Select angles[1] such that abs(angles[1] - guessAngle1) is a minimum [rad]
Animation			
if animation = true			
Diameter	arrowDiameter	world.defaultArrowDiameter	Diameter of relative arrow from frame_a to frame_b [m]
Color	arrowColor	Modelica.Mechanics.MultiBody. ..	Color of relative arrow from frame_a to frame_b
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system a
Frame_b	frame_b	Coordinate system b
output RealOutput	y[n_out]	Measured data as signal vector
Frame_resolve	frame_resolve	If connected, the output signals are resolved in this frame

---

Modelica.Mechanics.MultiBody.Sensors.Distance

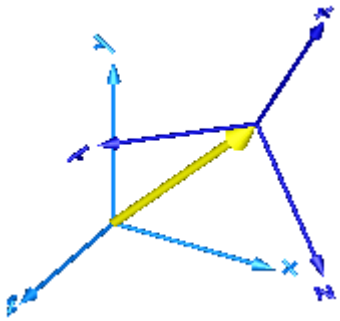


Measure the distance between the origins of two frame connectors

Information

The **distance** between the origins of frame_a and of frame_b are determined and provided at the output signal connector **distance**. This distance is always positive. **Derivatives** of this signal can be easily obtained by connecting the block Modelica.Blocks.Continuous.Der to "distance" (this block performs analytic differentiation of the input signal using the der(..) operator).

In the following figure the animation of a Distance sensor is shown. The light blue coordinate system is frame_a, the dark blue coordinate system is frame_b, and the yellow arrow is the animated sensor.



If the distance is smaller as parameter **s_small** (in the "advanced" menu), it is approximated such that its derivative is finite for zero distance. Without such an approximation, the derivative would be infinite and a division by zero would occur. The approximation is performed in the following way: If distance > s_small, it is computed as sqrt(r*r) where r is the position vector from the origin of frame_a to the origin of frame_b. If the

distance becomes smaller as s_small, the "sqrt()" function is approximated by a second order polynomial, such that the function value and its first derivative are identical for sqrt() and the polynomial at s_small. Futhermore, the polynomial passes through zero. The effect is, that the distance function is continuous and differentiable everywhere. The derivative at zero distance is 3/(2*s_small).

Parameters

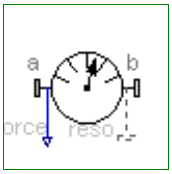
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show arrow)
if animation = true			
Diameter	arrowDiameter	world.defaultArrowDiameter	Diameter of relative arrow from frame_a to frame_b [m]
Color	arrowColor	Modelica.Mechanics.MultiBody...	Color of relative arrow from frame_a to frame_b
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)
Advanced			
Position	s_small	1.E-10	Prevent zero-division if distance between frame_a and frame_b is zero [m]

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
output RealOutput	distance	Distance between the origin of frame_a and the origin of frame_b

---

Modelica.Mechanics.MultiBody.Sensors.CutForce

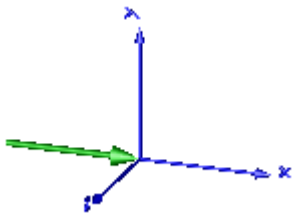


Measure cut force vector

Information

The cut-force acting at the component to which frame_b is connected is determined and provided at the output signal connector **force** (= frame_a.f). If parameter **positiveSign** = **false**, the negative cut-force is provided (= frame_b.f). If **frame_resolve** is connected to another frame, then the cut-force is resolved in frame_resolve. If **frame_resolve** is **not** connected then the coordinate system in which the cut-force is resolved is defined by parameter **resolveInFrame_a**. If this parameter is **true**, then the cut-force is resolved in frame_a, otherwise it is resolved in the world frame.

In the following figure the animation of a CutForce sensor is shown. The dark blue coordinate system is frame_b, and the green arrow is the cut force acting at frame_b and with negative sign at frame_a.



Parameters

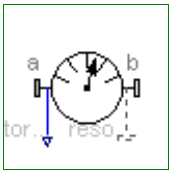
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show arrow)
Boolean	positiveSign	true	= true, if force with positive sign is returned (= frame_a.f), otherwise with negative sign (= frame_b.f)
Boolean	resolveInFrame_a	true	= true, if force is resolved in frame_a/frame_b, otherwise in the world frame (if connector frame_resolve is connected, the force is resolved in frame_resolve)
if animation = true			
Real	N_to_m	1000	Force arrow scaling (length = force/N_to_m) [N/m]
Diameter	forceDiameter	world.defaultArrowDiameter	Diameter of force arrow [m]
Color	forceColor	Modelica.Mechanics.MultiBody...	Color of force arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the output signals are resolved in this frame (cut-force/-torque are set to zero)
output RealOutput	force[3]	Cut force resolved in frame_a/frame_b or in frame_resolved, if connected

---

Modelica.Mechanics.MultiBody.Sensors.CutTorque

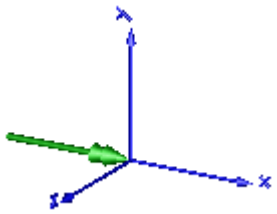


Measure cut torque vector

Information

The cut-torque acting at the component to which frame_b is connected is determined and provided at the output signal connector **torque** (= frame_a.t). If parameter **positiveSign** = **false**, the negative cut-force is provided (= frame_b.t). If **frame_resolve** is connected to another frame, then the cut-torque is resolved in frame_resolve. If **frame_resolve** is **not** connected then the coordinate system in which the cut-torque is resolved is defined by parameter **resolveInFrame_a**. If this parameter is **true**, then the cut-torque is resolved in frame_a, otherwise it is resolved in the world frame.

In the following figure the animation of a CutTorque sensor is shown. The dark blue coordinate system is frame_b, and the green arrow is the cut torque acting at frame_b and with negative sign at frame_a.



Parameters

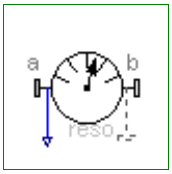
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show arrow)
Boolean	positiveSign	true	= true, if torque with positive sign is returned (= frame_a.t), otherwise with negative sign (= frame_b.t)
Boolean	resolveInFrame_a	true	= true, if torque is resolved in frame_a/frame_b, otherwise in the world frame (if connector frame_resolve is connected, the torque is resolved in frame_resolve)
if animation = true			
Real	Nm_to_m	1000	Torque arrow scaling (length = torque/Nm_to_m) [N.m/m]
Diameter	torqueDiameter	world.defaultArrowDiameter	Diameter of torque arrow [m]
Color	torqueColor	Modelica.Mechanics.MultiBody...	Color of torque arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system with one cut-force and cut-torque
Frame_resolve	frame_resolve	If connected, the output signals are resolved in this frame (cut-force/-torque are set to zero)
output RealOutput	torque[3]	Cut torque resolved in frame_a/frame_b or in frame_resolved, if connected

---

Modelica.Mechanics.MultiBody.Sensors.CutForceAndTorque



Measure cut force and cut torque vector

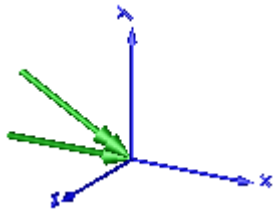
Information

The cut-force and cut-torque acting at the component to which frame_b is connected are determined and provided at the output signal connector **load**:

```
load[1:3] = frame_a.f;
load[4:6] = frame_a.t;
```

If parameter **positiveSign** = **false**, the negative cut-force and negative cut-torque is provided (= frame_b.f and frame_b.t). If **frame_resolve** is connected to another frame, then the cut-force and cut-torque are resolved in frame_resolve. If **frame_resolve** is **not** connected then the coordinate system in which the cut-force and cut-torque is resolved is defined by parameter **resolveInFrame_a**. If this parameter is **true**, then the cut-force and cut-torque is resolved in frame_a, otherwise it is resolved in the world frame.

In the following figure the animation of a CutForceAndTorque sensor is shown. The dark blue coordinate system is frame_b, and the green arrows are the cut force and the cut torque, respectively, acting at frame_b and with negative sign at frame_a.



Parameters

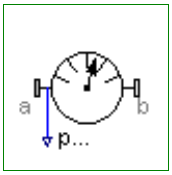
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled (show force and torque arrow)
Boolean	positiveSign	true	= true, if force and torque with positive sign is returned (= frame_a.f/.t), otherwise with negative sign (= frame_b.f/.t)
Boolean	resolveInFrame_a	true	= true, if force and torque are resolved in frame_a/frame_b, otherwise in the world frame (if connector frame_resolve is connected, the force/torque is resolved in frame_resolve)
if animation = true			
Real	N_to_m	1000	Force arrow scaling (length = force/N_to_m) [N/m]
Real	Nm_to_m	1000	Torque arrow scaling (length = torque/Nm_to_m) [N.m/m]
Diameter	forceDiameter	world.defaultArrowDiameter	Diameter of force arrow [m]
Diameter	torqueDiameter	forceDiameter	Diameter of torque arrow [m]
Color	forceColor	Modelica.Mechanics.MultiBody...	Color of force arrow
Color	torqueColor	Modelica.Mechanics.MultiBody...	Color of torque arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system with one cut-force and cut-torque
Frame_resolve	frame_resolv	If connected, the output signals are resolved in this frame (cut-force/-torque

	e	are set to zero)
output RealOutput	load[6]	Cut force and cut torque resolved in frame_a/frame_b or in frame_resolved, if connected

Modelica.Mechanics.MultiBody.Sensors.Power



Measure power flowing from frame_a to frame_b

Information

This component provides the power flowing from frame_a to frame_b as output signal **power**.

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system fixed to the component with one cut-force and cut-torque
Frame_b	frame_b	Coordinate system fixed to the component with one cut-force and cut-torque
output RealOutput	power	Power at frame_a as output signal

Modelica.Mechanics.MultiBody.Types

Constants and types with choices, especially to build menus

Information

In this package **types** and **constants** are defined that are used in the MultiBody library. The types have additional annotation choices definitions that define the menus to be built up in the graphical user interface when the type is used as parameter in a declaration.

Package Content

Name	Description
R Axis	Axis vector with choices for menus
S AxisLabel	Label of axis with choices for menus
I RotationSequence	Sequence of planar frame rotations with choices for menus
I Color	RGB representation of color (will be improved with a color editor)
R SpecularCoefficient	Reflection of ambient light (= 0: light is completely absorbed)
S ShapeType	Type of shape (box, sphere, cylinder, pipecylinder, cone, pipe, beam, gearwheel, spring, dxf-file)
R ShapeExtra	Reflection of ambient light (= 0: light is completely absorbed)
R AngularVelocity_degs	Angular velocity type in deg/s
R AngularAcceleration_degs2	Angular acceleration type in deg/s^2
e RotationTypes	Type, constants and menu choices for rotation types, as temporary solution until enumerations are available
e GravityTypes	Type, constants and menu choices for gravity fields, as temporary solution until enumerations are available

Init	Type, constants and menu choices to define initialization, as temporary solution until enumerations are available
Defaults	Default settings of the MultiBody library via constants

Types and constants

```
  type Axis = Modelica.Icons.TypeReal[3] "Axis vector with choices for menus";

  type AxisLabel = Modelica.Icons.TypeString
  "Label of axis with choices for menus";

  type RotationSequence = Modelica.Icons.TypeInteger[3] (min={1,1,1}, max={3,3,3})
  "Sequence of planar frame rotations with choices for menus";

  type Color = Modelica.Icons.TypeInteger[3] (each min=0, each max=255)
  "RGB representation of color (will be improved with a color editor)";

  type SpecularCoefficient = Modelica.Icons.TypeReal
  "Reflection of ambient light (= 0: light is completely absorbed)";

  type ShapeType = Modelica.Icons.TypeString
  "Type of shape (box, sphere, cylinder, pipecylinder, cone, pipe, beam,
gearwheel, spring, dxf-file)";

  type ShapeExtra = Modelica.Icons.TypeReal
  "Reflection of ambient light (= 0: light is completely absorbed)";

  type AngularVelocity_degs = Modelica.Icons.TypeReal(final
quantity="AngularVelocity", final unit
      =      "deg/s") "Angular velocity type in deg/s";

  type AngularAcceleration_degs2 = Modelica.Icons.TypeReal (final
quantity="AngularAcceleration",
        final unit="deg/s2") "Angular acceleration type in deg/s^2";
```

Modelica.Mechanics.MultiBody.Types.RotationTypes

Type, constants and menu choices for rotation types, as temporary solution until enumerations are available

Information

Package Content

Name	Description
RotationAxis=1	
TwoAxesVectors=2	
PlanarRotationSequence=3	
Temp	Temporary type of RotationTypes with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer RotationAxis=1;

  constant Integer TwoAxesVectors=2;

  constant Integer PlanarRotationSequence=3;

  type Temp
  "Temporary type of RotationTypes with choices for menus (until enumerations
are available)"

    extends Modelica.Icons.TypeInteger;

  end Temp;
```

Modelica.Mechanics.MultiBody.Types.GravityTypes

Type, constants and menu choices for gravity fields, as temporary solution until enumerations are available

Information

Package Content

Name	Description
NoGravity=0	
UniformGravity=1	
PointGravity=2	
Temp	Temporary type of gravity field with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer NoGravity=0;

  constant Integer UniformGravity=1;

  constant Integer PointGravity=2;

  type Temp
  "Temporary type of gravity field with choices for menus (until enumerations
are available)"

    extends Modelica.Icons.TypeInteger;

  end Temp;
```

Modelica.Mechanics.MultiBody.Types.Init

Type, constants and menu choices to define initialization, as temporary solution until enumerations

are available

Information

Package Content

Name	Description
Free=1	
PositionVelocity=2	
SteadyState=3	
Position=4	
Velocity=5	
VelocityAcceleration=6	
PositionVelocityAcceleration=7	
Temp	Temporary type of Init with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer Free=1;

  constant Integer PositionVelocity=2;

  constant Integer SteadyState=3;

  constant Integer Position=4;

  constant Integer Velocity=5;

  constant Integer VelocityAcceleration=6;

  constant Integer PositionVelocityAcceleration=7;

  type Temp
  "Temporary type of Init with choices for menus (until enumerations are
available)"

    extends Modelica.Icons.TypeInteger;

  end Temp;
```

Modelica.Mechanics.MultiBody.Types.Defaults

Default settings of the MultiBody library via constants

Information

This package contains constants used as default setting in the MultiBody library.

Package Content

Name	Description
BodyColor={0,128,255}	Default color for body shapes that have mass (light blue)
RodColor={155,155,155}	Default color for massless rod shapes (grey)
JointColor={255,0,0}	Default color for elementary joints (red)
ForceColor={0,128,0}	Default color for force arrow (dark green)
TorqueColor={0,128,0}	Default color for torque arrow (dark green)
SpringColor={0,0,255}	Default color for a spring (blue)
SensorColor={255,255,0}	Default color for sensors (yellow)
FrameColor={0,0,0}	Default color for frame axes and labels (black)
ArrowColor={0,0,255}	Default color for arrows and double arrows (blue)
FrameHeadLengthFraction=5.0	Frame arrow head length / arrow diameter
FrameHeadWidthFraction=3.0	Frame arrow head width / arrow diameter
FrameLabelHeightFraction=3.0	Height of frame label / arrow diameter
ArrowHeadLengthFraction=4.0	Arrow head length / arrow diameter
ArrowHeadWidthFraction=3.0	Arrow head width / arrow diameter
BodyCylinderDiameterFraction=3	Default for body cylinder diameter as a fraction of body sphere diameter
JointRodDiameterFraction=2	Default for rod diameter as a fraction of joint sphere diameter attached to rod

Types and constants

```
constant Types.Color BodyColor={0,128,255}
"Default color for body shapes that have mass (light blue)";

constant Types.Color RodColor={155,155,155}
"Default color for massless rod shapes (grey)";

constant Types.Color JointColor={255,0,0}
"Default color for elementary joints (red)";

constant Types.Color ForceColor={0,128,0}
"Default color for force arrow (dark green)";

constant Types.Color TorqueColor={0,128,0}
"Default color for torque arrow (dark green)";

constant Types.Color SpringColor={0,0,255}
"Default color for a spring (blue)";

constant Types.Color SensorColor={255,255,0}
"Default color for sensors (yellow)";

constant Types.Color FrameColor={0,0,0}
"Default color for frame axes and labels (black)";

constant Types.Color ArrowColor={0,0,255}
"Default color for arrows and double arrows (blue)";

constant Real FrameHeadLengthFraction=5.0
```

```
  "Frame arrow head length / arrow diameter";

  constant Real FrameHeadWidthFraction=3.0
  "Frame arrow head width / arrow diameter";

  constant Real FrameLabelHeightFraction=3.0
  "Height of frame label / arrow diameter";

  constant Real ArrowHeadLengthFraction=4.0
  "Arrow head length / arrow diameter";

  constant Real ArrowHeadWidthFraction=3.0 "Arrow head width / arrow diameter";

  constant SI.Diameter BodyCylinderDiameterFraction=3
  "Default for body cylinder diameter as a fraction of body sphere diameter";

  constant Real JointRodDiameterFraction=2
  "Default for rod diameter as a fraction of joint sphere diameter attached to
rod";
```

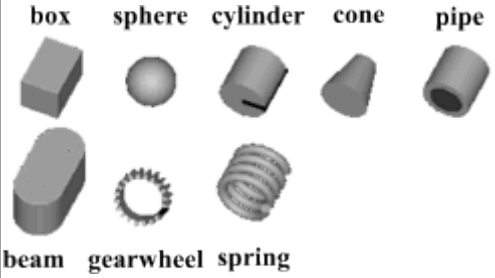
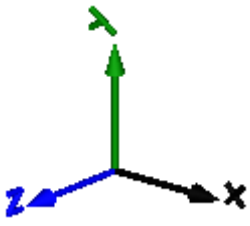
Modelica.Mechanics.MultiBody.Visualizers

3-dimensional visual objects used for animation

Information

Package **Visualizers** contains components to visualize 3-dimensional shapes. These components are the basis for the animation features of the MultiBody library.

Content

FixedShape FixedShape2	Animation shape of a part with fixed sizes. FixedShape2 has additionally a frame_b for easier connection to further visual objects. The following shape types are supported: 
FixedFrame	Visualizing a coordinate system including axes labels with fixed sizes: 
FixedArrow,	Visualizing an arrow. Model "FixedArrow" provides a fixed sized arrow, model "SignalArrow"

SignalArrow	provides an arrow with dynamically varying length that is defined by an input signal vector:
Advanced	**Package** that contains components to visualize 3-dimensional shapes where all parts of the shape can vary dynamically. Basic knowledge of Modelica is needed in order to utilize the components of this package.

The colors of the visualization components are declared with the predefined type **MultiBody.Types.Color**. This is a vector with 3 elements, {r, g, b}, and specifies the color of the shape. {r,g,b} are the "red", "green" and "blue" color parts. Note, r g, b are given as Integer[3] in the ranges 0 .. 255, respectively.

Package Content

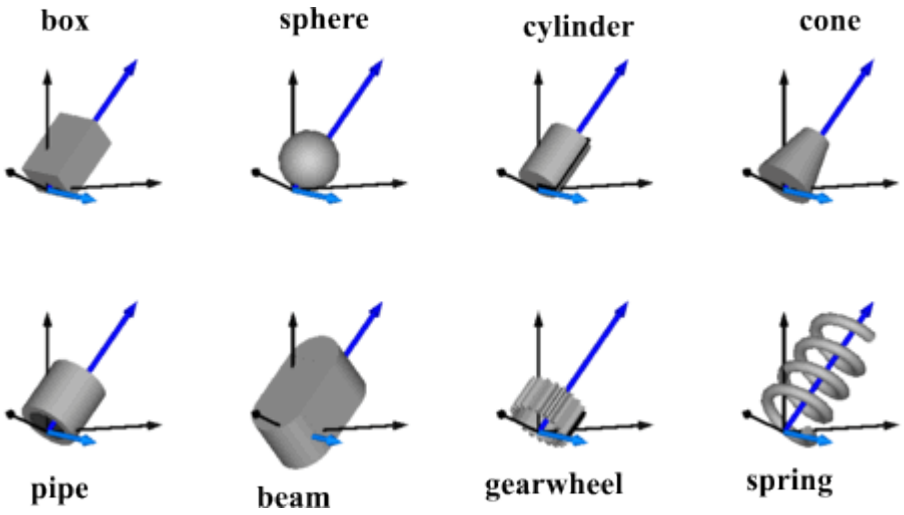
Name	Description
FixedShape	Animation shape of a part with fixed shape type and dynamically varying shape definition
FixedShape2	Animation shape of a part with fixed shape type and dynamically varying shape definition with two frames
FixedFrame	Visualizing a coordinate system including axes labels (visualization data may vary dynamically)
FixedArrow	Visualizing an arrow with dynamically varying size in frame_a
SignalArrow	Visualizing an arrow with dynamically varying size in frame_a based on input signal
Advanced	Visualizers that require basic knowledge about Modelica in order to use them

Modelica.Mechanics.MultiBody.Visualizers.FixedShape

Animation shape of a part with fixed shape type and dynamically varying shape definition

Information

Model **FixedShape** defines a visual shape that is shown at the location of its frame_a. All describing data such as size and color can vary dynamically by providing appropriate expressions in the input fields of the parameter menu. The only exception is parameter shapeType that cannot be changed during simulation. The following shapes are currently supported via parameter **shapeType** (e.g., shapeType="box"):



The dark blue arrows in the figure above are directed along variable **lengthDirection**. The light blue arrows are directed along variable **widthDirection**. The **coordinate systems** in the figure represent frame_a of the FixedShape component.

Additionally external shapes are specified as DXF-files (only 3-dim.Face is supported). External shapes must be named "1", "2" etc.. The corresponding definitions should be in files "1.dxf", "2.dxf" etc.Since the DXF-files contain color and dimensions for the individual faces, the corresponding information in the model is currently ignored. The DXF-files must be found in the current directory.

The sizes of any of the above components are specified by the **length**, **width** and **height** variables. Via variable **extra** additional data can be defined:

shapeType	Meaning of parameter extra
"cylinder"	if extra > 0, a black line is included in the cylinder to show the rotation of it.
"cone"	extra = diameter-left-side / diameter-right-side, i.e., extra = 1: cylinder extra = 0: "real" cone.
"pipe"	extra = outer-diameter / inner-diameter, i.e, extra = 1: cylinder that is completely hollow extra = 0: cylinder without a hole.
"gearwheel"	extra is the number of teeth of the gear.
"spring"	extra is the number of windings of the spring. Additionally, "height" is **not** the "height" but 2*coil-width.

Parameter **color** is a vector with 3 elements, {r, g, b}, and specifies the color of the shape. {r,g,b} are the "red", "green" and "blue" color parts. Note, r g, b are given as Integer[3] in the ranges 0 .. 255, respectively. The predefined type **MultiBody.Types.Color** contains a temporary menu definition of the colors used in the MultiBody library (this will be replaced by a color editor).

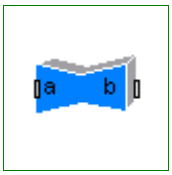
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
ShapeType	shapeType	"box"	Type of shape
Position	r_shape[3]	{0,0,0}	Vector from frame_a to shape origin, resolved in frame_a [m]
Axis	lengthDirection	{1,0,0}	Vector in length direction of shape, resolved in frame_a
Axis	widthDirection	{0,1,0}	Vector in width direction of shape, resolved in frame_a
Distance	length	1	Length of shape [m]
Distance	width	0.1	Width of shape [m]
Distance	height	0.1	Height of shape [m]
Color	color	{0,128,255}	Color of shape
ShapeExtra	extra	0.0	Additional data for cylinder, cone, pipe, gearwheel and spring
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic..	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system in which visualization data is resolved

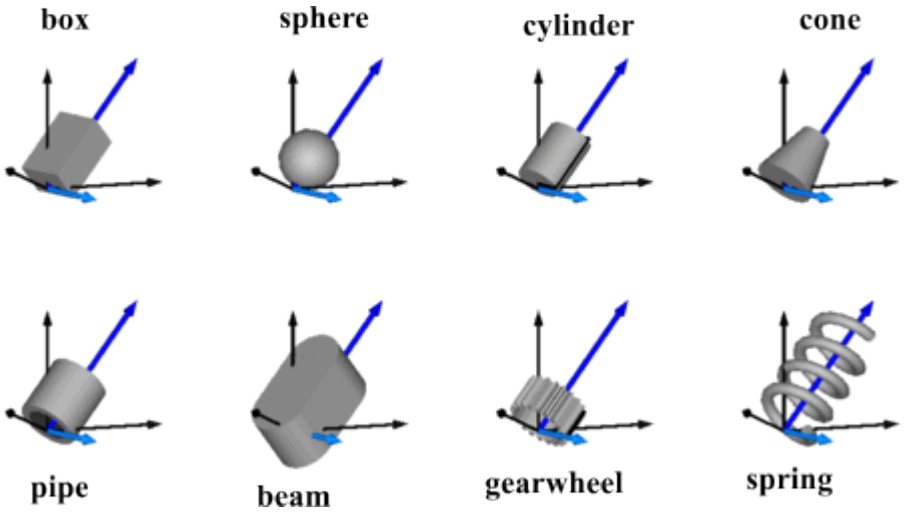
Modelica.Mechanics.MultiBody.Visualizers.FixedShape2



Animation shape of a part with fixed shape type and dynamically varying shape definition with two frames

Information

Model **FixedShape2** defines a visual shape that is shown at the location of its frame_a. This model is identical to **FixedShape** with the only difference that an additional frame_b is present which is parallel to frame_a. This makes it more convenient to connect several visual shapes together when building up more complex visual objects. All describing data such as size and color can vary dynamically by providing appropriate expressions in the input fields of the parameter menu. The only exception is parameter shapeType that cannot be changed during simulation. The following shapes are currently supported via parameter **shapeType** (e.g., shapeType="box"):



The dark blue arrows in the figure above are directed along variable **lengthDirection**. The light blue arrows are directed along variable **widthDirection**. The **coordinate systems** in the figure represent frame_a of the FixedShape component.

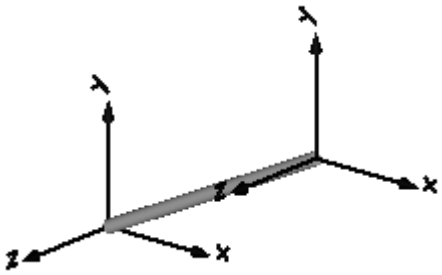
Additionally external shapes are specified as DXF-files (only 3-dim.Face is supported). External shapes must be named "1", "2" etc.. The corresponding definitions should be in files "1.dxf", "2.dxf" etc.Since the DXF-files contain color and dimensions for the individual faces, the corresponding information in the model is currently ignored. The DXF-files must be found in the current directory.

The sizes of any of the above components are specified by the **length**, **width** and **height** variables. Via variable **extra** additional data can be defined:

shapeType	Meaning of parameter extra
"cylinder"	if extra > 0, a black line is included in the cylinder to show the rotation of it.
"cone"	extra = diameter-left-side / diameter-right-side, i.e., extra = 1: cylinder extra = 0: "real" cone.
"pipe"	extra = outer-diameter / inner-diameter, i.e, extra = 1: cylinder that is completely hollow extra = 0: cylinder without a hole.
"gearwheel"	extra is the number of teeth of the gear.
"spring"	extra is the number of windings of the spring. Additionally, "height" is **not** the "height" but 2*coil-width.

Parameter **color** is a vector with 3 elements, {r, g, b}, and specifies the color of the shape. {r,g,b} are the "red", "green" and "blue" color parts. Note, r g, b are given as Integer[3] in the ranges 0 .. 255, respectively. The predefined type **MultiBody.Types.Color** contains a temporary menu definition of the colors used in the MultiBody library (this will be replaced by a color editor).

In the following figure the relationships between frame_a and frame_b are shown. The origin of frame_b with respect to frame_a is specified via parameter vector **r**.



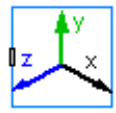
Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
ShapeType	shapeType	"box"	Type of shape
Position	r_shape[3]	{0,0,0}	Vector from frame_a to shape origin, resolved in frame_a [m]
Axis	lengthDirection	r - r_shape	Vector in length direction of shape, resolved in frame_a
Axis	widthDirection	{0,1,0}	Vector in width direction of shape, resolved in frame_a
Length	length	Frames.length(r - r_shape)	Length of shape [m]
Distance	width	0.1	Width of shape [m]
Distance	height	width	Height of shape [m]
ShapeExtra	extra	0.0	Additional data for cylinder, cone, pipe, gearwheel and spring
Color	color	{0,128,255}	Color of shape
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic..	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system a (all shape definition vectors are resolved in this frame)
Frame_b	frame_b	Coordinate system b

Modelica.Mechanics.MultiBody.Visualizers.FixedFrame

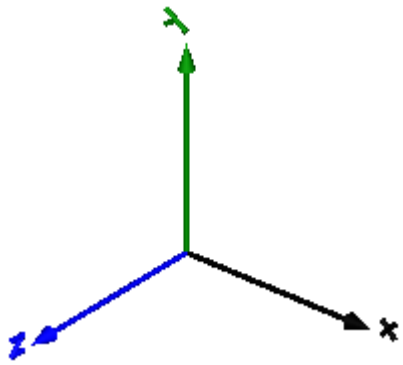


Visualizing a coordinate system including axes labels (visualization data may vary dynamically)

Information

Model **FixedFrame** visualizes the three axes of its coordinate system **frame_a** together with appropriate

axes labels. A typical example is shown in the following figure:



The sizes of the axes, the axes colors and the specular coefficient (= reflection factor for ambient light) can vary dynamically by providing appropriate expressions in the input fields of the parameter menu.

Parameters

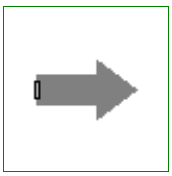
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Boolean	showLabels	true	= true, if labels shall be shown
Distance	length	0.5	Length of axes arrows [m]
Distance	diameter	length/world.defaultFrameDia...	Diameter of axes arrows [m]
Color	color_x	Modelica.Mechanics.MultiBody.. .	Color of x-arrow
Color	color_y	color_x	Color of y-arrow
Color	color_z	color_x	Color of z-arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system in which visualization data is resolved

---

Modelica.Mechanics.MultiBody.Visualizers.FixedArrow



Visualizing an arrow with dynamically varying size in frame_a

Information

Model **FixedArrow** defines an arrow that is shown at the location of its frame_a.

The direction of the arrow specified with vector **n** is with respect to frame_a, i.e., the local frame to which the

arrow component is attached. The direction and length of the arrow, its diameter and color can vary dynamically by providing appropriate expressions in the input fields of the parameter menu.

Parameters

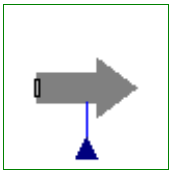
Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Position	r_tail[3]	{0,0,0}	Vector from frame_a to arrow tail, resolved in frame_a [m]
Axis	n	{1,0,0}	Vector in arrow direction, resolved in frame_a
Length	length	0.1	Length of complete arrow [m]
Diameter	diameter	world.defaultArrowDiameter	Diameter of arrow line [m]
Color	color	{0,0,255}	Color of arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic..	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system in which visualization data is resolved

---

Modelica.Mechanics.MultiBody.Visualizers.SignalArrow



Visualizing an arrow with dynamically varying size in frame_a based on input signal

Information

Model **SignalArrow** defines an arrow that is dynamically visualized at the location where its frame_a is attached. The position vector from the tail to the head of the arrow, resolved in frame_a, is defined via the signal vector of the connector **r_head** (Real r_head[3]):

The tail of the arrow is defined with parameter **r_tail** with respect to frame_a (vector from the origin of frame_a to the arrow tail).

Parameters

Type	Name	Default	Description
Boolean	animation	true	= true, if animation shall be enabled
if animation = true			
Position	r_tail[3]	{0,0,0}	Vector from frame_a to arrow tail, resolved in frame_a [m]
Diameter	diameter	world.defaultArrowDiameter	Diameter of arrow line [m]
Color	color	{0,0,255}	Color of arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic..	Reflection of ambient light (= 0: light is completely absorbed)

Connectors

Type	Name	Description
Frame_a	frame_a	Coordinate system in which visualization data is resolved
input RealInput	r_head[3]	Position vector from origin of frame_a to head of arrow, resolved in frame_a

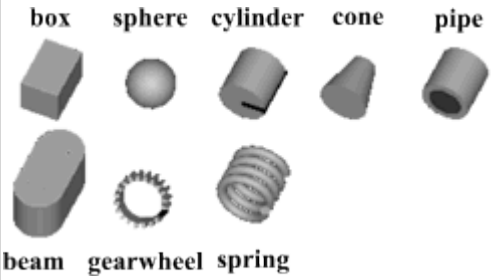
Modelica.Mechanics.MultiBody.Visualizers.Advanced

Visualizers that require basic knowledge about Modelica in order to use them

Information

Package **Visualizers.Advanced** contains components to visualize 3-dimensional shapes with dynamical sizes. None of the components has a frame connector. The position and orientation is set via modifiers. Basic knowledge of Modelica is needed in order to utilize the components of this package. These components have also to be used for models, where the forces and torques in the frame connector are set via equations (in this case, the models of the Visualizers package cannot be used, since they all have frame connectors).

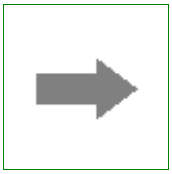
Content

Arrow	Visualizing an arrow where all parts of the arrow can vary dynamically:
DoubleArrow	Visualizing a double arrow where all parts of the arrow can vary dynamically:
Shape	Animation shape of a part with dynamically varying sizes. The following shape types are supported: 

Package Content

Name	Description
Arrow	Visualizing an arrow with variable size; all data have to be set as modifiers (see info layer)
DoubleArrow	Visualizing a double arrow with variable size; all data have to be set as modifiers (see info layer)
Shape	Different visual shapes with variable size; all data have to be set as modifiers (see info layer)

Modelica.Mechanics.MultiBody.Visualizers.Advanced.Arrow



Visualizing an arrow with variable size; all data have to be set as modifiers (see info layer)

Information

Model **Arrow** defines an arrow that is dynamically visualized at the defined location (see variables below).

The variables under heading **Parameters** below are declared as (time varying) **input** variables. If the default equation is not appropriate, a corresponding modifier equation has to be provided in the model where an **Arrow** instance is used, e.g., in the form

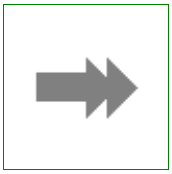
```
Visualizers.Advanced.Arrow arrow(diameter = sin(time));
```

Variable **color** is an Integer vector with 3 elements, {r, g, b}, and specifies the color of the shape. {r,g,b} are the "red", "green" and "blue" color parts. Note, r g, b are given in the range 0 .. 255. The predefined type **MultiBody.Types.Color** contains a menu definition of the colors used in the MultiBody library (will be replaced by a color editor).

Parameters

Type	Name	Default	Description
Orientation	R	Frames.nullRotation()	Orientation object to rotate the world frame into the arrow frame.
Position	r[3]	{0,0,0}	Position vector from origin of world frame to origin of arrow frame, resolved in world frame [m]
Position	r_tail[3]	{0,0,0}	Position vector from origin of arrow frame to arrow tail, resolved in arrow frame [m]
Position	r_head[3]	{0,0,0}	Position vector from arrow tail to the head of the arrow, resolved in arrow frame [m]
Diameter	diameter	world.defaultArrowDiameter	Diameter of arrow line [m]
Color	color	Modelica.Mechanics.MultiBody...	Color of arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Material property describing the reflecting of ambient light (= 0 means, that light is completely absorbed)

Modelica.Mechanics.MultiBody.Visualizers.Advanced.DoubleArrow



Visualizing a double arrow with variable size; all data have to be set as modifiers (see info layer)

Information

Model **DoubleArrow** defines a double arrow that is dynamically visualized at the defined location (see

variables below).

The variables under heading **Parameters** below are declared as (time varying) **input** variables. If the default equation is not appropriate, a corresponding modifier equation has to be provided in the model where an **Arrow** instance is used, e.g., in the form

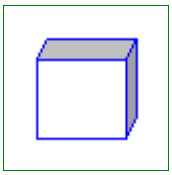
```
Visualizers.Advanced.DoubleArrow doubleArrow(diameter = sin(time));
```

Variable **color** is an Integer vector with 3 elements, {r, g, b}, and specifies the color of the shape. {r,g,b} are the "red", "green" and "blue" color parts. Note, r g, b are given in the range 0 .. 255. The predefined type **MultiBody.Types.Color** contains a menu definition of the colors used in the MultiBody library (will be replaced by a color editor).

Parameters

Type	Name	Default	Description
Orientation	R	Frames.nullRotation()	Orientation object to rotate the world frame into the arrow frame.
Position	r[3]	{0,0,0}	Position vector from origin of world frame to origin of arrow frame, resolved in world frame [m]
Position	r_tail[3]	{0,0,0}	Position vector from origin of arrow frame to double arrow tail, resolved in arrow frame [m]
Position	r_head[3]	{0,0,0}	Position vector from double arrow tail to the head of the double arrow, resolved in arrow frame [m]
Diameter	diameter	world.defaultArrowDiameter	Diameter of arrow line [m]
Color	color	Modelica.Mechanics.MultiBody...	Color of double arrow
SpecularCoefficient	specularCoefficient	world.defaultSpecularCoeffic...	Material property describing the reflecting of ambient light (= 0 means, that light is completely absorbed)

Modelica.Mechanics.MultiBody.Visualizers.Advanced.Shape

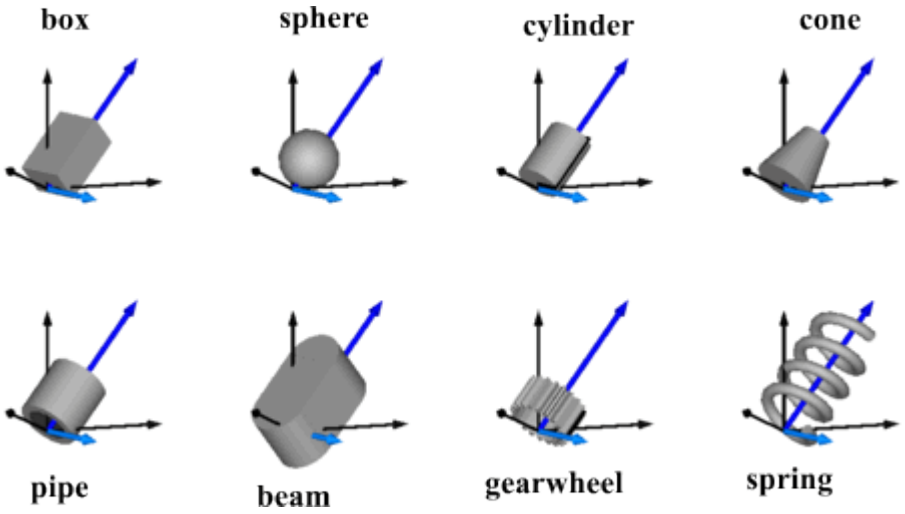


Different visual shapes with variable size; all data have to be set as modifiers (see info layer)

Information

Model **Shape** defines a visual shape that is shown at the location of its reference coordinate system, called 'object frame' below. All describing variables such as size and color can vary dynamically (with the only exception of parameter shapeType). The default equations in the declarations should be modified by providing appropriate equations. Model **Shape** is usually used as a basic building block to implement simpler to use graphical components.

The following shapes are supported via parameter **shapeType** (e.g., shapeType="box"):



The dark blue arrows in the figure above are directed along variable **lengthDirection**. The light blue arrows are directed along variable **widthDirection**. The **coordinate systems** in the figure represent frame_a of the Shape component.

Additionally, external shapes are specified as DXF-files (only 3-dim.Face is supported). External shapes must be named "1", "2" etc.. The corresponding definitions should be in files "1.dxf", "2.dxf" etc.Since the DXF-files contain color and dimensions for the individual faces, the corresponding information in the model is currently ignored. The DXF-files must be found either in the current directory or in the directory where the Shape instance is stored that references the DXF file.

Via input variable **extra** additional sizing data is defined according to:

shapeType	Meaning of variable extra
"cylinder"	if extra > 0, a black line is included in the cylinder to show the rotation of it.
"cone"	extra = diameter-left-side / diameter-right-side, i.e., extra = 1: cylinder extra = 0: "real" cone.
"pipe"	extra = outer-diameter / inner-diameter, i.e, extra = 1: cylinder that is completely hollow extra = 0: cylinder without a hole.
"gearwheel"	extra is the number of teeth of the gear.
"spring"	extra is the number of windings of the spring. Additionally, "height" is **not** the "height" but 2*coil-width.

Parameter **color** is an Integer vector with 3 elements, {r, g, b}, and specifies the color of the shape. {r,g,b} are the "red", "green" and "blue" color parts. Note, r g, b are given in the range 0 .. 255. The predefined type **MultiBody.Types.Color** contains a menu definition of the colors used in the MultiBody library (will be replaced by a color editor).

The variables under heading **Parameters** below are declared as (time varying) **input** variables. If the default equation is not appropriate, a corresponding modifier equation has to be provided in the model where a **Shape** instance is used, e.g., in the form

```
Visualizers.Advanced.Shape shape(length = sin(time));
```

Parameters

Type	Name	Default	Description
ShapeType	shapeType	"box"	Type of shape (box, sphere, cylinder, pipecylinder, cone, pipe, beam, gearwheel,

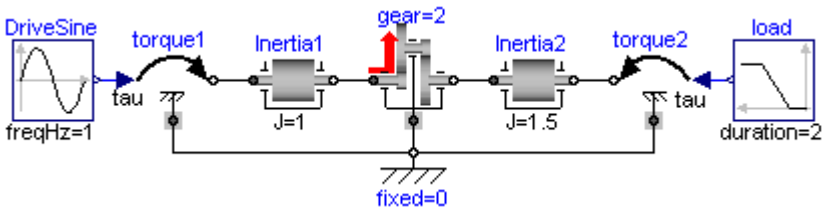
			spring)
Orientation	R	Frames.nullRotation( )	Orientation object to rotate the world frame into the object frame
Position	r[3]	{0,0,0}	Position vector from origin of world frame to origin of object frame, resolved in world frame [m]
Position	r_shape[3]	{0,0,0}	Position vector from origin of object frame to shape origin, resolved in object frame [m]
Real	lengthDirection[3]	{1,0,0}	Vector in length direction, resolved in object frame
Real	widthDirection[3]	{0,1,0}	Vector in width direction, resolved in object frame
Length	length	0	Length of visual object [m]
Length	width	0	Width of visual object [m]
Length	height	0	Height of visual object [m]
ShapeExtra	extra	0.0	Additional size data for some of the shape types
Real	color[3]	{255,0,0}	Color of shape
SpecularCoefficient	specularCoefficient	0.7	Reflection of ambient light (= 0: light is completely absorbed)

Modelica.Mechanics.Rotational

Library to model 1-dimensional, rotational mechanical systems

Information

Library **Rotational** is a **free** Modelica package providing 1-dimensional, rotational mechanical components to model in a convenient way drive trains with frictional losses. A typical, simple example is shown in the next figure:



For an introduction, have especially a look at:

- Rotational.UsersGuide discusses the most important aspects how to use this library.
- Rotational.Examples contains examples that demonstrate the usage of this library.

Copyright © 1998-2007, Modelica Association and DLR.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
UsersGuide	User's Guide of Rotational Library

Examples	Demonstration examples of the components of this package
Sensors	Sensors to measure variables in 1D rotational mechanical components
Interfaces	Connectors and partial models for 1D rotational mechanical components
Inertia	1D-rotational component with inertia
IdealGear	Ideal gear without inertia
IdealPlanetary	Ideal planetary gear box
IdealGearR2T	Gearbox transforming rotational into translational motion
Spring	Linear 1D rotational spring
Damper	Linear 1D rotational damper
SpringDamper	Linear 1D rotational spring and damper in parallel
ElastoBacklash	Backlash connected in series to linear spring and damper (backlash is modeled with elasticity)
BearingFriction	Coulomb friction in bearings
Clutch	Clutch based on Coulomb friction
OneWayClutch	Series connection of freewheel and clutch
Brake	Brake based on Coulomb friction
LossyGear	Gear with mesh efficiency and bearing friction (stuck/rolling possible)
GearEfficiency	Obsolete component (use model LossyGear instead)
Gear	Realistic model of a gearbox
Gear2	Realistic model of a gearbox (based on LossyGear)
Position	Forced movement of a flange according to a reference angle signal
Speed	Forced movement of a flange according to a reference angular velocity signal
Accelerate	Forced movement of a flange according to an acceleration signal
Move	Forced movement of a flange according to an angle, speed and angular acceleration signal
Fixed	Flange fixed in housing at a given angle
Torque	Input signal acting as external torque on a flange
Torque2	Input signal acting as torque on two flanges
LinearSpeedDependentTorque	Linear dependency of torque versus speed
QuadraticSpeedDependentTorque	Quadratic dependency of torque versus speed
ConstantTorque	Constant torque, not dependent on speed
ConstantSpeed	Constant speed, not dependent on torque
TorqueStep	Constant torque, not dependent on speed
RelativeStates	Definition of relative state variables
InitializeFlange	Initializes a flange with pre-defined angle, speed and angular acceleration (usually, this is reference data from a control bus)
Types	Constants and types with choices, especially to build menus

Modelica.Mechanics.Rotational.UsersGuide

User's Guide of Rotational Library

Library **Rotational** is a **free** Modelica package providing 1-dimensional, rotational mechanical components to model in a convenient way drive trains with frictional losses.

Package Content

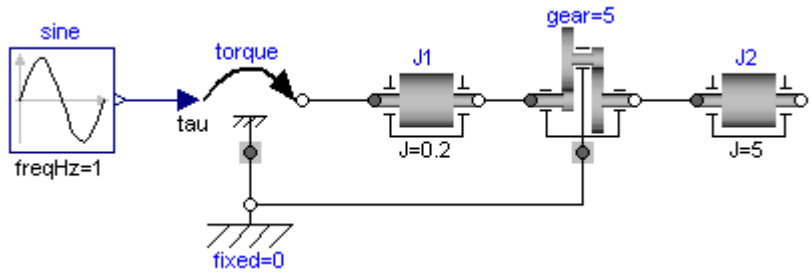
Name	Description
Overview	Overview
FlangeConnectors	Flange Connectors
SupportTorques	Support Torques
SignConventions	Sign Conventions
UserDefinedComponents	User Defined Components
RequirementsForSimulationTool	Requirements for Simulation Tools
ReleaseNotes	Release notes
Contact	Contact

Modelica.Mechanics.Rotational.UsersGuide.Overview

Overview

This package contains components to model **1-dimensional rotational mechanical** systems, including different types of gearboxes, shafts with inertia, external torques, spring/damper elements, frictional elements, backlash, elements to measure angle, angular velocity, angular acceleration and the cut-torque of a flange. In sublibrary **Examples** several examples are present to demonstrate the usage of the elements. Just open the corresponding example model and simulate the model according to the provided description.

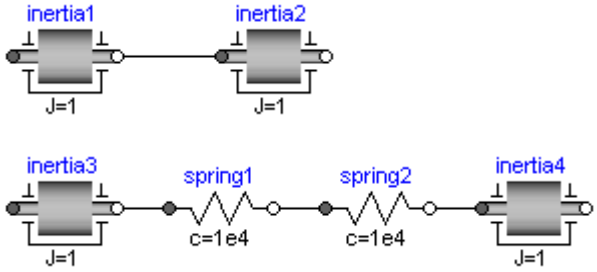
A unique feature of this library is the **component-oriented** modeling of **Coulomb friction** elements, such as friction in bearings, clutches, brakes, and gear efficiency. Even (dynamically) coupled friction elements, e.g., as in automatic gearboxes, can be handeled **without** introducing stiffness which leads to fast simulations. The underlying theory is new and is based on the solution of mixed continuous/discrete systems of equations, i.e., equations where the **unknowns** are of type **Real**, **Integer** or **Boolean**. Provided appropriate numerical algorithms for the solution of such types of systems are available in the simulation tool, the simulation of (dynamically) coupled friction elements of this library is **efficient** and **reliable**.



A simple example of the usage of this library is given in the figure above. This drive consists of a shaft with inertia J1=0.2 which is connected via an ideal gearbox with gear ratio=5 to a second shaft with inertia J2=5. The left shaft is driven via an external, sinusoidal torque. The **filled** and **non-filled grey squares** at the left and right side of a component represent **mechanical flanges**. Drawing a line between such squares means

that the corresponding flanges are **rigidly attached** to each other. By convention in this library, the connector characterized as a **filled** grey square is called **flange_a** and placed at the left side of the component in the "design view" and the connector characterized as a **non-filled** grey square is called **flange_b** and placed at the right side of the component in the "design view". The two connectors are completely **identical**, with the only exception that the graphical layout is a little bit different in order to distinguish them for easier access of the connector variables. For example, `J1.flange_a.tau` is the cut-torque in the connector `flange_a` of component `J1`.

The components of this library can be **connected** together in an **arbitrary** way. E.g., it is possible to connect two springs or two shafts with inertia directly together, see figure below.



Modelica.Mechanics.Rotational.UsersGuide.FlangeConnectors

Flange Connectors

A flange is described by the connector class Interfaces.**Flange_a** or Interfaces.**Flange_b**. As already noted, the two connector classes are completely identical. There is only a difference in the icons, in order to easier identify a flange variable in a diagram. Both connector classes contain the following variables:

```
SIunits.Angle       phi  "absolute rotation angle of flange";
flow SIunits.Torque tau  "cut-torque in the flange";
```

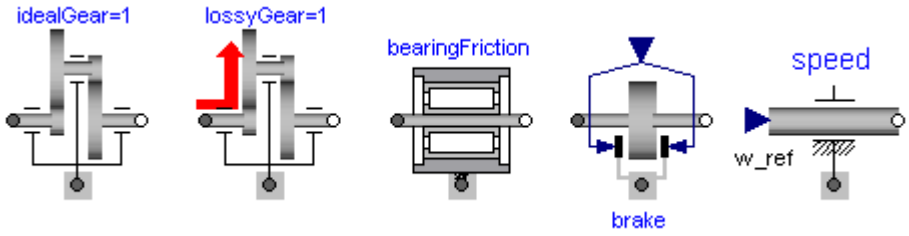
If needed, the angular velocity `w` and the angular acceleration `a` of a flange connector can be determined by differentiation of the flange angle `phi`:

```
w = der(phi);     a = der(w);
```

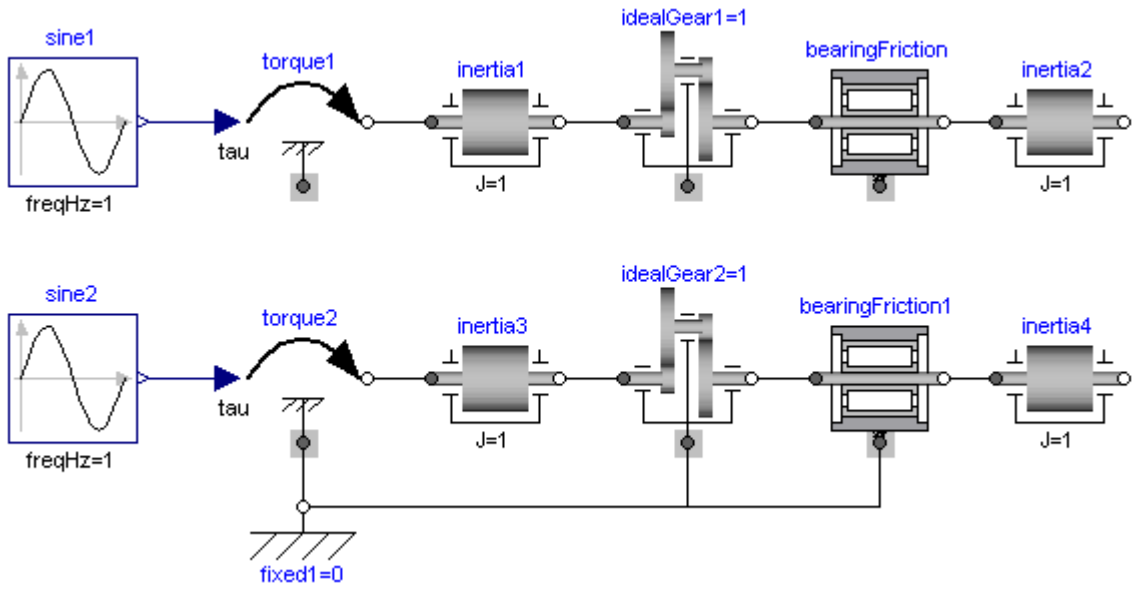
Modelica.Mechanics.Rotational.UsersGuide.SupportTorques

Support Torques

The following figure shows examples of components equipped with a bearing flange (framed flange in the lower center), which can be used to fix components on the ground or on other rotating elements or to combine them with force elements. If the bearing flange is not connected, the components are assumed to be mounted on the ground. Otherwise, the bearing connector offers the possibility to consider, e.g., gearboxes mounted on the ground via spring-damper-systems (cf. example `ElasticBearing`). Independently, these components provide a variable `tau_support` stating the support torque exerted on the bearing.



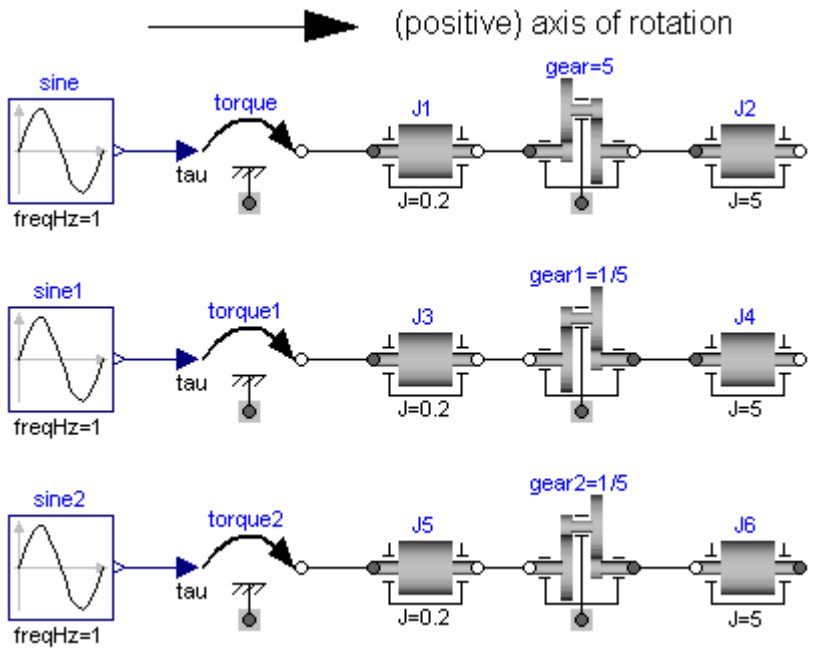
In general, it is not necessary to connect the bearing flange with a fixation, i.e., the two implementations in the following figure give identical results.



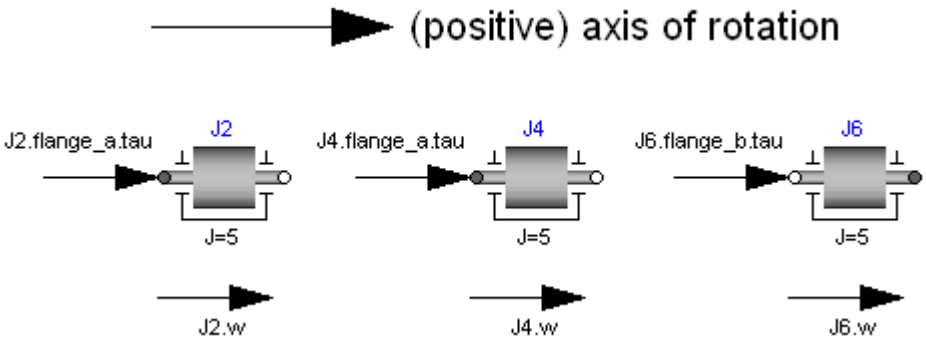
Modelica.Mechanics.Rotational.UsersGuide.SignConventions

Sign Conventions

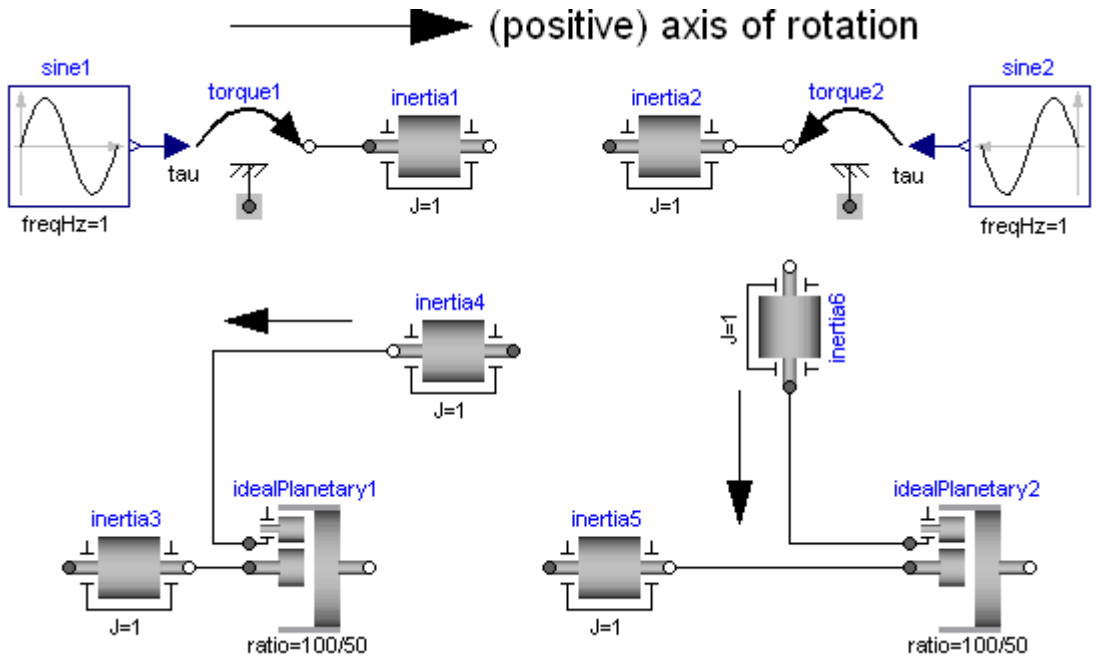
The variables of a component of this library can be accessed in the usual way. However, since most of these variables are basically elements of **vectors**, i.e., have a direction, the question arises how the signs of variables shall be interpreted. The basic idea is explained at hand of the following figure:



In the figure, three identical drive trains are shown. The only difference is that the gear of the middle drive train and the gear as well as the right inertia of the lower drive train are horizontally flipped with regards to the upper drive train. The signs of variables are now interpreted in the following way: Due to the 1-dimensional nature of the model, all components are basically connected together along one line (more complicated cases are discussed below). First, one has to define a **positive** direction of this line, called **axis of rotation**. In the top part of the figure this is characterized by an arrow defined as `axis of rotation`. The simple rule is now: If a variable of a component is positive and can be interpreted as the element of a vector (e.g. torque or angular velocity vector), the corresponding vector is directed into the positive direction of the axis of rotation. In the following figure, the right-most inertias of the figure above are displayed with the positive vector direction displayed according to this rule:



The cut-torques `J2.flange_a.tau, J4.flange_a.tau, J6.flange_b.tau` of the right inertias are all identical and are directed into the direction of rotation if the values are positive. Similiarily, the angular velocities `J2.w, J4.w, J6.w` of the right inertias are all identical and are also directed into the direction of rotation if the values are positive. Some special cases are shown in the next figure:



In the upper part of the figure, two variants of the connection of an external torque and an inertia are shown. In both cases, a positive signal input into the torque component accelerates the inertias `inertia1`, `inertia2` into the positive axis of rotation, i.e., the angular accelerations `inertia1.a`, `inertia2.a` are positive and are directed along the "axis of rotation" arrow. In the lower part of the figure the connection of inertias with a planetary gear is shown. Note, that the three flanges of the planetary gearbox are located along the axis of rotation and that the axis direction determines the positive rotation along these flanges. As a result, the positive rotation for `inertia4`, `inertia6` is as indicated with the additional grey arrows.

Modelica.Mechanics.Rotational.UsersGuide.UserDefinedComponents

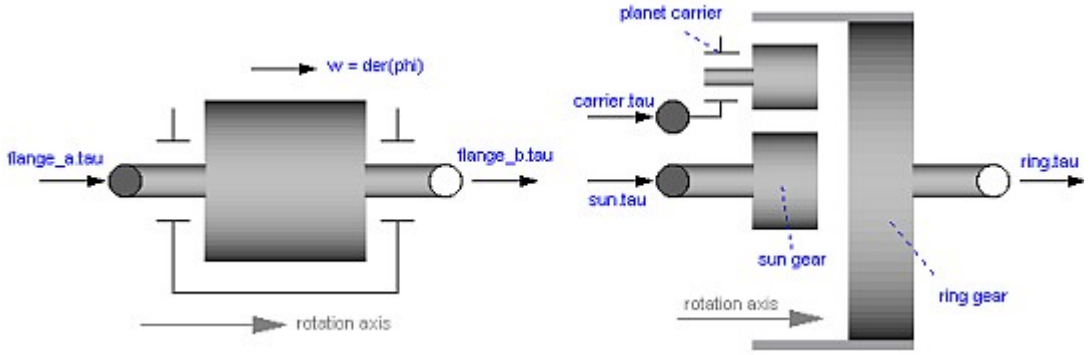
User Defined Components

In this section some hints are given to define your own 1-dimensional rotational components which are compatible with the elements of this package. It is convenient to define a new component by inheritance from one of the following base classes, which are defined in sublibrary Interfaces:

Name	Description
`Rigid`	Rigid connection of two rotational 1D flanges (used for elements with inertia).
`Compliant`	Compliant connection of two rotational 1D flanges (used for force laws such as a spring or a damper).
`TwoFlanges`	General connection of two rotational 1D flanges (used for gearboxes).
`AbsoluteSensor`	Measure absolute flange variables.
`RelativeSensor`	Measure relative flange variables.

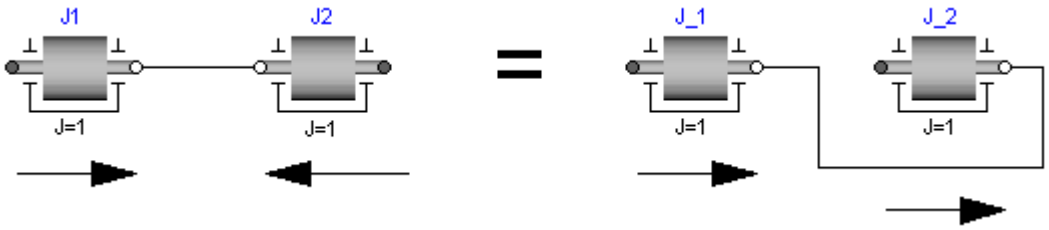
The difference between these base classes are the auxiliary variables defined in the model and the relations between the flange variables already defined in the base class. For example, in model **Rigid** the flanges flange_a and flange_b are rigidly connected, i.e., flange_a.phi = flange_b.phi, whereas in model **Compliant** the cut-torques are the same, i.e., flange_a.tau + flange_b.tau = 0.

The equations of a mechanical component are vector equations, i.e., they need to be expressed in a common coordinate system. Therefore, for a component a **local axis of rotation** has to be defined. All vector quantities, such as cut-torques or angular velocities have to be expressed according to this definition. Examples for such a definition are given in the following figure for an inertia component and a planetary gearbox:



As can be seen, all vectors are directed into the direction of the rotation axis. The angles in the flanges are defined correspondingly. For example, the angle `sun.phi` in the flange of the sun wheel of the planetary gearbox is positive, if rotated in mathematical positive direction (= counter clock wise) along the axis of rotation.

On first view, one may assume that the selected local coordinate system has an influence on the usage of the component. But this is not the case, as shown in the next figure:



In the figure the **local** axes of rotation of the components are shown. The connection of two inertias in the left and in the right part of the figure are completely equivalent, i.e., the right part is just a different drawing of the left part. This is due to the fact, that by a connection, the two local coordinate systems are made identical and the (automatically) generated connection equations (= angles are identical, cut-torques sum-up to zero) are also expressed in this common coordinate system. Therefore, even if in the left figure it seems to be that the angular velocity vector of `J2` goes from right to left, in reality it goes from left to right as shown in the right part of the figure, where the local coordinate systems are drawn such that they are aligned. Note, that the simple rule stated in section 4 (Sign conventions) also determines that the angular velocity of `J2` in the left part of the figure is directed from left to right.

To summarize, the local coordinate system selected for a component is just necessary, in order that the equations of this component are expressed correctly. The selection of the coordinate system is arbitrary and has no influence on the usage of the component. Especially, the actual direction of, e.g., a cut-torque is most easily determined by the rule of section 4. A more strict determination by aligning coordinate systems and then using the vector direction of the local coordinate systems, often requires a re-drawing of the diagram and is therefore less convenient to use.

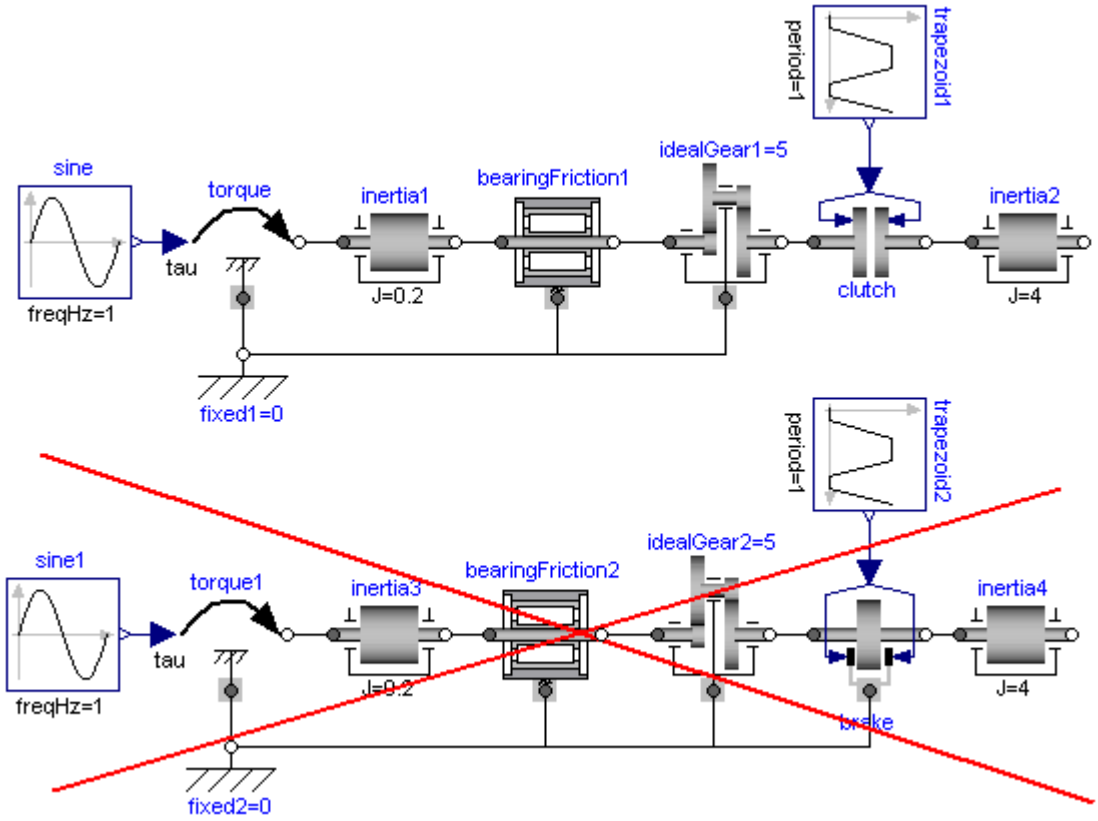
Modelica.Mechanics.Rotational.UsersGuide.RequirementsForSimulationTool

Requirements for Simulation Tools

This library is designed in a fully object oriented way in order that components can be connected together in every meaningful combination (e.g. direct connection of two springs or two inertias). As a consequence, most models lead to a system of differential-algebraic equations of **index 3** (= constraint equations have to be differentiated twice in order to arrive at a state space representation) and the Modelica translator or the simulator has to cope with this system representation. According to our present knowledge, this requires that the Modelica translator is able to symbolically differentiate equations (otherwise it is e.g. not possible to provide consistent initial conditions; even if consistent initial conditions are present, most numerical DAE integrators can cope at most with index 2

DAEs).

The elements of this library can be connected together in an arbitrary way. However, difficulties may occur, if the elements which can **lock** the **relative motion** between two flanges are connected **rigidly** together such that essentially the **same relative motion** can be locked. The reason is that the cut-torque in the locked phase is not uniquely defined if the elements are locked at the same time instant (i.e., there does not exist a unique solution) and some simulation systems may not be able to handle this situation, since this leads to a singularity during simulation. Currently, this type of problem can occur with the Coulomb friction elements **BearingFriction, Clutch, Brake, LossyGear** when the elements become stuck:



In the figure above two typical situations are shown: In the upper part of the figure, the series connection of rigidly attached BearingFriction and Clutch components are shown. This does not hurt, because the BearingFriction element can lock the relative motion between the element and the housing, whereas the clutch element can lock the relative motion between the two connected flanges. Contrary, the drive train in the lower part of the figure may rise to simulation problems, because the BearingFriction element and the Brake element can lock the relative motion between a flange and the housing and these flanges are rigidly connected together, i.e., essentially the same relative motion can be locked. These difficulties may be solved by either introducing a compliance between these flanges or by combining the BearingFriction and Brake element into one component and resolving the ambiguity of the frictional torque in the stuck mode. A tool may handle this situation also **automatically**, by picking one solution of the infinitely many, e.g., the one where the difference to the value of the previous time instant is as small as possible.

Modelica.Mechanics.Rotational.UsersGuide.ReleaseNotes

Release notes

- *December 12, 2005* by Martin Otter:
  Package documentation split into several parts and provided as User's Guide.
- *October 27, 2003* by Martin Otter and Christian Schweiger:
  Bearing flanges added for mounted components and support torque computation implemented.

New component `Torque2` and new example `ElasticBearing`.

- *October 21, 2002* by Martin Otter and Christian Schweiger:
  New components **LossyGear** (with corresponding examples) and **Gear2**.
  Interface **FrictionBase** adapted to new initialization.
- *June 19, 2000* by Martin Otter:
  New elements:

`IdealGearR2T`	Ideal gear transforming rotational in translational motion
`Position`	Forced movement of a flange with a reference angle given as input signal
`RelativeStates`	Definition of relative state variables

  Icon of Rotational.Torque changed. Elements Acceleration, Torque, Fixed, Sensors ordered according to the Translational library.
- *Nov. 4, 1999* by Martin Otter:
  Improved documentation and improved graphical layout of the diagram level. Changes according to the Twente meeting introduced. Especially: Alias names, instead of extends. Model Shaft renamed to Inertia. Torque1D renamed to Torque. AccMotion renamed to Accelerate. LockedL, LockedR replaced by Fixed. SpeedSensor splitted into AngleSensor and SpeedSensor. RelSpeedSensor splitted into RelAngleSensor and RelSpeedSensor. Initialization of friction elements improved. Flanges renamed to flange_a, flange_b. MoveAngle renamed to KinematicPTP, vectorized and moved to Blocks.Sources.
  Advice given from P. Beater, H. Elmqvist, S.E. Mattsson, H. Olsson is appreciated.
- *July 18, 1999* by Martin Otter:
  Documentation and icons improved. Appropriate initial conditions introduced as start values in the demo models. Bearing model replaced by FixedRight and FixedLeft models; sensor elements replaced by TorqueSensor, SpeedSensor, AccSensor; new sensor elements RelSpeedSensor, RelAccSensor to measure relative kinematic quantitites. New elements GearEfficiency and Gear.
- *June 30, 1999* by Martin Otter:
  Realized a first version based on an existing Dymola library of Martin Otter and Hilding Elmqvist.

Modelica.Mechanics.Rotational.UsersGuide.Contact

Contact

Main Authors:

Martin Otter and Christian Schweiger
Deutsches Zentrum für Luft und Raumfahrt e.V. (DLR)
Institut für Robotik und Mechatronik
Abteilung für Entwurfsorientierte Regelungstechnik
Postfach 1116
D-82230 Wessling
Germany
email: Martin.Otter@dlr.de, Christian.Schweiger@dlr.de

Modelica.Mechanics.Rotational.Examples

Demonstration examples of the components of this package

Information

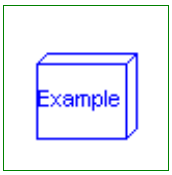
This package contains example models to demonstrate the usage of the Modelica.Mechanics.Rotational package. Open the models and simulate them according to the provided description in the models.

Package Content

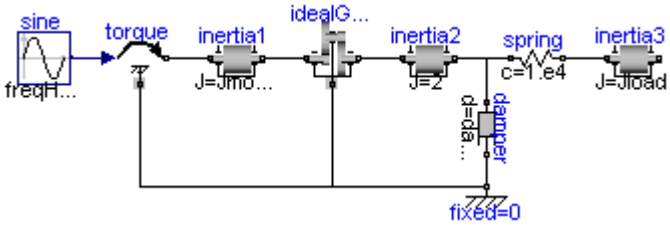
Name	Description

First	First example: simple drive train
Friction	Drive train with clutch and brake
CoupledClutches	Drive train with 3 dynamically coupled clutches
LossyGearDemo1	Example to show that gear efficiency may lead to stuck motion
LossyGearDemo2	Example to show combination of LossyGear and BearingFriction
ElasticBearing	Example to show possible usage of bearing flange

Modelica.Mechanics.Rotational.Examples.First



First example: simple drive train



Information

The drive train consists of a motor inertia which is driven by a sine-wave motor torque. Via a gearbox the rotational energy is transmitted to a load inertia. Elasticity in the gearbox is modeled by a spring element. A linear damper is used to model the damping in the gearbox bearing.

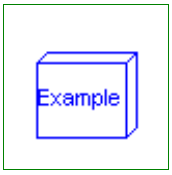
Note, that a force component (like the damper of this example) which is acting between a shaft and the housing has to be fixed in the housing on one side via component Fixed.

Simulate for 1 second and plot the following variables:
angular velocities of inertias inertia2 and 3: inertia2.w, inertia3.w

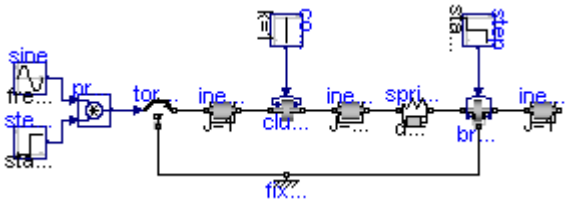
Parameters

Type	Name	Default	Description
Real	amplitude	10	
Frequency	freqHz	5	[Hz]
Inertia	Jmotor	0.1	[kg.m2]
Inertia	Jload	2	[kg.m2]
Real	ratio	10	
Real	damping	10	

Modelica.Mechanics.Rotational.Examples.Friction



Drive train with clutch and brake



Information

This drive train contains a frictional **clutch** and a **brake**. Simulate the system for 1 second using the following initial values (defined already in the model):

```
inertia1.w =  90 (or brake.w)
inertia2.w =  90
inertia3.w = 100
```

Plot the output signals

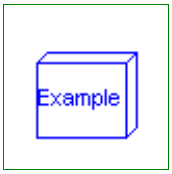
```
tMotor      Torque of motor
tClutch     Torque in clutch
tBrake      Torque in brake
tSpring     Torque in spring
```

as well as the absolute angular velocities of the three inertia components (inertia1.w, inertia2.w, inertia3.w).

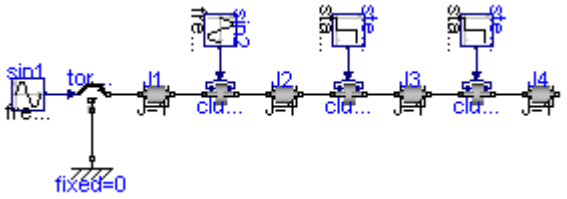
Parameters

Type	Name	Default	Description
Time	startTime	0.5	Start time of step [s]

Modelica.Mechanics.Rotational.Examples.CoupledClutches



Drive train with 3 dynamically coupled clutches



Information

This example demonstrates how variable structure drive trains are handeled. The drive train consists of 4 inertias and 3 clutches, where the clutches are controlled by input signals. The system has 2^3=8 different configurations and 3^3 = 27 different states (every clutch may be in forward sliding, backward sliding or locked mode when the relative angular velocity is zero). By invoking the clutches at different time instances, the switching of the configurations can be studied.

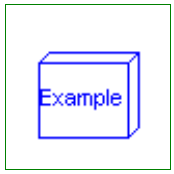
Simulate the system for 1.2 seconds with the following initial values:
J1.w = 10.

Plot the following variables:
angular velocities of inertias (J1.w, J2.w, J3.w, J4.w), frictional torques of clutches (clutchX.tau), frictional mode of clutches (clutchX.mode) where mode = -1/0/+1 means backward sliding, locked, forward sliding.

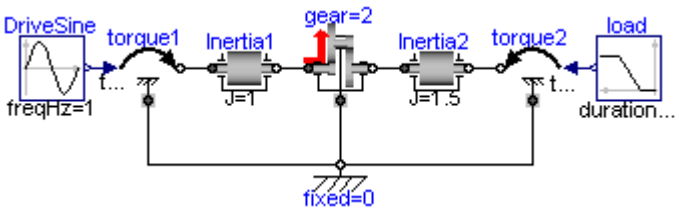
Parameters

Type	Name	Default	Description
Frequency	freqHz	0.2	frequency of sine function to invoke clutch1 [Hz]
Time	T2	0.4	time when clutch2 is invoked [s]
Time	T3	0.9	time when clutch3 is invoked [s]

Modelica.Mechanics.Rotational.Examples.LossyGearDemo1



Example to show that gear efficiency may lead to stuck motion



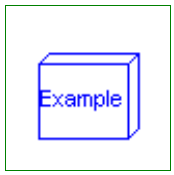
Information

This model contains two inertias which are connected by an ideal gear where the friction between the teeth of the gear is modeled in a physical meaningful way (friction may lead to stuck mode which locks the motion of the gear). The friction is defined by an efficiency factor (= 0.5) for forward and backward driving condition leading to a torque dependent friction loss. Simulate for about 0.5 seconds. The friction in the gear will take all modes (forward and backward rolling, as well as stuck).

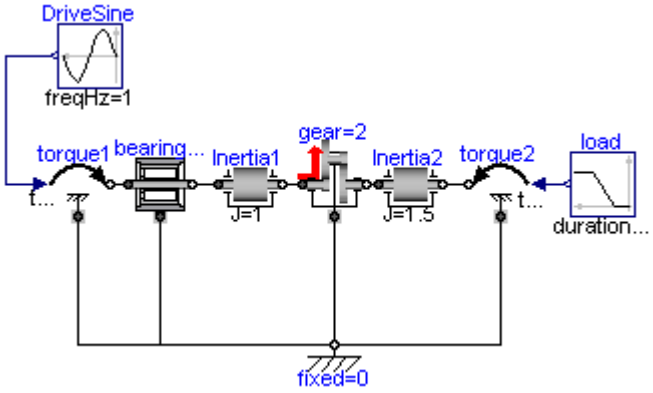
You may plot:

```
Inertia1.w,
Inertia2.w : angular velocities of inertias
powerLoss  : power lost in the gear
gear.mode  :  1 = forward rolling
              0 = stuck (w=0)
             -1 = backward rolling
```

Modelica.Mechanics.Rotational.Examples.LossyGearDemo2



Example to show combination of LossyGear and BearingFriction



Information

This model contains bearing friction and gear friction (= efficiency). If both friction models are stuck, there is no unique solution. Still a reliable Modelica simulator, such as Dymola, should be able to handle this situation.

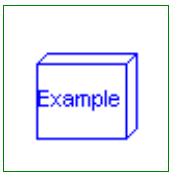
Simulate for about 0.5 seconds. The friction elements are in all modes (forward and backward rolling, as well as stuck).

You may plot:

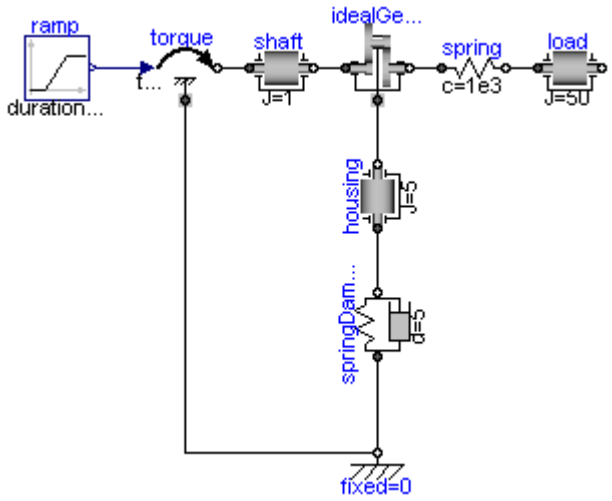
```
Inertia1.w,
Inertia2.w           : angular velocities of inertias
powerLoss            : power lost in the gear
bearingFriction.mode:  1 = forward rolling
                       0 = stuck (w=0)
                      -1 = backward rolling
gear.mode            :  1 = forward rolling
                       0 = stuck (w=0)
                      -1 = backward rolling
```

Note: This combination of LossyGear and BearingFriction is not recommended to use, as component LossyGear includes the functionality of component BearingFriction (only *peak* not supported).

Modelica.Mechanics.Rotational.Examples.ElasticBearing



Example to show possible usage of bearing flange



Information

This model demonstrates the usage of the bearing flange. The gearbox is not connected rigidly to the ground, but by a spring-damper-system. This allows examination of the gearbox housing dynamics.

Simulate for about 10 seconds and plot the angular velocities of the inertias `housing.w`, `shaft.w` and `load.w`.

Modelica.Mechanics.Rotational.Sensors

Sensors to measure variables in 1D rotational mechanical components

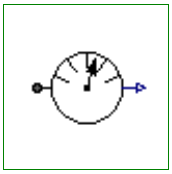
Information

This package contains ideal sensor components that provide the connector variables as signals for further processing with the Modelica.Blocks library.

Package Content

Name	Description
AngleSensor	Ideal sensor to measure the absolute flange angle
SpeedSensor	Ideal sensor to measure the absolute flange angular velocity
AccSensor	Ideal sensor to measure the absolute flange angular acceleration
RelAngleSensor	Ideal sensor to measure the relative angle between two flanges
RelSpeedSensor	Ideal sensor to measure the relative angular velocity between two flanges
RelAccSensor	Ideal sensor to measure the relative angular acceleration between two flanges
TorqueSensor	Ideal sensor to measure the torque between two flanges (= flange_a.tau)
PowerSensor	Ideal sensor to measure the power between two flanges (= flange_a.tau*der(flange_a.phi))

Modelica.Mechanics.Rotational.Sensors.AngleSensor



Ideal sensor to measure the absolute flange angle

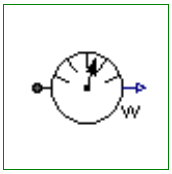
Information

Measures the **absolute angle phi** of a flange in an ideal way and provides the result as output signal **phi** (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	flange to be measured
output RealOutput	phi	Absolute angle of flange

Modelica.Mechanics.Rotational.Sensors.SpeedSensor



Ideal sensor to measure the absolute flange angular velocity

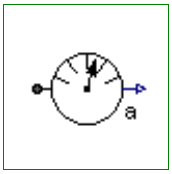
Information

Measures the **absolute angular velocity w** of a flange in an ideal way and provides the result as output signal **w** (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	flange to be measured
output RealOutput	w	Absolute angular velocity of flange

Modelica.Mechanics.Rotational.Sensors.AccSensor



Ideal sensor to measure the absolute flange angular acceleration

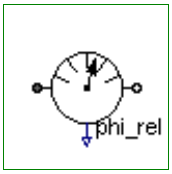
Information

Measures the **absolute angular acceleration a** of a flange in an ideal way and provides the result as output signal **a** (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	flange to be measured
output RealOutput	a	Absolute angular acceleration of flange

Modelica.Mechanics.Rotational.Sensors.RelAngleSensor



Ideal sensor to measure the relative angle between two flanges

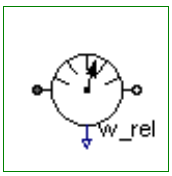
Information

Measures the **relative angle phi_rel** between two flanges in an ideal way and provides the result as output signal **phi_rel** (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
output RealOutput	phi_rel	Relative angle between two flanges (= flange_b.phi - flange_a.phi)

Modelica.Mechanics.Rotational.Sensors.RelSpeedSensor



Ideal sensor to measure the relative angular velocity between two flanges

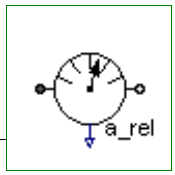
Information

Measures the **relative angular velocity w_rel** between two flanges in an ideal way and provides the result as output signal **w_rel** (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
output RealOutput	w_rel	Relative angular velocity between two flanges (= der(flange_b.phi) - der(flange_a.phi))

Modelica.Mechanics.Rotational.Sensors.RelAccSensor



Ideal sensor to measure the relative angular acceleration between two flanges

Information

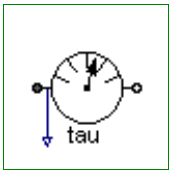
Measures the **relative angular acceleration a_rel** between two flanges in an ideal way and provides the result as output signal **a_rel** (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
output RealOutput	a_rel	Relative angular acceleration between two flanges

---

Modelica.Mechanics.Rotational.Sensors.TorqueSensor



Ideal sensor to measure the torque between two flanges (= flange_a.tau)

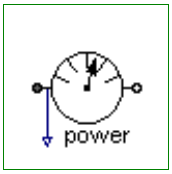
Information

Measures the **cut-torque between two flanges** in an ideal way and provides the result as output signal **tau** (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
output RealOutput	tau	Torque in flange flange_a and flange_b (= flange_a.tau = -flange_b.tau)

Modelica.Mechanics.Rotational.Sensors.PowerSensor



Ideal sensor to measure the power between two flanges (= flange_a.tau*der(flange_a.phi))

Information

Measures the **power between two flanges** in an ideal way and provides the result as output signal **power** (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
output RealOutput	power	Power in flange flange_a

Modelica.Mechanics.Rotational.Interfaces

Connectors and partial models for 1D rotational mechanical components

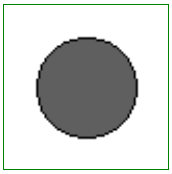
Information

This package contains connectors and partial models for 1-dim. rotational mechanical components. The components of this package can only be used as basic building elements for models.

Package Content

Name	Description
Flange_a	1D rotational flange (filled square icon)
Flange_b	1D rotational flange (non-filled square icon)
Rigid	Base class for the rigid connection of two rotational 1D flanges
Compliant	Base class for the compliant connection of two rotational 1D flanges
TwoFlanges	Base class for a component with two rotational 1D flanges
Bearing	Base class for interface classes with bearing connector
TwoFlangesAndBearing	Base class for a equation-based component with two rotational 1D flanges and one rotational 1D bearing flange
TwoFlangesAndBearingH	Base class for a hierarchically composed component with two rotational 1D flanges and one rotational bearing flange
FrictionBase	Base class of Coulomb friction elements
AbsoluteSensor	Base class to measure a single absolute flange variable
RelativeSensor	Base class to measure a single relative variable between two flanges
PartialSpeedDependentTorque	Partial model of a torque acting at the flange (accelerates the flange)

Modelica.Mechanics.Rotational.Interfaces.Flange_a



1D rotational flange (filled square icon)

Information

This is a connector for 1D rotational mechanical systems and models a mechanical flange. The following variables are defined in this connector:

```
phi: Absolute rotation angle of the flange in [rad].
tau: Cut-torque in the flange in [Nm].
```

There is a second connector for flanges: Flange_b. The connectors Flange_a and Flange_b are completely identical. There is only a difference in the icons, in order to easier identify a flange variable in a diagram. For a discussion on the actual direction of the cut-torque tau and of the rotation angle, see the information text of package Rotational (section 4. Sign conventions).

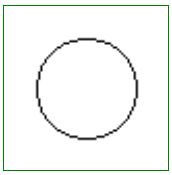
If needed, the absolute angular velocity w and the absolute angular acceleration a of the flange can be determined by differentiation of the flange angle phi:

```
w = der(phi);     a = der(w)
```

Contents

Type	Name	Description
Angle	phi	Absolute rotation angle of flange [rad]
flow Torque	tau	Cut torque in the flange [N.m]

Modelica.Mechanics.Rotational.Interfaces.Flange_b



1D rotational flange (non-filled square icon)

Information

This is a connector for 1D rotational mechanical systems and models a mechanical flange. The following variables are defined in this connector:

```
phi: Absolute rotation angle of the flange in [rad].
tau: Cut-torque in the flange in [Nm].
```

There is a second connector for flanges: Flange_a. The connectors Flange_a and Flange_b are completely identical. There is only a difference in the icons, in order to easier identify a flange variable in a diagram. For a discussion on the actual direction of the cut-torque tau and of the rotation angle, see the information text of package Rotational (section 4. Sign conventions).

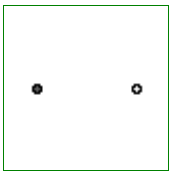
If needed, the absolute angular velocity w and the absolute angular acceleration a of the flange can be determined by differentiation of the flange angle phi:

```
w = der(phi);     a = der(w)
```

Contents

Type	Name	Description
Angle	phi	Absolute rotation angle of flange [rad]
flow Torque	tau	Cut torque in the flange [N.m]

Modelica.Mechanics.Rotational.Interfaces.Rigid



Base class for the rigid connection of two rotational 1D flanges

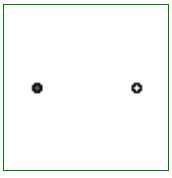
Information

This is a 1D rotational component with two rigidly connected flanges, i.e., flange_a.phi = flange_b.phi. It is used e.g. to built up components with inertia.

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Rotational.Interfaces.Compliant



Base class for the compliant connection of two rotational 1D flanges

Information

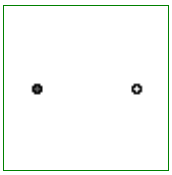
This is a 1D rotational component with a compliant connection of two rotational 1D flanges where inertial effects between the two flanges are neglected. The basic assumption is that the cut-torques of the two flanges sum-up to zero, i.e., they have the same absolute value but opposite sign: flange_a.tau +

flange_b.tau = 0. This base class is used to built up force elements such as springs, dampers, friction.

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Rotational.Interfaces.TwoFlanges



Base class for a component with two rotational 1D flanges

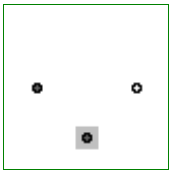
Information

This is a 1D rotational component with two flanges. It is used e.g. to build up parts of a drive train consisting of several base components. There are specialized versions of this base class for rigidly connected flanges (Interfaces.Rigid) and for a compliant connection of flanges (Interfaces.Compliant).

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	

Modelica.Mechanics.Rotational.Interfaces.Bearing



Base class for interface classes with bearing connector

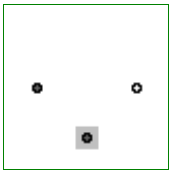
Information

This is a 1D rotational component with two flanges and an additional bearing flange. It is a superclass for the two components TwoFlangesAndBearing and TwoFlangesAndBearingH.

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.Interfaces.TwoFlangesAndBearing



Base class for a equation-based component with two rotational 1D flanges and one rotational 1D bearing flange

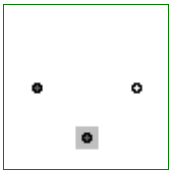
Information

This is a 1D rotational component with two flanges and an additional bearing flange. It is used e.g. to build up equation-based parts of a drive train.

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.Interfaces.TwoFlangesAndBearingH



Base class for a hierarchically composed component with two rotational 1D flanges and one rotational bearing flange

Information

This is a 1D rotational component with two flanges and an additional bearing flange. It is used e.g. to build up parts of a drive train consisting of several base components.

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.Interfaces.FrictionBase

Base class of Coulomb friction elements

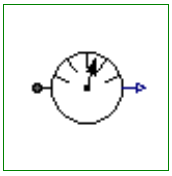
Information

Basic model for Coulomb friction that models the stuck phase in a reliable way.

Parameters

Type	Name	Default	Description
Advanced			
AngularVelocity	w_small	1e10	Relative angular velocity near to zero if jumps due to a reinit(..) of the velocity can occur (set to low value only if such impulses can occur) [rad/s]

Modelica.Mechanics.Rotational.Interfaces.AbsoluteSensor



Base class to measure a single absolute flange variable

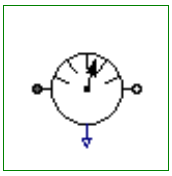
Information

This is the base class of a 1D rotational component with one flange and one output signal y in order to measure an absolute kinematic quantity in the flange and to provide the measured signal as output signal for further processing with the blocks of package Modelica.Blocks.

Connectors

Type	Name	Description
Flange_a	flange_a	(left) flange to be measured (flange axis directed INTO cut plane)
output RealOutput	y	

Modelica.Mechanics.Rotational.Interfaces.RelativeSensor



Base class to measure a single relative variable between two flanges

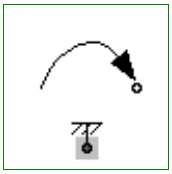
Information

This is a base class for 1D rotational components with two rigidly connected flanges and one output signal y in order to measure relative kinematic quantities between the two flanges or the cut-torque in the flange and to provide the measured signal as output signal for further processing with the blocks of package Modelica.Blocks.

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
output RealOutput	y	

Modelica.Mechanics.Rotational.Interfaces.PartialSpeedDependentTorque



Partial model of a torque acting at the flange (accelerates the flange)

Information

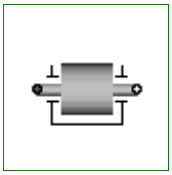
Partial model of torque dependent on speed that accelerates the flange.

Connectors

Type	Name	Description
Flange_b	flange	Flange on which torque is acting
Flange_a	bearing	Bearing at which the reaction torque (i.e., -flange.tau) is acting

---

Modelica.Mechanics.Rotational.Inertia



1D-rotational component with inertia

Information

Rotational component with **inertia** and two rigidly connected flanges.

Parameters

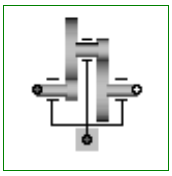
Type	Name	Default	Description
Inertia	J	1	Moment of inertia [kg.m2]

Initialization			
Temp	initType	Modelica.Mechanics.Rotationa..	Type of initialization (defines usage of start values below)
Angle	phi_start	0	Initial or guess value of rotor rotation angle phi [rad]
AngularVelocity	w_start	0	Initial or guess value of angular velocity w = der(phi) [rad/s]
AngularAcceleration	a_start	0	Initial value of angular acceleration a = der(w) [rad/s2]
Angle	phi.start	phi_start	Absolute rotation angle of component (= flange_a.phi = flange_b.phi) [rad]
Advanced			
Temp	stateSelection	Modelica.Blocks.Types.StateS...	Priority to use phi and w as states

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Rotational.IdealGear



Ideal gear without inertia

Information

This element characterices any type of gear box which is fixed in the ground and which has one driving shaft and one driven shaft. The gear is **ideal**, i.e., it does not have inertia, elasticity, damping or backlash. If these effects have to be considered, the gear has to be connected to other elements in an appropriate way.

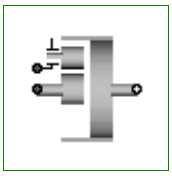
Parameters

Type	Name	Default	Description
Real	ratio	1	Transmission ratio (flange_a.phi/flange_b.phi)

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.IdealPlanetary



Ideal planetary gear box

Information

The IdealPlanetary gear box is an ideal gear without inertia, elasticity, damping or backlash consisting of an inner **sun** wheel, an outer **ring** wheel and a **planet** wheel located between sun and ring wheel. The bearing of the planet wheel shaft is fixed in the planet **carrier**. The component can be connected to other elements at

the sun, ring and/or carrier flanges. It is not possible to connect to the planet wheel. If inertia shall not be neglected, the sun, ring and carrier inertias can be easily added by attaching inertias (= model Inertia) to the corresponding connectors. The inertias of the planet wheels are always neglected.

The icon of the planetary gear signals that the sun and carrier flanges are on the left side and the ring flange is on the right side of the gear box. However, this component is generic and is valid independantly how the flanges are actually placed (e.g. sun wheel may be placed on the right side instead on the left side in reality).

The ideal planetary gearbox is uniquely defined by the ratio of the number of ring teeth zr with respect to the number of sun teeth zs. For example, if there are 100 ring teeth and 50 sun teeth then ratio = zr/zs = 2. The number of planet teeth zp has to fulfill the following relationship:

```
zp := (zr - zs) / 2
```

Therefore, in the above example zp = 25 is required.

According to the overall convention, the positive direction of all vectors, especially the absolute angular velocities and cut-torques in the flanges, are along the axis vector displayed in the icon.

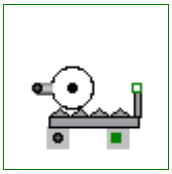
Parameters

Type	Name	Default	Description
Real	ratio	100/50	number of ring_teeth/sun_teeth (e.g. ratio=100/50)

Connectors

Type	Name	Description
Flange_a	sun	sun flange (flange axis directed INTO cut plane)
Flange_a	carrier	carrier flange (flange axis directed INTO cut plane)
Flange_b	ring	ring flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Rotational.IdealGearR2T



Gearbox transforming rotational into translational motion

Information

This is an ideal mass- and inertialess gearbox which transforms a 1D-rotational into a 1D-translational motion. If elasticity, damping or backlash has to be considered, this ideal gearbox has to be connected with corresponding elements.

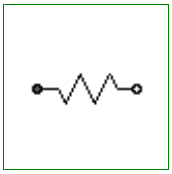
Parameters

Type	Name	Default	Description
Real	ratio	1	transmission ratio (flange_a.phi/flange_b.s) [rad/m]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearingR	
Flange_a	bearingT	

Modelica.Mechanics.Rotational.Spring



Linear 1D rotational spring

Information

A **linear 1D rotational spring**. The component can be connected either between two inertias/gears to describe the shaft elasticity, or between a inertia/gear and the housing (component Fixed), to describe a coupling of the element with the housing via a spring.

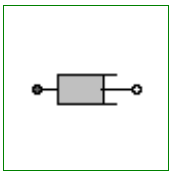
Parameters

Type	Name	Default	Description
Real	c		Spring constant [N.m/rad]
Angle	phi_rel0	0	Unstretched spring angle [rad]
Initialization			
Angle	phi_rel.start	0	Relative rotation angle (= flange_b.phi - flange_a.phi) [rad]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Rotational.Damper



Linear 1D rotational damper

Information

Linear, velocity dependent damper element. It can be either connected between an inertia or gear and the housing (component Fixed), or between two inertia/gear elements.

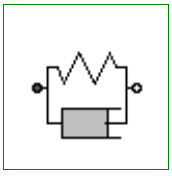
Parameters

Type	Name	Default	Description
Real	d	0	Damping constant [N.m.s/rad]
Initialization			
Angle	phi_rel.start	0	Relative rotation angle (= flange_b.phi - flange_a.phi) [rad]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Rotational.SpringDamper



Linear 1D rotational spring and damper in parallel

Information

A **spring** and **damper** element **connected in parallel**. The component can be connected either between two inertias/gears to describe the shaft elasticity and damping, or between an inertia/gear and the housing (component Fixed), to describe a coupling of the element with the housing via a spring/damper.

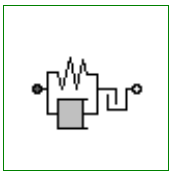
Parameters

Type	Name	Default	Description
Real	c		Spring constant [N.m/rad]
Angle	phi_rel0	0	Unstretched spring angle [rad]
Real	d	0	Damping constant [N.m.s/rad]
Initialization			
Temp	initType	Modelica.Mechanics.Rotationa...	Type of initialization (defines usage of start values below)
Angle	phi_rel_start	0	Initial or guess value of relative rotation angle phi_rel [rad]
AngularVelocity	w_rel_start	0	Initial or guess value of relative angular velocity w_rel = der(phi_rel) [rad/s]
Angle	phi_rel.start	phi_rel_start	Relative rotation angle (= flange_b.phi - flange_a.phi) [rad]
Advanced			
Temp	stateSelection	Modelica.Blocks.Types.StateS...	Priority to use phi_rel and w_rel as states

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Rotational.ElastoBacklash



Backlash connected in series to linear spring and damper (backlash is modeled with elasticity)

Information

This element consists of a **backlash** element **connected in series** to a **spring** and **damper** element which are **connected in parallel**. The spring constant shall be non-zero, otherwise the component cannot be used.

In combination with components IdealGear, the ElastoBacklash model can be used to model a gear box with backlash, elasticity and damping.

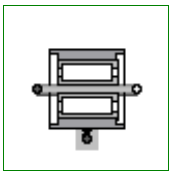
Parameters

Type	Name	Default	Description
Angle	b	0	Total backlash [rad]
Real	c	1.e5	Spring constant (c > 0 required) [N.m/rad]
Angle	phi_rel0	0	Unstretched spring angle [rad]
Real	d	0	Damping constant [N.m.s/rad]
Initialization			
Angle	phi_rel.start	0	Relative rotation angle (= flange_b.phi - flange_a.phi) [rad]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Rotational.BearingFriction



Coulomb friction in bearings

Information

This element describes **Coulomb friction** in **bearings**, i.e., a frictional torque acting between a flange and the housing. The positive sliding friction torque "tau" has to be defined by table "tau_pos" as function of the absolute angular velocity "w". E.g.

```
 w | tau
---+-----
 0 |   0
 1 |   2
 2 |   5
 3 |   8
```

gives the following table:

```
tau_pos = [0, 0; 1, 2; 2, 5; 3, 8];
```

Currently, only linear interpolation in the table is supported. Outside of the table, extrapolation through the last two table entries is used. It is assumed that the negative sliding friction force has the same characteristic with negative values. Friction is modelled in the following way:

When the absolute angular velocity "w" is not zero, the friction torque is a function of w and of a constant normal force. This dependency is defined via table tau_pos and can be determined by measurements, e.g. by driving the gear with constant velocity and measuring the needed motor torque (= friction torque).

When the absolute angular velocity becomes zero, the elements connected by the friction element become stuck, i.e., the absolute angle remains constant. In this phase the friction torque is calculated from a torque balance due to the requirement, that the absolute acceleration shall be zero. The elements begin to slide when the friction torque exceeds a threshold value, called the maximum static friction torque, computed via:

```
maximum_static_friction = peak * sliding_friction(w=0)   (peak >= 1)
```

This procedure is implemented in a "clean" way by state events and leads to continuous/discrete systems of equations if friction elements are dynamically coupled which have to be solved by appropriate numerical methods. The method is described in:

Otter M., Elmqvist H., and Mattsson S.E. (1999):
Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. CACSD'99, Aug. 22.-26, Hawaii.

More precise friction models take into account the elasticity of the material when the two elements are "stuck", as well as other effects, like hysteresis. This has the advantage that the friction element can be completely described by a differential equation without events. The drawback is that the system becomes stiff (about 10-20 times slower simulation) and that more material constants have to be supplied which requires more sophisticated identification. For more details, see the following references, especially (Armstrong and Canudas de Witt 1996):

Armstrong B. (1991):
Control of Machines with Friction. Kluwer Academic Press, Boston MA.

Armstrong B., and Canudas de Wit C. (1996):
Friction Modeling and Compensation. The Control Handbook, edited by W.S.Levine, CRC Press, pp. 1369-1382.

Canudas de Wit C., Olsson H., Astroem K.J., and Lischinsky P. (1995):
A new model for control of systems with friction. IEEE Transactions on Automatic Control, Vol. 40, No. 3, pp. 419-425.

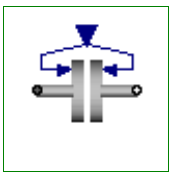
Parameters

Type	Name	Default	Description
Real	tau_pos[:, :]	[0, 1]	[w,tau] Positive sliding friction characteristic (w>=0)
Real	peak	1	peak*tau_pos[1,2] = Maximum friction torque for w==0
Initialization			
Boolean	startForward.start	**false**	true, if w_rel=0 and start of forward sliding or w_rel > w_small
Boolean	startBackward.start	**false**	true, if w_rel=0 and start of backward sliding or w_rel < -w_small
Boolean	locked.start	false	true, if w_rel=0 and not sliding
Advanced			
AngularVelocity	w_small	1e10	Relative angular velocity near to zero if jumps due to a reinit(..) of the velocity can occur (set to low value only if such impulses can occur) [rad/s]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.Clutch



Clutch based on Coulomb friction

Information

This component models a **clutch**, i.e., a component with two flanges where friction is present between the two flanges and these flanges are pressed together via a normal force. The normal force fn has to be provided as input signal f_normalized in a normalized form (0 ≤ f_normalized ≤ 1), fn = fn_max*f_normalized, where fn_max has to be provided as parameter. Friction in the clutch is modelled in the following way:

When the relative angular velocity is not zero, the friction torque is a function of the velocity dependent friction coefficient mue(w_rel) , of the normal force "fn", and of a geometry constant "cgeo" which takes into account the geometry of the device and the assumptions on the friction distributions:

```
frictional_torque = cgeo * mue(w_rel) * fn
```

Typical values of coefficients of friction:

```
dry operation    :  mue = 0.2 .. 0.4
operating in oil:  mue = 0.05 .. 0.1
```

When plates are pressed together, where **ri** is the inner radius, **ro** is the outer radius and **N** is the number of

friction interfaces, the geometry constant is calculated in the following way under the assumption of a uniform rate of wear at the interfaces:

```
cgeo = N*(r0 + ri)/2
```

The positive part of the friction characteristic **mue**(w_rel), w_rel >= 0, is defined via table mue_pos (first column = w_rel, second column = mue). Currently, only linear interpolation in the table is supported.

When the relative angular velocity becomes zero, the elements connected by the friction element become stuck, i.e., the relative angle remains constant. In this phase the friction torque is calculated from a torque balance due to the requirement, that the relative acceleration shall be zero. The elements begin to slide when the friction torque exceeds a threshold value, called the maximum static friction torque, computed via:

```
frictional_torque = peak * cgeo * mue(w_rel=0) * fn    (peak >= 1)
```

This procedure is implemented in a "clean" way by state events and leads to continuous/discrete systems of equations if friction elements are dynamically coupled. The method is described in:

Otter M., Elmqvist H., and Mattsson S.E. (1999):
Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. CACSD'99, Aug. 22.-26, Hawaii.

More precise friction models take into account the elasticity of the material when the two elements are "stuck", as well as other effects, like hysteresis. This has the advantage that the friction element can be completely described by a differential equation without events. The drawback is that the system becomes stiff (about 10-20 times slower simulation) and that more material constants have to be supplied which requires more sophisticated identification. For more details, see the following references, especially (Armstrong and Canudas de Witt 1996):

Armstrong B. (1991):
Control of Machines with Friction. Kluwer Academic Press, Boston MA.

Armstrong B., and Canudas de Wit C. (1996):
Friction Modeling and Compensation. The Control Handbook, edited by W.S.Levine, CRC Press, pp. 1369-1382.

Canudas de Wit C., Olsson H., Astroem K.J., and Lischinsky P. (1995):
A new model for control of systems with friction. IEEE Transactions on Automatic Control, Vol. 40, No. 3, pp. 419-425.

Parameters

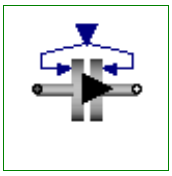
Type	Name	Default	Description
Real	mue_pos[:, :]	[0, 0.5]	[w,mue] positive sliding friction coefficient (w_rel>=0)
Real	peak	1	peak*mue_pos[1,2] = maximum value of mue for w_rel==0
Real	cgeo	1	Geometry constant containing friction distribution assumption
Force	fn_max	1	Maximum normal force [N]
Initialization			
Angle	phi_rel.start	0	Relative rotation angle (= flange_b.phi - flange_a.phi) [rad]
Boolean	startForward.start	**false**	true, if w_rel=0 and start of forward sliding or w_rel > w_small
Boolean	startBackward.start	**false**	true, if w_rel=0 and start of backward sliding or w_rel < -w_small
Boolean	locked.start	false	true, if w_rel=0 and not sliding
Advanced			

AngularVelocity	w_small	1e10	Relative angular velocity near to zero if jumps due to a reinit(..) of the velocity can occur (set to low value only if such impulses can occur) [rad/s]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
input RealInput	f_normalized	Normalized force signal 0..1 (normal force = fn_max*f_normalized; clutch is engaged if > 0)

Modelica.Mechanics.Rotational.OneWayClutch



Series connection of freewheel and clutch

Information

This component models a **one-way clutch**, i.e., a component with two flanges where friction is present between the two flanges and these flanges are pressed together via a normal force. These flanges maybe sliding with respect to each other Parallel connection of ClutchCombi and of FreeWheel. The element is introduced to resolve the ambiguity of the constraint torques of the elements.

A one-way-clutch is an element where a clutch is connected in parallel to a free wheel. This special element is provided, because such a parallel connection introduces an ambiguity into the model (the constraint torques are not uniquely defined when both elements are stuck) and this element resolves it by introducing **one** constraint torque and not two.

Note, initial values have to be chosen for the model, such that the relative speed of the one-way-clutch >= 0. Otherwise, the configuration is physically not possible and an error occurs.

The normal force fn has to be provided as input signal f_normalized in a normalized form (0 ≤ f_normalized ≤ 1), fn = fn_max*f_normalized, where fn_max has to be provided as parameter. Friction in the clutch is modelled in the following way:

When the relative angular velocity is positive, the friction torque is a function of the velocity dependent friction coefficient mue(w_rel) , of the normal force "fn", and of a geometry constant "cgeo" which takes into account the geometry of the device and the assumptions on the friction distributions:

```
frictional_torque = cgeo * mue(w_rel) * fn
```

Typical values of coefficients of friction:

```
dry operation    :  mue = 0.2 .. 0.4
operating in oil:  mue = 0.05 .. 0.1
```

When plates are pressed together, where **ri** is the inner radius, **ro** is the outer radius and **N** is the number of friction interfaces, the geometry constant is calculated in the following way under the assumption of a uniform rate of wear at the interfaces:

```
cgeo = N*(r0 + ri)/2
```

The positive part of the friction characteristic **mue**(w_rel), w_rel >= 0, is defined via table mue_pos (first column = w_rel, second column = mue). Currently, only linear interpolation in the table is supported.

When the relative angular velocity becomes zero, the elements connected by the friction element become stuck, i.e., the relative angle remains constant. In this phase the friction torque is calculated from a torque balance due to the requirement, that the relative acceleration shall be zero. The elements begin to slide when the friction torque exceeds a threshold value, called the maximum static friction torque, computed via:

```
frictional_torque = peak * cgeo * mue(w_rel=0) * fn    (peak >= 1)
```

This procedure is implemented in a "clean" way by state events and leads to continuous/discrete systems of equations if friction elements are dynamically coupled. The method is described in:

Otter M., Elmqvist H., and Mattsson S.E. (1999):
Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. CACSD'99, Aug. 22.-26, Hawaii.

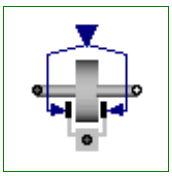
Parameters

Type	Name	Default	Description
Real	mue_pos[:, :]	[0, 0.5]	[w,mue] positive sliding friction coefficient (w_rel>=0)
Real	peak	1	peak*mue_pos[1,2] = maximum value of mue for w_rel==0
Real	cgeo	1	Geometry constant containing friction distribution assumption
Force	fn_max	1	Maximum normal force [N]
Initialization			
Angle	phi_rel.start	0	Relative rotation angle (= flange_b.phi - flange_a.phi) [rad]
Advanced			
AngularVelocity	w_small	1e10	Relative angular velocity near to zero if jumps due to a reinit(..) of the velocity can occur (set to low value only if such impulses can occur) [rad/s]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
input RealInput	f_normalized	Normalized force signal 0..1 (normal force = fn_max*f_normalized; clutch is engaged if > 0)

Modelica.Mechanics.Rotational.Brake



Brake based on Coulomb friction

Information

This component models a **brake**, i.e., a component where a frictional torque is acting between the housing and a flange and a controlled normal force presses the flange to the housing in order to increase friction. The normal force fn has to be provided as input signal f_normalized in a normalized form (0 ≤ f_normalized ≤ 1), fn = fn_max*f_normalized, where fn_max has to be provided as parameter. Friction in the brake is modelled in the following way:

When the absolute angular velocity "w" is not zero, the friction torque is a function of the velocity dependent friction coefficient mue(w) , of the normal force "fn", and of a geometry constant "cgeo" which takes into account the geometry of the device and the assumptions on the friction distributions:

```
frictional_torque = cgeo * mue(w) * fn
```

Typical values of coefficients of friction:

```
dry operation   :  mue = 0.2 .. 0.4
```

```
operating in oil:  mue = 0.05 .. 0.1
```

When plates are pressed together, where **ri** is the inner radius, **ro** is the outer radius and **N** is the number of friction interfaces, the geometry constant is calculated in the following way under the assumption of a uniform rate of wear at the interfaces:

```
cgeo = N*(r0 + ri)/2
```

The positive part of the friction characteristic **mue**(w), w >= 0, is defined via table mue_pos (first column = w, second column = mue). Currently, only linear interpolation in the table is supported.

When the absolute angular velocity becomes zero, the elements connected by the friction element become stuck, i.e., the absolute angle remains constant. In this phase the friction torque is calculated from a torque balance due to the requirement, that the absolute acceleration shall be zero. The elements begin to slide when the friction torque exceeds a threshold value, called the maximum static friction torque, computed via:

```
frictional_torque = peak * cgeo * mue(w=0) * fn    (peak >= 1)
```

This procedure is implemented in a "clean" way by state events and leads to continuous/discrete systems of equations if friction elements are dynamically coupled. The method is described in:

Otter M., Elmqvist H., and Mattsson S.E. (1999):
Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. CACSD'99, Aug. 22.-26, Hawaii.

More precise friction models take into account the elasticity of the material when the two elements are "stuck", as well as other effects, like hysteresis. This has the advantage that the friction element can be completely described by a differential equation without events. The drawback is that the system becomes stiff (about 10-20 times slower simulation) and that more material constants have to be supplied which requires more sophisticated identification. For more details, see the following references, especially (Armstrong and Canudas de Witt 1996):

Armstrong B. (1991):
Control of Machines with Friction. Kluwer Academic Press, Boston MA.

Armstrong B., and Canudas de Wit C. (1996):
Friction Modeling and Compensation. The Control Handbook, edited by W.S.Levine, CRC Press, pp. 1369-1382.

Canudas de Wit C., Olsson H., Astroem K.J., and Lischinsky P. (1995):
A new model for control of systems with friction. IEEE Transactions on Automatic Control, Vol. 40, No. 3, pp. 419-425.

Parameters

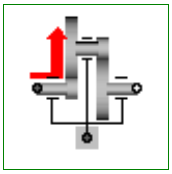
Type	Name	Default	Description
Real	mue_pos[:, :]	[0, 0.5]	[w,mue] positive sliding friction coefficient (w_rel>=0)
Real	peak	1	peak*mue_pos[1,2] = maximum value of mue for w_rel==0
Real	cgeo	1	Geometry constant containing friction distribution assumption
Force	fn_max	1	Maximum normal force [N]
Initialization			
Boolean	startForward.start	**false**	true, if w_rel=0 and start of forward sliding or w_rel > w_small
Boolean	startBackward.start	**false**	true, if w_rel=0 and start of backward sliding or w_rel < -w_small
Boolean	locked.start	false	true, if w_rel=0 and not sliding

Advanced			
AngularVelocity	w_small	1e10	Relative angular velocity near to zero if jumps due to a reinit(..) of the velocity can occur (set to low value only if such impulses can occur) [rad/s]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	
input RealInput	f_normalized	Normalized force signal 0..1 (normal force = fn_max*f_normalized; brake is active if > 0)

Modelica.Mechanics.Rotational.LossyGear



Gear with mesh efficiency and bearing friction (stuck/rolling possible)

Information

This component models the gear ratio and the **losses** of a standard gear box in a **reliable** way including the stuck phases that may occur at zero speed. The gear boxes that can be handeled are fixed in the ground, have one input and one output shaft, and are essentially described by the equations:

```
    flange_a.phi   = i*flange_b.phi
(-flange_b.tau) = i*(eta_mf*flange_a.tau - tau_bf)
```

where

- **i** is the constant **gear ratio**,
- **eta_mf** = eta_mf(w) is the **mesh efficiency** due to the friction between the teeth of the gear wheels,
- **tau_bf** = tau_bf(w) is the **bearing friction torque**, and
- **w_a** = der(flange_a.phi) is the speed of flange_a

The loss terms "eta_mf" and "tau_bf" are functions of the *absolute value* of the input shaft speed w_a and of the energy flow direction. They are defined by parameter **lossTable[:,5]** where the columns of this table have the following meaning:

|w_a|	eta_mf1	eta_mf2	|tau_bf1|	|tau_bf2|
...	...	...	...	...
...	...	...	...	...

with

|w_a|	Absolute value of angular velocity of input shaft flange_a
eta_mf1	Mesh efficiency in case of input shaft driving
eta_mf2	Mesh efficiency in case of output shaft driving
|tau_bf1|	Absolute bearing friction torque in case of input shaft driving
|tau_bf2|	Absolute bearing friction torque in case of output shaft driving

With these variables, the mesh efficiency and the bearing friction are formally defined as:

```
if flange_a.tau*w_a > 0 or flange_a.tau==0 and w_a > 0 then
   eta_mf := eta_mf1
   tau_bf := tau_bf1
elseif flange_a.tau*w_a < 0 or flange_a.tau==0 and w_a < 0 then
   eta_mf := 1/eta_mf2
   tau_bf := tau_bf2
```

```
else // w_a == 0
   eta_mf and tau_bf are computed such that der(w_a) = 0
end if;
```

Note, that the losses are modeled in a physically meaningful way taking into account that at zero speed the movement may be locked due to the friction in the gear teeth and/or in the bearings. Due to this important property, this component can be used in situations where the combination of the components Modelica.Mechanics.Rotational.IdealGear and Modelica.Mechanics.Rotational.GearEfficiency will fail because, e.g., chattering occurs when using the Modelica.Mechanics.Rotational.GearEfficiency model.

Acknowledgement: The essential idea to model efficiency in this way is from Christoph Pelchen, ZF Friedrichshafen.

For detailed information:

Pelchen C., Schweiger C., and Otter M.: "Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes," in *Proceedings of the 2nd International Modelica Conference, Oberpfaffenhofen, Germany,* pp. 257-266, The Modelica Association and Institute of Robotics and Mechatronics, Deutsches Zentrum für Luft- und Raumfahrt e. V., March 18-19, 2002.

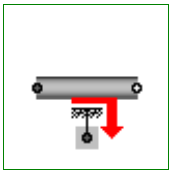
Parameters

Type	Name	Default	Description
Real	i	1	Transmission ratio (flange_a.phi/flange_b.phi)
Real	lossTable[:, 5]	[0, 1, 1, 0, 0]	Array for mesh efficiencies and bearing friction depending on speed

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.GearEfficiency



Obsolete component (use model LossyGear instead)

Information

THIS COMPONENT IS **OBSOLETE** and should **no longer be used**. It is only kept for **backward compatibility** purposes. Use model Modelica.Mechanics.Rotational.LossyGear instead which implements gear efficiency in a much more reliable way.

This component consists of two rigidly connected flanges flange_a and flange_b without inertia where an **efficency** coefficient **eta** reduces the driven torque as function of the driving torque depending on the direction of the energy flow, i.e., energy is always lost. This can be seen as a simple model of the Coulomb friction acting between the teeth of a gearbox.

Note, that most gearbox manufacturers provide tables of the efficiency of a gearbox as function of the angular velocity (efficiency becomes zero, if the angular velocity is zero). However, such a table is practically useless for simulation purposes, because in gearboxes always two types of friction is present: (1) Friction in the **bearings** and (2) friction between the teeth of the gear. (1) leads to a velocity dependent, additive loss-torque, whereas (2) leads to a torque-dependent reduction of the driving torque. The gearbox manufacturers measure both effects together and determine the gear efficiency from it, although for simulation purposes the two effects need to be separated. Assume for example that only constant bearing friction, i.e., bearingTorque=const., is present, i.e.,

```
(1)  loadTorque = motorTorque - sign(w)*bearingTorque
```

Gearbox manufacturers use the loss-formula

```
(2)  loadTorque = eta*motorTorque
```

Comparing (1) and (2) gives a formulat for the efficiency eta:

```
eta = (1 - sign(w)*bearingTorque/motorTorque)
```

When the motorTorque becomes smaller as the bearingTorque, (2) is useless, because the efficiency is zero. To summarize, be careful to determine the gear **efficiency** of this element from tables of the gear manufacturers.

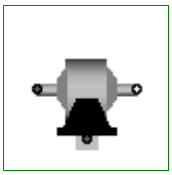
Parameters

Type	Name	Default	Description
Real	eta	1	Efficiency

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.Gear



Realistic model of a gearbox

Information

This component models the essential effects of a gearbox, in particular gear **efficiency** due to friction between the teeth, **bearing friction**, gear **elasticity** and **damping**, **backlash**. The inertia of the gear wheels is not modeled. If necessary, inertia has to be taken into account by connecting components of model Inertia to the left and/or the right flange.

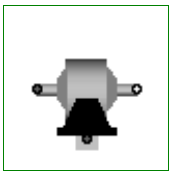
Parameters

Type	Name	Default	Description
Real	ratio	1	transmission ratio (flange_a.phi/flange_b.phi)
Real	eta	1	Gear efficiency
Real	friction_pos[:, :]	[0, 1]	[w,tau] positive sliding friction characteristic (w>=0)
Real	peak	1	peak*friction_pos[1,2] = maximum friction torque at zero velocity
Real	c	1.e5	Gear elasticity (spring constant) [N.m/rad]
Real	d	0	(relative) gear damping [N.m.s/rad]
Angle	b	0	Total backlash [rad]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.Gear2



Realistic model of a gearbox (based on LossyGear)

Information

This component models the essential effects of a gearbox, in particular

- in component **lossyGear**
  - gear **efficiency** due to friction between the teeth
  - **bearing friction**
- in component **elastoBacklash**
  - gear **elasticity**
  - **damping**
  - **backlash**

The inertia of the gear wheels is not modeled. If necessary, inertia has to be taken into account by connecting components of model Inertia to the left and/or the right flange of component GearNew.

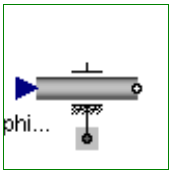
Parameters

Type	Name	Default	Description
Real	i	1	transmission ratio (flange_a.phi/flange_b.phi)
Real	lossTable[:, 5]	[0, 1, 1, 0, 0]	Array for mesh efficiencies and bearing friction depending on speed (see docu of LossyGear)
Real	c	1.e5	Gear elasticity (spring constant) [N.m/rad]
Real	d	0	(relative) gear damping [N.m.s/rad]
Angle	b	0	Total backlash [rad]

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.Position



Forced movement of a flange according to a reference angle signal

Information

The input signal **phi_ref** defines the **reference angle** in [rad]. Flange **flange_b** is **forced** to move according to this reference motion. According to parameter **exact** (default = **false**), this is done in the following way:

1. **exact=true**
   The reference angle is treated **exactly**. This is only possible, if the input signal is defined by an analytical function which can be differentiated at least twice. If this prerequisite is fulfilled, the Modelica translator will differentiate the input signal twice in order to compute the reference acceleration of the flange.
2. **exact=false**
   The reference angle is **filtered** and the second derivative of the filtered curve is used to compute the reference acceleration of the flange. This second derivative is **not** computed by numerical differentiation but by an appropriate realization of the filter. For filtering, a second order Bessel filter

is used. The critical frequency (also called cut-off frequency) of the filter is defined via parameter **f_crit** in [Hz]. This value should be selected in such a way that it is higher as the essential low frequencies in the signal.

The input signal can be provided from one of the signal generator blocks of the block library Modelica.Blocks.Sources.

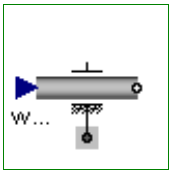
Parameters

Type	Name	Default	Description
Boolean	exact	false	true/false exact treatment/filtering the input signal
Frequency	f_crit	50	if exact=false, critical frequency of filter to filter input signal [Hz]

Connectors

Type	Name	Description
input RealInput	phi_ref	reference angle of flange_b as input signal
Flange_b	flange_b	
Flange_a	bearing	

Modelica.Mechanics.Rotational.Speed



Forced movement of a flange according to a reference angular velocity signal

Information

The input signal **w_ref** defines the **reference angle** in [rad]. Flange **flange_b** is **forced** to move according to this reference motion. According to parameter **exact** (default = **false**), this is done in the following way:

1. **exact=true**
   The reference angle is treated **exactly**. This is only possible, if the input signal is defined by an analytical function which can be differentiated at least twice. If this prerequisite is fulfilled, the Modelica translator will differentiate the input signal twice in order to compute the reference acceleration of the flange.
2. **exact=false**
   The reference angle is **filtered** and the second derivative of the filtered curve is used to compute the reference acceleration of the flange. This second derivative is **not** computed by numerical differentiation but by an appropriate realization of the filter. For filtering, a second order Bessel filter is used. The critical frequency (also called cut-off frequency) of the filter is defined via parameter **f_crit** in [Hz]. This value should be selected in such a way that it is higher as the essential low frequencies in the signal.

The input signal can be provided from one of the signal generator blocks of the block library Modelica.Blocks.Sources.

Release Notes:

- *October 27, 2003* by Christian Schweiger.
  Realized based on component `Position` (implemented by Martin Otter).

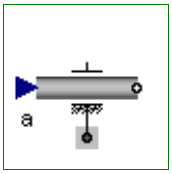
Parameters

Type	Name	Default	Description
Boolean	exact	false	true/false exact treatment/filtering the input signal
Frequency	f_crit	50	if exact=false, critical frequency of filter to filter input signal [Hz]

Connectors

Type	Name	Description
Flange_b	flange_b	
input RealInput	w_ref	Reference angular velocity of flange_b as input signal
Flange_a	bearing	

Modelica.Mechanics.Rotational.Accelerate



Forced movement of a flange according to an acceleration signal

Information

The input signal **a** defines an **angular acceleration** in [rad/s2]. Flange **flange_b** is **forced** to move with this acceleration. The angular velocity **w** and the rotation angle **phi** of the flange are automatically determined by integration of the acceleration.

The input signal can be provided from one of the signal generator blocks of the block library Modelica.Blocks.Sources.

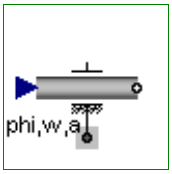
Parameters

Type	Name	Default	Description
Angle	phi_start	0	Start angle [rad]
AngularVelocity	w_start	0	Start angular velocity [rad/s]

Connectors

Type	Name	Description
Flange_b	flange_b	
input RealInput	a	absolute angular acceleration of flange_b as input signal
Flange_a	bearing	

Modelica.Mechanics.Rotational.Move



Forced movement of a flange according to an angle, speed and angular acceleration signal

Information

Flange **flange_b** is **forced** to move with a predefined motion according to the input signals:

```
u[1]: angle of flange
u[2]: angular velocity of flange
u[3]: angular acceleration of flange
```

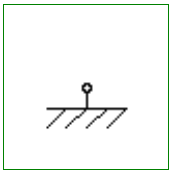
The user has to guarantee that the input signals are consistent to each other, i.e., that u[2] is the derivative of u[1] and that u[3] is the derivative of u[2]. There are, however, also applications where by purpose these conditions do not hold. For example, if only the position dependent terms of a mechanical system shall be calculated, one may provide angle = angle(t) and set the angular velocity and the angular acceleration to zero.

The input signals can be provided from one of the signal generator blocks of the block library Modelica.Blocks.Sources.

Connectors

Type	Name	Description
input RealInput	u[3]	angle, angular velocity and angular acceleration of flange_b as input signals
Flange_b	flange_b	Flange that is forced to move according to input signals u
Flange_a	bearing	Bearing flange (if not connected, it is assumed that it is fixed on ground)

Modelica.Mechanics.Rotational.Fixed



Flange fixed in housing at a given angle

Information

The **flange** of a 1D rotational mechanical system is **fixed** at an angle phi0 in the **housing**. May be used:

- to connect a compliant element, such as a spring or a damper, between an inertia or gearbox component and the housing.
- to fix a rigid element, such as an inertia, with a specific angle to the housing.

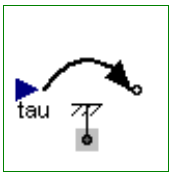
Parameters

Type	Name	Default	Description
Angle	phi0	0	Fixed offset angle of housing [rad]

Connectors

Type	Name	Description
Flange_b	flange_b	(right) flange fixed in housing

Modelica.Mechanics.Rotational.Torque



Input signal acting as external torque on a flange

Information

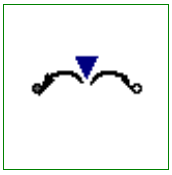
The input signal **tau** defines an external torque in [Nm] which acts (with negative sign) at a flange connector, i.e., the component connected to this flange is driven by torque **tau**.

The input signal can be provided from one of the signal generator blocks of Modelica.Blocks.Sources.

Connectors

Type	Name	Description
input RealInput	tau	Torque driving the flange (a positive value accelerates the flange)
Flange_b	flange_b	(Right) flange
Flange_a	bearing	

Modelica.Mechanics.Rotational.Torque2



Input signal acting as torque on two flanges

Information

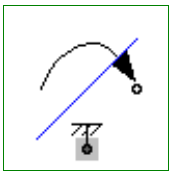
The input signal **tau** defines an external torque in [Nm] which acts at both flange connectors, i.e., the components connected to these flanges are driven by torque **tau**.

The input signal can be provided from one of the signal generator blocks of Modelica.Blocks.Sources.

Connectors

Type	Name	Description
Flange_a	flange_a	
Flange_b	flange_b	
input RealInput	tau	Torque driving the two flanges (a positive value accelerates the flange)

Modelica.Mechanics.Rotational.LinearSpeedDependentTorque



Linear dependency of torque versus speed

Information

Model of torque, linearly dependent on angular velocity of flange.
Parameter TorqueDirection chooses whether direction of torque is the same in both directions of rotation or not.

Parameters

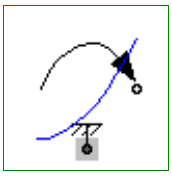
Type	Name	Default	Description
Torque	tau_nominal		nominal torque (if negative, torque is acting as load) [N.m]
Boolean	TorqueDirection	true	same direction of torque in both directions of rotation
AngularVelocity	w_nominal		nominal speed [rad/s]

Connectors

Type	Name	Description
Flange_b	flange	Flange on which torque is acting
Flange_a	bearing	Bearing at which the reaction torque (i.e., -flange.tau) is acting

---

Modelica.Mechanics.Rotational.QuadraticSpeedDependentTorque



Quadratic dependency of torque versus speed

Information

Model of torque, quadratic dependent on angular velocity of flange.
Parameter TorqueDirection chooses whether direction of torque is the same in both directions of rotation or not.

Parameters

Type	Name	Default	Description
Torque	tau_nominal		nominal torque (if negative, torque is acting as load) [N.m]
Boolean	TorqueDirection	true	same direction of torque in both directions of rotation

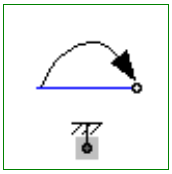
AngularVelocity	w_nominal		nominal speed [rad/s]

Connectors

Type	Name	Description
Flange_b	flange	Flange on which torque is acting
Flange_a	bearing	Bearing at which the reaction torque (i.e., -flange.tau) is acting

---

Modelica.Mechanics.Rotational.ConstantTorque



Constant torque, not dependent on speed

Information

Model of constant torque, not dependent on angular velocity of flange.
Positive torque acts accelerating.

Parameters

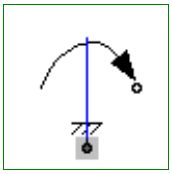
Type	Name	Default	Description
Torque	tau_constant		constant torque (if negative, torque is acting as load) [N.m]

Connectors

Type	Name	Description
Flange_b	flange	Flange on which torque is acting
Flange_a	bearing	Bearing at which the reaction torque (i.e., -flange.tau) is acting

---

Modelica.Mechanics.Rotational.ConstantSpeed



Constant speed, not dependent on torque

Information

Model of **fixed** angular verlocity of flange, not dependent on torque.

Parameters

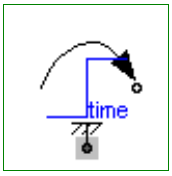
Type	Name	Default	Description
AngularVelocity	w_fixed		fixed speed (if negative, torque is acting as load) [rad/s]

Connectors

Type	Name	Description
Flange_b	flange	Flange on which torque is acting
Flange_a	bearing	Bearing at which the reaction torque (i.e., -flange.tau) is acting

---

Modelica.Mechanics.Rotational.TorqueStep



Constant torque, not dependent on speed

Information

Model of a torque step at time .
Positive torque acts accelerating.

Parameters

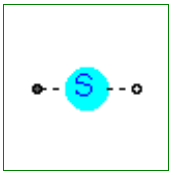
Type	Name	Default	Description
Torque	stepTorque	1	height of torque step (if negative, torque is acting as load) [N.m]
Torque	offsetTorque	0	offset of torque [N.m]
Time	startTime	0	output = offset for time < startTime [s]

Connectors

Type	Name	Description
Flange_b	flange	Flange on which torque is acting
Flange_a	bearing	Bearing at which the reaction torque (i.e., -flange.tau) is acting

---

Modelica.Mechanics.Rotational.RelativeStates



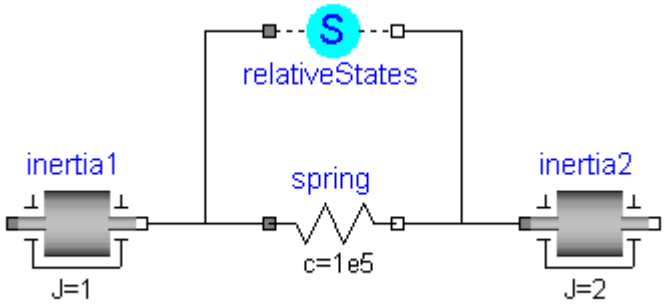
Definition of relative state variables

Information

Usually, the absolute angle and the absolute angular velocity of Modelica.Mechanics.Rotational.Inertia models are used as state variables. In some circumstances, relative quantities are better suited, e.g., because it may be easier to supply initial values. In such cases, model **RelativeStates** allows the definition of state variables in the following way:

- Connect an instance of this model between two flange connectors.
- The **relative rotation angle** and the **relative angular velocity** between the two connectors are used as **state variables**.

An example is given in the next figure



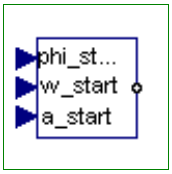
Here, the relative angle and the relative angular velocity between the two inertias are used as state variables. Additionally, the simulator selects either the absolute angle and absolute angular velocity of model inertia1 or of model inertia2 as state variables.

Connectors

Type	Name	Description
Flange_a	flange_a	

Flange_b	flange_b	

Modelica.Mechanics.Rotational.InitializeFlange



Initializes a flange with pre-defined angle, speed and angular acceleration (usually, this is reference data from a control bus)

Information

This component is used to optionally initialize the angle, speed, and/or angular acceleration of the flange to which this component is connected. Via parameters use_phi_start, use_w_start, use_a_start the corresponding input signals phi_start, w_start, a_start are conditionally activated. If an input is activated, the corresponding flange property is initialized with the input value at start time.

For example, if "use_phi_start = true", then flange.phi is inialized with the value of the input signal "phi_start" at the start time.

Additionally, it is optionally possible to define the "StateSelect" attribute of the flange angle and the flange speed via paramater "stateSelection".

This component is especially useful when the initial values of a flange shall be set according to reference signals of a controller that are provided via a signal bus.

Parameters

Type	Name	Default	Description
Boolean	use_phi_start	true	= true, if initial angle is defined by input phi_start, otherwise not initialized
Boolean	use_w_start	true	= true, if initial speed is defined by input w_start, otherwise not initialized
Boolean	use_a_start	true	= true, if initial angular acceleration is defined by input a_start, otherwise not initialized
Temp	stateSelection	Modelica.Blocks.Types.StateS..	Priority to use flange angle and speed as states

Connectors

Type	Name	Description
input RealInput	phi_start	Initial angle of flange
input RealInput	w_start	Initial speed of flange
input RealInput	a_start	Initial angular acceleration of flange
Flange_b	flange	Flange that is initialized

Modelica.Mechanics.Rotational.Types

Constants and types with choices, especially to build menus

Information

In this package **types** and **constants** are defined that are used in library Modelica.Blocks. The types have additional annotation choices definitions that define the menus to be built up in the graphical user interface when the type is used as parameter in a declaration.

Package Content

Name	Description
Init	Type, constants and menu choices to define initialization of absolute rotational quantities
InitRel	Type, constants and menu choices to define initialization of relative rotational quantities

---

Modelica.Mechanics.Rotational.Types.Init

Type, constants and menu choices to define initialization of absolute rotational quantities

Information

Type **Init** defines initialization of absolute rotational quantities.

Package Content

Name	Description
NoInit=1	no initialization (phi_start, w_start are guess values)
SteadyState=2	steady state initialization (der(phi)=der(w)=0)
InitialState=3	initialization with phi_start, w_start
InitialAngle=4	initialization with phi_start
InitialSpeed=5	initialization with w_start
InitialAcceleration=6	initialization with a_start
InitialAngleAcceleration=7	initialization with phi_start, a_start
InitialSpeedAcceleration=8	initialization with w_start, a_start
InitialAngleSpeedAcceleration=9	initialization with phi_start, w_start, a_start
Temp	Temporary type of absolute initialization with choices for menus (until enumerations are available)

Types and constants

```
constant Integer NoInit=1
"no initialization (phi_start, w_start are guess values)";

constant Integer SteadyState=2
"steady state initialization (der(phi)=der(w)=0)";

constant Integer InitialState=3 "initialization with phi_start, w_start";

constant Integer InitialAngle=4 "initialization with phi_start";

constant Integer InitialSpeed=5 "initialization with w_start";

constant Integer InitialAcceleration=6 "initialization with a_start";

constant Integer InitialAngleAcceleration=7
"initialization with phi_start, a_start";

constant Integer InitialSpeedAcceleration=8
"initialization with w_start, a_start";
```

```
  constant Integer InitialAngleSpeedAcceleration=9
  "initialization with phi_start, w_start, a_start";

  type Temp
  "Temporary type of absolute initialization with choices for menus (until
enumerations are available)"
    extends Modelica.Icons.TypeInteger(min=1,max=9);

  end Temp;
```

Modelica.Mechanics.Rotational.Types.InitRel

Type, constants and menu choices to define initialization of relative rotational quantities

Information

Type **Init** defines initialization of relative rotational quantities.

Package Content

Name	Description
NoInit=1	no initialization (phi_rel_start, w_rel_start are guess values)
SteadyState=2	steady state initialization (der(phi_rel)=der(w_rel)=0)
InitialState=3	initialization with phi_rel_start, w_rel_start
InitialAngle=4	initialization with phi_rel_start
InitialSpeed=5	initialization with w_rel_start
Temp	Temporary type of absolute initialization with choices for menus (until enumerations are available)

Types and constants

```
  constant Integer NoInit=1
  "no initialization (phi_rel_start, w_rel_start are guess values)";

  constant Integer SteadyState=2
  "steady state initialization (der(phi_rel)=der(w_rel)=0)";

  constant Integer InitialState=3
  "initialization with phi_rel_start, w_rel_start";

  constant Integer InitialAngle=4 "initialization with phi_rel_start";

  constant Integer InitialSpeed=5 "initialization with w_rel_start";

  type Temp
  "Temporary type of absolute initialization with choices for menus (until
enumerations are available)"
    extends Modelica.Icons.TypeInteger(min=1,max=5);

  end Temp;
```

Modelica.Mechanics.Translational

Library to model 1-dimensional, translational mechanical systems

Information

This package contains components to model *1-dimensional translational mechanical* systems.

The *filled* and *non-filled green squares* at the left and right side of a component represent *mechanical flanges*. Drawing a line between such squares means that the corresponding flanges are *rigidly attached* to each other. The components of this library can be usually connected together in an arbitrary way. E.g. it is possible to connect two springs or two sliding masses with inertia directly together.

The only *connection restriction* is that the Coulomb friction elements (Stop) should be only connected together provided a compliant element, such as a spring, is in between. The reason is that otherwise the frictional force is not uniquely defined if the elements are stuck at the same time instant (i.e., there does not exist a unique solution) and some simulation systems may not be able to handle this situation, since this leads to a singularity during simulation. It can only be resolved in a "clean way" by combining the two connected friction elements into one component and resolving the ambiguity of the frictional force in the stuck mode.

Another restriction arises if the hard stops in model Stop are used, i. e. the movement of the mass is limited by a stop at smax or smin. **This requires the states Stop.s and Stop.v** . If these states are eliminated during the index reduction the model will not work. To avoid this any inertias should be connected via springs to the Stop element, other sliding masses, dampers or hydraulic chambers must be avoided.

In the *icon* of every component an *arrow* is displayed in grey color. This arrow characterizes the coordinate system in which the vectors of the component are resolved. It is directed into the positive translational direction (in the mathematical sense). In the flanges of a component, a coordinate system is rigidly attached to the flange. It is called *flange frame* and is directed in parallel to the component coordinate system. As a result, e.g., the positive cut-force of a "left" flange (flange_a) is directed into the flange, whereas the positive cut-force of a "right" flange (flange_b) is directed out of the flange. A flange is described by a Modelica connector containing the following variables:

```
SIunits.Position s  "absolute position of flange";
flow Force f        "cut-force in the flange";
```

This library is designed in a fully object oriented way in order that components can be connected together in every meaningful combination (e.g. direct connection of two springs or two shafts with inertia). As a consequence, most models lead to a system of differential-algebraic equations of *index 3* (= constraint equations have to be differentiated twice in order to arrive at a state space representation) and the Modelica translator or the simulator has to cope with this system representation. According to our present knowledge, this requires that the Modelica translator is able to symbolically differentiate equations (otherwise it is e.g. not possible to provide consistent initial conditions; even if consistent initial conditions are present, most numerical DAE integrators can cope at most with index 2 DAEs).

Main Author:
Peter Beater
Universität Paderborn, Abteilung Soest
Fachbereich Maschinenbau/Automatisierungstechnik
Lübecker Ring 2
D 59494 Soest
Germany
email: Beater@mailso.uni-paderborn.de

Copyright © 1998-2007, Modelica Association and Universität Paderborn, FB 12.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Examples	Demonstration examples of the components of this package
Sensors	Sensors for 1-dim. translational mechanical quantities
Interfaces	Interfaces for 1-dim. translational mechanical components
SlidingMass	Sliding mass with inertia
Stop	Sliding mass with hard stop and Stribeck friction
Rod	Rod without inertia
Spring	Linear 1D translational spring
Damper	Linear 1D translational damper
SpringDamper	Linear 1D translational spring and damper in parallel
ElastoGap	1D translational spring damper combination with gap
Position	Forced movement of a flange according to a reference position
Speed	Forced movement of a flange according to a reference speed
Accelerate	Forced movement of a flange according to an acceleration signal
Move	Forced movement of a flange according to a position, velocity and acceleration signal
Fixed	Fixed flange
Force	External force acting on a drive train element as input signal
RelativeStates	Definition of relative state variables

Modelica.Mechanics.Translational.Examples

Demonstration examples of the components of this package

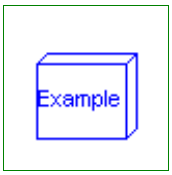
Information

This package contains example models to demonstrate the usage of the Translational package. Open the models and simulate them according to the provided description in the models.

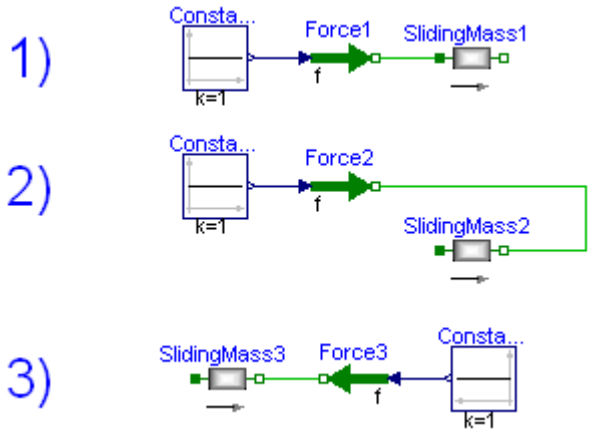
Package Content

Name	Description
SignConvention	Examples for the used sign conventions.
InitialConditions	Setting of initial conditions
WhyArrows	Use of arrows in Mechanics.Translational
Accelerate	Use of model accelerate.
Damper	Use of damper models.
Oscillator	Oscillator demonstrates the use of initial conditions.
Sensors	Sensors for translational systems.
Friction	Use of model Stop
PreLoad	Preload of a spool using ElastoGap models.

Modelica.Mechanics.Translational.Examples.SignConvention



Examples for the used sign conventions.



Information

If all arrows point in the same direction a positive force results in a positive acceleration a, velocity v and position s.

For a force of 1 N and a mass of 1 Kg this leads to

```
a = 1 m/s2
v = 1 m/s after 1 s (SlidingMass1.v)
s = 0.5 m after 1 s (SlidingMass1.s)
```

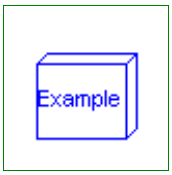
The acceleration is not available for plotting.

System 1) and 2) are equivalent. It doesn't matter whether the force pushes at flange_a in system 1 or pulls at flange_b in system 2.

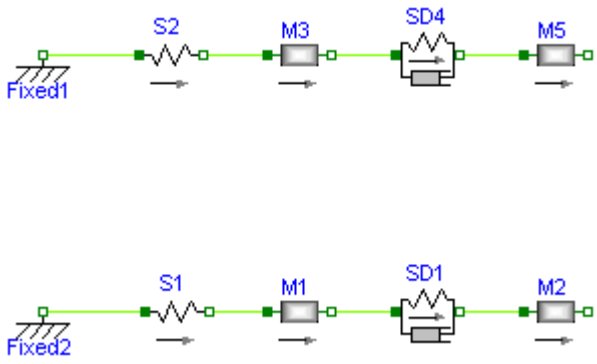
It is of course possible to ignore the arrows and connect the models in an arbitrary way. But then it is hard see in what direction the force acts.

In the third system the two arrows are opposed which means that the force acts in the opposite direction (in the same direction as in the two other examples).

Modelica.Mechanics.Translational.Examples.InitialConditions



Setting of initial conditions

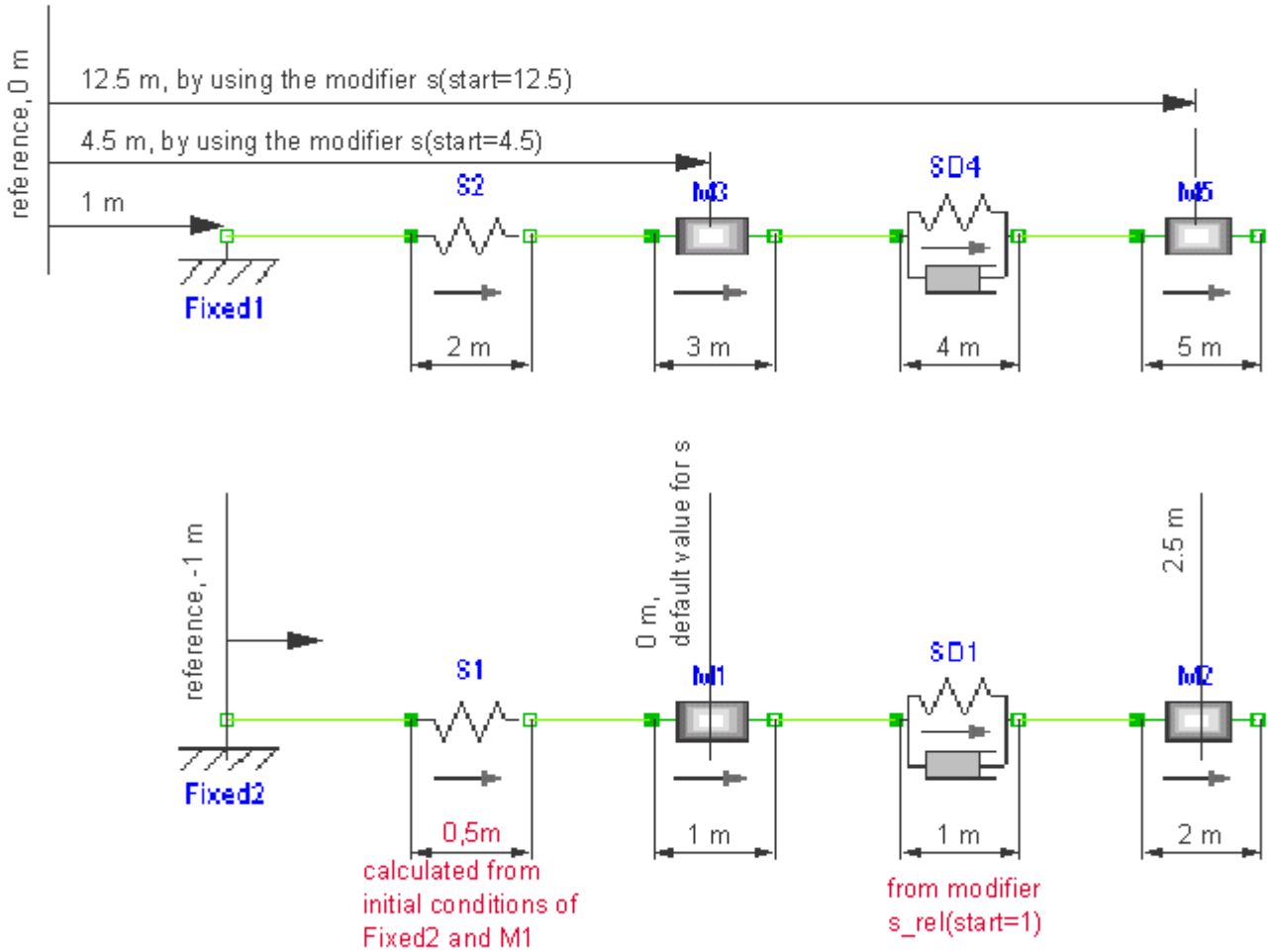


Information

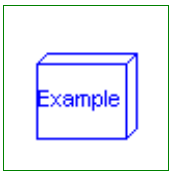
There are several ways to set initial conditions. In the first system the position of the sliding mass m3 was defined by using the modifier s(start=4.5), the position of m5 by s(start=12.5). These positions were chosen such that the system is a rest. To calculate these values start at the left (Fixed1) with a value of 1 m. The spring has an unstreched length of 2 m and m3 an length of 3 m, which leads to

```
  1   m (Fixed1)
+ 2   m (Spring S2)
+ 3/2 m (half of the length of SlidingMass m3)
-------
  4,5 m = s(start = 4.5) for m3
+ 3/2 m (half of the length of SlidingMass m3)
+ 4   m (SpringDamper 4
+ 5/2 m (half of length of SlidingMass m5)
-------
 12,5 m = s(start = 12.5) for m5
```

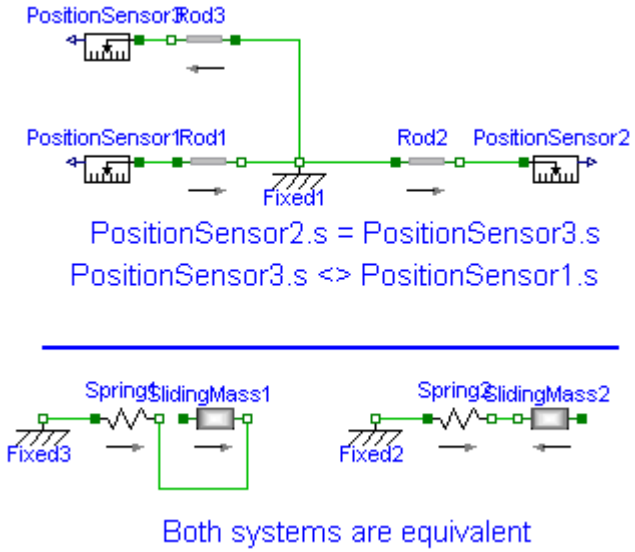
This selection of initial conditions has the effect that Dymola selects those variables (m3.s and m5.s) as state variables. In the second example the length of the springs are given as start values but they cannot be used as state for pure springs (only for the spring/damper combination). In this case the system is not at rest.



Modelica.Mechanics.Translational.Examples.WhyArrows



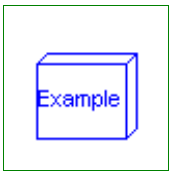
Use of arrows in Mechanics.Translational



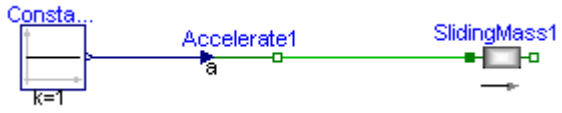
Information

When using the models of the translational sublibrary it is recommended to make sure that all arrows point in the same direction because then all component have the same reference system. In the example the distance from flange_a of Rod1 to flange_b of Rod2 is 2 m. The distance from flange_a of Rad1 to flange_b of Rod3 is also 2 m though it is difficult to see that. Without the arrows it would be almost impossible to notice. That all arrows point in the same direction is a sufficient condition for an easy use of the library. There are cases where horizontally flipped models can be used without problems.

Modelica.Mechanics.Translational.Examples.Accelerate

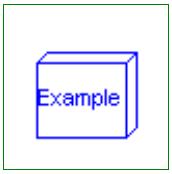


Use of model accelerate.

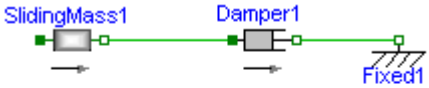


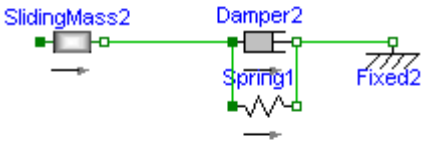
Information

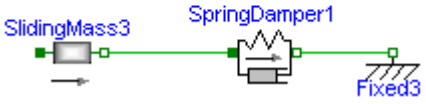
Modelica.Mechanics.Translational.Examples.Damper



Use of damper models.

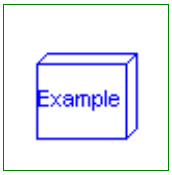




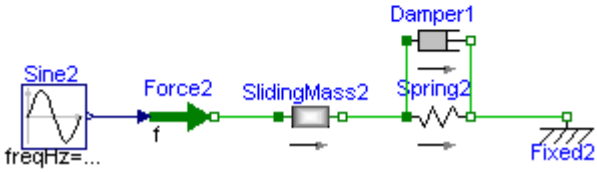


Information

Modelica.Mechanics.Translational.Examples.Oscillator



Oscillator demonstrates the use of initial conditions.



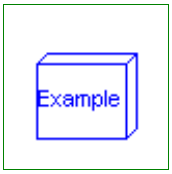
Information

A spring - mass system is a mechanical oscillator. If no damping is included and the system is excited at resonance frequency infinite amplitudes will result. The resonant frequency is given by omega_res = sqrt(c / m) with:

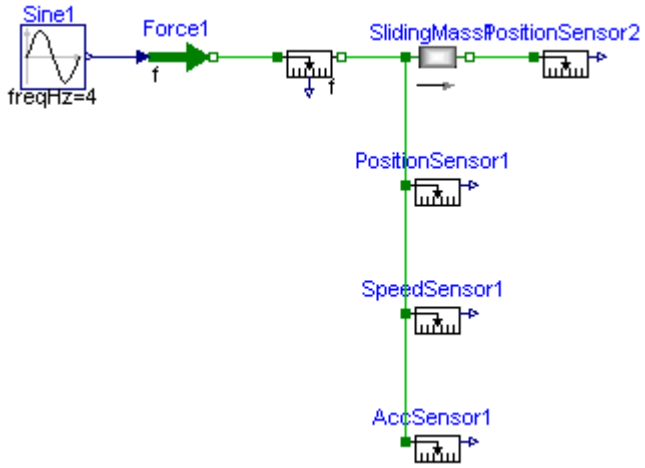
```
c spring stiffness
m mass
```

To make sure that the system is initially at rest the initial conditions s(start=0) and v(start=0) for the SlindingMass are set. If damping is added the amplitudes are bounded.

Modelica.Mechanics.Translational.Examples.Sensors



Sensors for translational systems.



Information

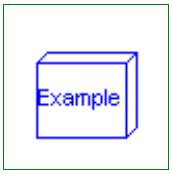
These sensors measure

```
force f in N
position s in m
velocity v in m/s
acceleration a in m/s2
```

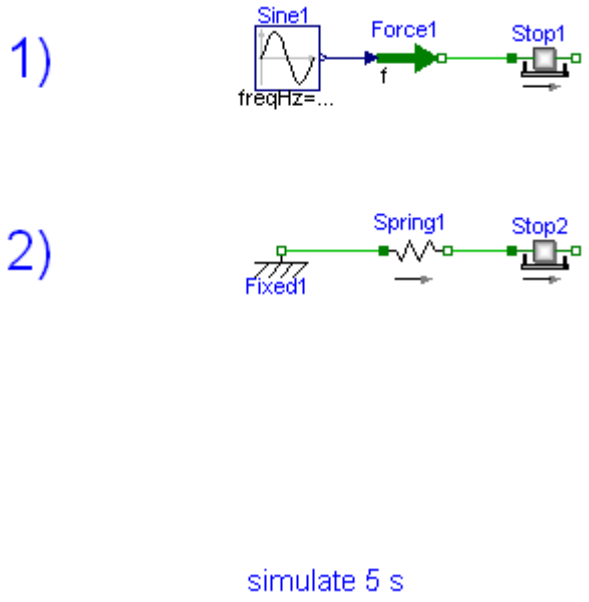
Dhe measured velocity and acceleration is independent on the flange the sensor is connected to. The position depends on the flange (flange_a or flange_b) and the length L of the component. Plot PositionSensor1.s, PositionSensor2.s and SlidingMass1.s to see the difference.

Error:Found no end-tag in HTML-documentation

Modelica.Mechanics.Translational.Examples.Friction



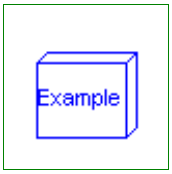
Use of model Stop



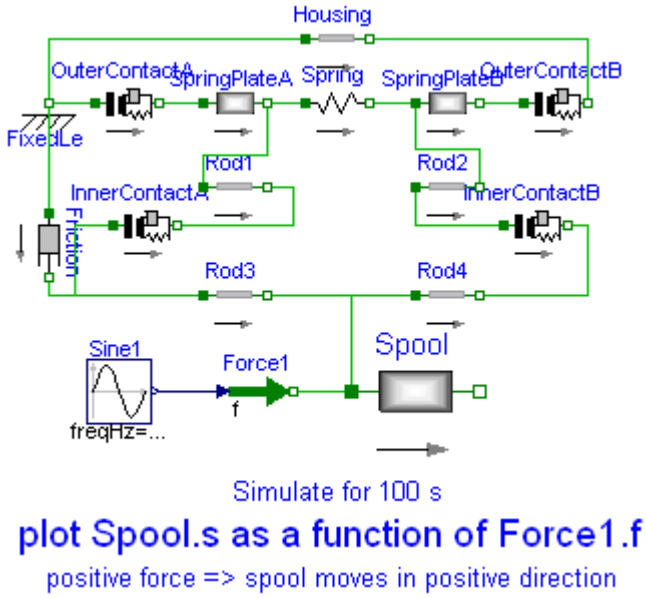
Information

1. Simulate and then plot Stop1.f as a function of Stop1.v This gives the Stribeck curve.
2. This model gives an example for a hard stop. However there can arise some problems with the used modeling approach (use of Reinit, convergence problems). In this case use the ElastoGap to model a stop (see example Preload).

Modelica.Mechanics.Translational.Examples.PreLoad



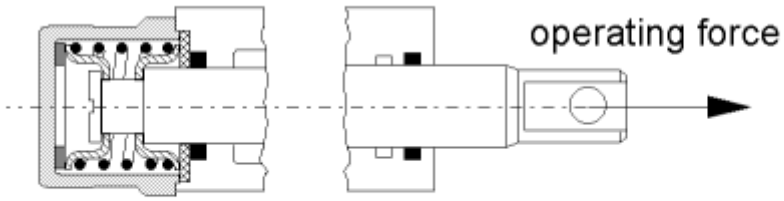
Preload of a spool using ElastoGap models.



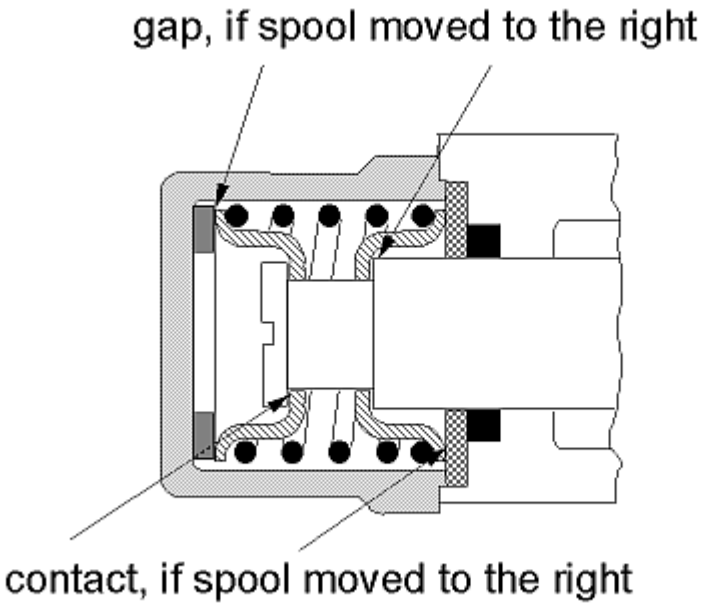
Information

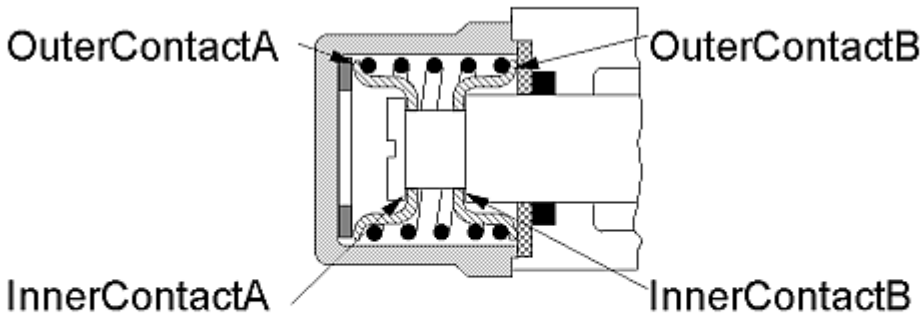
When designing hydraulic valves it is often necessary to hold the spool in a certain position as long as an

external force is below a threshold value. If this force exceeds the treshold value a linear relation between force and position is desired. There are designs that need only one spring to accomplish this task. Using the ElastoGap elements this design can be modelled easily. Drawing of spool.



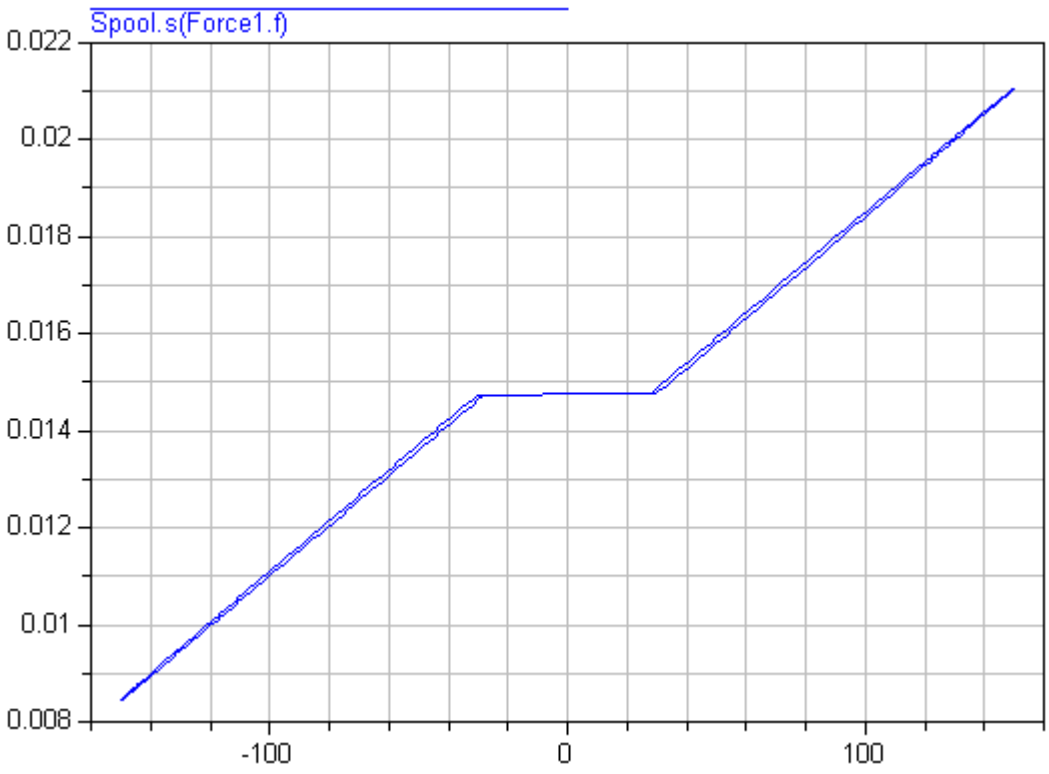
<





contact, if spool moved to the right

Spool position s as a function of working force f.



Modelica.Mechanics.Translational.Sensors

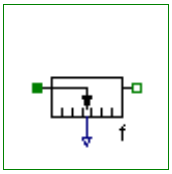
Sensors for 1-dim. translational mechanical quantities

Information

Package Content

Name	Description
ForceSensor	Ideal sensor to measure the force between two flanges
PositionSensor	Ideal sensor to measure the absolute position
SpeedSensor	Ideal sensor to measure the absolute velocity
AccSensor	Ideal sensor to measure the absolute acceleration

Modelica.Mechanics.Translational.Sensors.ForceSensor



Ideal sensor to measure the force between two flanges

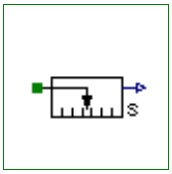
Information

Measures the *cut-force between two flanges* in an ideal way and provides the result as output signal (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
output RealOutput	f	force in flange_a and flange_b (f = flange_a.f = -flange_b.f)

Modelica.Mechanics.Translational.Sensors.PositionSensor



Ideal sensor to measure the absolute position

Information

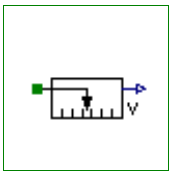
Measures the *absolute position s* of a flange in an ideal way and provides the result as output signals (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	flange to be measured (flange axis directed INTO cut plane, e. g. from left to right)
output RealOutput	s	Absolute position of flange as output signal

---

Modelica.Mechanics.Translational.Sensors.SpeedSensor



Ideal sensor to measure the absolute velocity

Information

Measures the *absolute velocity v* of a flange in an ideal way and provides the result as output signals (to be

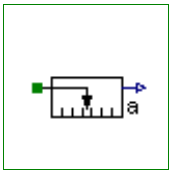
further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	flange to be measured (flange axis directed INTO cut plane, e. g. from left to right)
output RealOutput	v	Absolute velocity of flange as output signal

---

Modelica.Mechanics.Translational.Sensors.AccSensor



Ideal sensor to measure the absolute acceleration

Information

Measures the *absolute acceleration a* of a flange in an ideal way and provides the result as output signals (to be further processed with blocks of the Modelica.Blocks library).

Connectors

Type	Name	Description
Flange_a	flange_a	flange to be measured (flange axis directed INTO cut plane, e. g. from left to right)
output RealOutput	a	Absolute acceleration of flange as output signal

---

Modelica.Mechanics.Translational.Interfaces

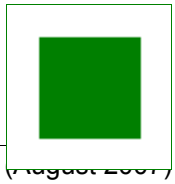
Interfaces for 1-dim. translational mechanical components

Information

Package Content

Name	Description
Flange_a	(left) 1D translational flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	right 1D translational flange (flange axis directed OUT OF cut plane)
Rigid	Rigid connection of two translational 1D flanges
Compliant	Compliant connection of two translational 1D flanges
TwoFlanges	Component with two translational 1D flanges
AbsoluteSensor	Device to measure a single absolute flange variable
RelativeSensor	Device to measure a single relative variable between two flanges
FrictionBase	Base class of Coulomb friction elements

Modelica.Mechanics.Translational.Interfaces.Flange_a



(left) 1D translational flange (flange axis directed INTO cut plane, e. g. from left to right)

Information

This is a flange for 1D translational mechanical systems. In the cut plane of the flange a unit vector n, called flange axis, is defined which is directed INTO the cut plane, i. e. from left to right. All vectors in the cut plane are resolved with respect to this unit vector. E.g. force f characterizes a vector which is directed in the direction of n with value equal to f. When this flange is connected to other 1D translational flanges, this means that the axes vectors of the connected flanges are identical.

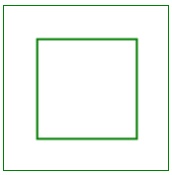
The following variables are transported through this connector:

```
s: Absolute position of the flange in [m]. A positive translation
   means that the flange is translated along the flange axis.
f: Cut-force in direction of the flange axis in [N].
```

Contents

Type	Name	Description
Position	s	absolute position of flange [m]
flow Force	f	cut force directed into flange [N]

Modelica.Mechanics.Translational.Interfaces.Flange_b



right 1D translational flange (flange axis directed OUT OF cut plane)

Information

This is a flange for 1D translational mechanical systems. In the cut plane of the flange a unit vector n, called flange axis, is defined which is directed OUT OF the cut plane. All vectors in the cut plane are resolved with respect to this unit vector. E.g. force f characterizes a vector which is directed in the direction of n with value equal to f. When this flange is connected to other 1D translational flanges, this means that the axes vectors of the connected flanges are identical.

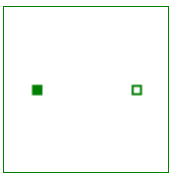
The following variables are transported through this connector:

```
s: Absolute position of the flange in [m]. A positive translation
   means that the flange is translated along the flange axis.
f: Cut-force in direction of the flange axis in [N].
```

Contents

Type	Name	Description
Position	s	absolute position of flange [m]
flow Force	f	cut force directed into flange [N]

Modelica.Mechanics.Translational.Interfaces.Rigid



Rigid connection of two translational 1D flanges

Information

This is a 1D translational component with two *rigidly* connected flanges. The distance between the left and the right flange is always constant, i. e. L. The forces at the right and left flange can be different. It is used e.g. to built up sliding masses.

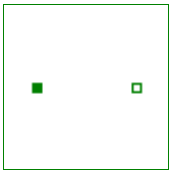
Parameters

Type	Name	Default	Description
Length	L	0	length of component from left flange to right flange (= flange_b.s - flange_a.s) [m]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, i. e. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane, i. e. from right to left)

Modelica.Mechanics.Translational.Interfaces.Compliant



Compliant connection of two translational 1D flanges

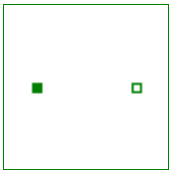
Information

This is a 1D translational component with a *compliant* connection of two translational 1D flanges where inertial effects between the two flanges are not included. The absolute value of the force at the left and the right flange is the same. It is used to built up springs, dampers etc.

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Translational.Interfaces.TwoFlanges



Component with two translational 1D flanges

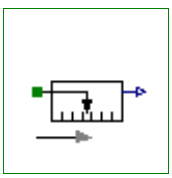
Information

This is a 1D translational component with two flanges. It is used e.g. to built up parts of a drive train consisting of several base components.

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Translational.Interfaces.AbsoluteSensor



Device to measure a single absolute flange variable

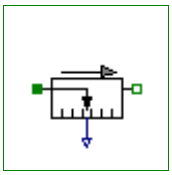
Information

This is the superclass of a 1D translational component with one flange and one output signal in order to measure an absolute kinematic quantity in the flange and to provide the measured signal as output signal for further processing with the Modelica.Blocks blocks.

Connectors

Type	Name	Description
Flange_a	flange_a	flange to be measured (flange axis directed INTO cut plane, e. g. from left to right)
output RealOutput	y	

Modelica.Mechanics.Translational.Interfaces.RelativeSensor



Device to measure a single relative variable between two flanges

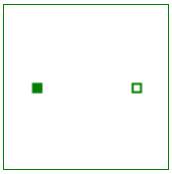
Information

This is a superclass for 1D translational components with two rigidly connected flanges and one output signal in order to measure relative kinematic quantities between the two flanges or the cut-force in the flange and to provide the measured signal as output signal for further processing with the Modelica.Blocks blocks.

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)
output RealOutput	y	

Modelica.Mechanics.Translational.Interfaces.FrictionBase



Base class of Coulomb friction elements

Information

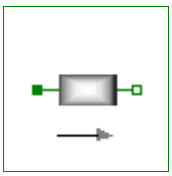
Parameters

Type	Name	Default	Description
Length	L	0	length of component from left flange to right flange (= flange_b.s - flange_a.s) [m]
Position	smax	25	right stop for (right end of) sliding mass [m]
Position	smin	-25	left stop for (left end of) sliding mass [m]
Velocity	v_small	1e-3	Relative velocity near to zero (see model info text) [m/s]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, i. e. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane, i. e. from right to left)

Modelica.Mechanics.Translational.SlidingMass



Sliding mass with inertia

Information

Sliding mass with *inertia, without friction* and two rigidly connected flanges.

The sliding mass has the length L, the position coordinate s is in the middle. Sign convention: A positive force at flange flange_a moves the sliding mass in the positive direction. A negative force at flange flange_a moves the sliding mass to the negative direction.

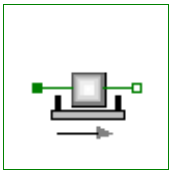
Parameters

Type	Name	Default	Description
Length	L	0	length of component from left flange to right flange (= flange_b.s - flange_a.s) [m]
Mass	m	1	mass of the sliding mass [kg]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, i. e. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane, i. e. from right to left)

Modelica.Mechanics.Translational.Stop

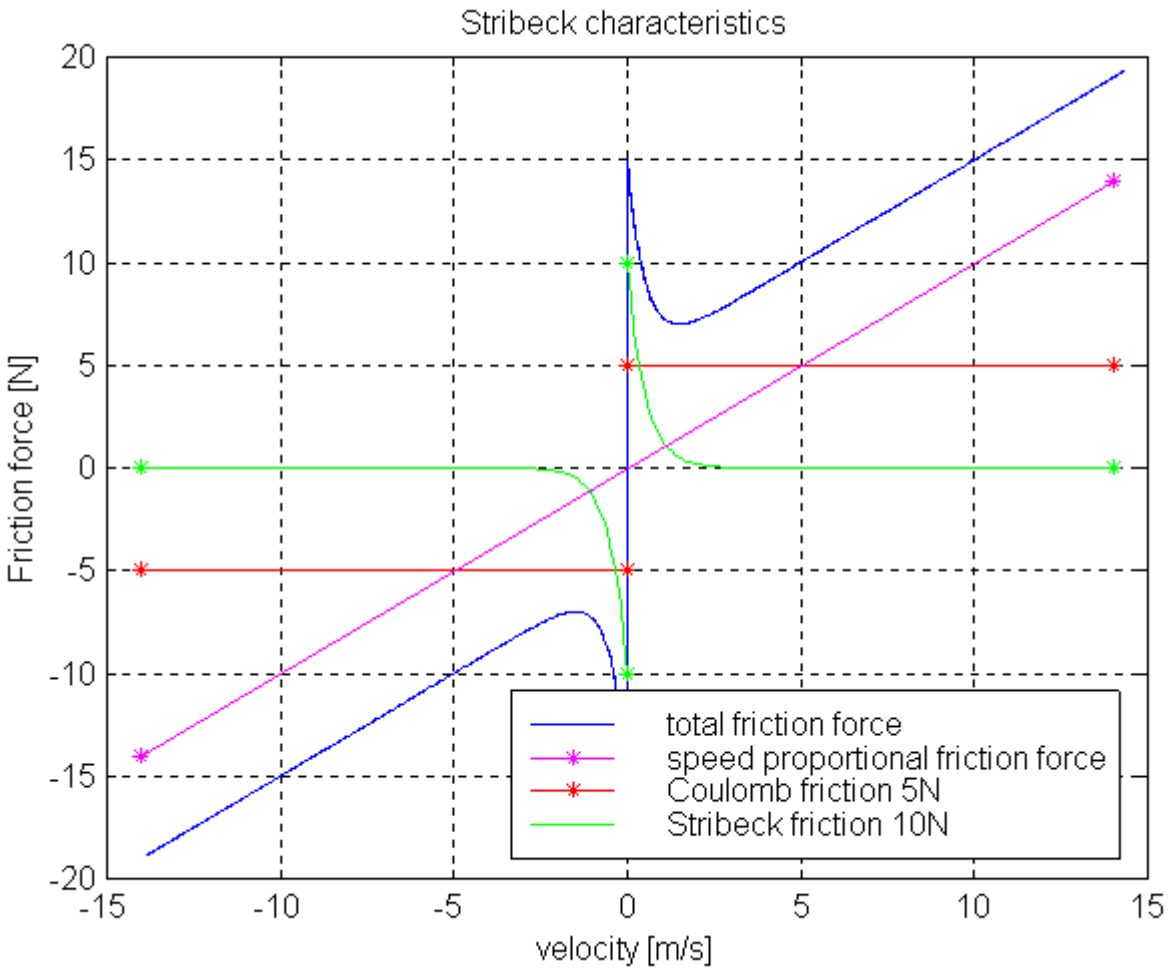


Sliding mass with hard stop and Stribeck friction

Information

This element describes the *Stribeck friction characteristics* of a sliding mass, i. e. the frictional force acting between the sliding mass and the support. Included is a *hard stop* for the position.
The surface is fixed and there is friction between sliding mass and surface. The frictional force f is given for positive velocity v by:

*f = F_Coulomb + F_prop * v + F_Stribeck * exp (-fexp * v)*



The distance between the left and the right connector is given by parameter L. The position of the center of gravity, coordinate s, is in the middle between the two flanges.

There are hard stops at smax and smin, i. e. if

flange_a.s >= smin
and
flange_b.s <= xmax

the sliding mass can move freely.

When the absolute velocity becomes zero, the sliding mass becomes stuck, i.e., the absolute position remains constant. In this phase the friction force is calculated from a force balance due to the requirement that the absolute acceleration shall be zero. The elements begin to slide when the friction force exceeds a threshold value, called the maximum static friction force, computed via:

maximum_static_friction = F_Coulomb + F_Stribeck

This requires the states Stop.s and Stop.v . If these states are eliminated during the index reduction the model will not work. To avoid this any inertias should be connected via springs to the Stop element, other sliding masses, dampers or hydraulic chambers must be avoided.

For more details of the used friction model see the following reference:

Beater P. (1999):

Entwurf hydraulischer Maschinen. Springer Verlag Berlin Heidelberg New York.

The friction model is implemented in a "clean" way by state events and leads to continuous/discrete systems of equations which have to be solved by appropriate numerical methods. The method is described in:

Otter M., Elmqvist H., and Mattsson S.E. (1999):

Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. CACSD'99, Aug. 22.-26, Hawaii.

More precise friction models take into account the elasticity of the material when the two elements are "stuck", as well as other effects, like hysteresis. This has the advantage that the friction element can be completely described by a differential equation without events. The drawback is that the system becomes stiff (about 10-20 times slower simulation) and that more material constants have to be supplied which requires more sophisticated identification. For more details, see the following references, especially (Armstrong and Canudas de Witt 1996):

Armstrong B. (1991):
Control of Machines with Friction. Kluwer Academic Press, Boston MA.

Armstrong B., and Canudas de Wit C. (1996):
Friction Modeling and Compensation. The Control Handbook, edited by W.S.Levine, CRC Press, pp. 1369-1382.

Canudas de Wit C., Olsson H., Astroem K.J., and Lischinsky P. (1995):
A *new model for control of systems with friction.* IEEE Transactions on Automatic Control, Vol. 40, No. 3, pp. 419-425.

Parameters

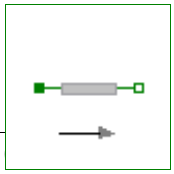
Type	Name	Default	Description
Length	L	0	length of component from left flange to right flange (= flange_b.s - flange_a.s) [m]
Position	smax	25	right stop for (right end of) sliding mass [m]
Position	smin	-25	left stop for (left end of) sliding mass [m]
Velocity	v_small	1e-3	Relative velocity near to zero (see model info text) [m/s]
Mass	m	1	mass [kg]
Real	F_prop	1	velocity dependent friction [N/ (m/s)]
Force	F_Coulomb	5	constant friction: Coulomb force [N]
Force	F_Stribeck	10	Stribeck effect [N]
Real	fexp	2	exponential decay [1/ (m/s)]
Initialization			
Position	s.start		absolute position of center of component (s = flange_a.s + L/2 = flange_b.s - L/2) [m]
Boolean	startForward.start	**false**	true, if v_rel=0 and start of forward sliding or v_rel > v_small
Boolean	startBackward.start	**false**	true, if v_rel=0 and start of backward sliding or v_rel < -v_small
Boolean	locked.start	false	true, if v_rel=0 and not sliding

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, i. e. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane, i. e. from right to left)

Modelica.Mechanics.Translational.Rod

Rod without inertia



Information

Rod *without inertia* and two rigidly connected flanges.

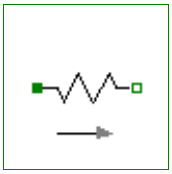
Parameters

Type	Name	Default	Description
Length	L	0	length of component from left flange to right flange (= flange_b.s - flange_a.s) [m]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, i. e. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane, i. e. from right to left)

Modelica.Mechanics.Translational.Spring



Linear 1D translational spring

Information

A *linear 1D translational spring*. The component can be connected either between two sliding masses, or between a sliding mass and the housing (model Fixed), to describe a coupling of the slidin mass with the housing via a spring.

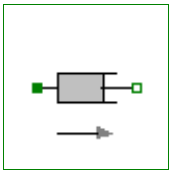
Parameters

Type	Name	Default	Description
Distance	s_rel0	0	unstretched spring length [m]
Real	c	1	spring constant [N/m]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Translational.Damper



Linear 1D translational damper

Information

Linear, velocity dependent damper element. It can be either connected between a sliding mass and the housing (model Fixed), or between two sliding masses.

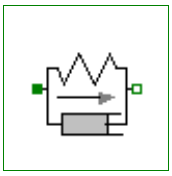
Parameters

Type	Name	Default	Description
Real	d	0	damping constant [N/ (m/s)] [N/ (m/s)]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Translational.SpringDamper



Linear 1D translational spring and damper in parallel

Information

A *spring and damper element connected in parallel*. The component can be connected either between two sliding masses to describe the elasticity and damping, or between a sliding mass and the housing (model Fixed), to describe a coupling of the sliding mass with the housing via a spring/damper.

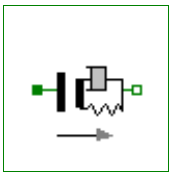
Parameters

Type	Name	Default	Description
Position	s_rel0	0	unstretched spring length [m]
Real	c	1	spring constant [N/m]
Real	d	1	damping constant [N/(m/s)]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Translational.ElastoGap



1D translational spring damper combination with gap

Information

A *linear translational spring damper combination that can lift off*. The component can be connected between a sliding mass and the housing (model Fixed), to describe the contact of a sliding mass with the housing.

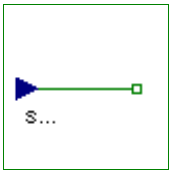
Parameters

Type	Name	Default	Description
Position	s_rel0	0	unstretched spring length [m]
Real	c	1	spring constant [N/m]
Real	d	1	damping constant [N/ (m/s)]

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Mechanics.Translational.Position



Forced movement of a flange according to a reference position

Information

The input signal **s_ref** defines the **reference position** in [m]. Flange **flange_b** is **forced** to move according to this reference motion. According to parameter **exact** (default = **false**), this is done in the following way:

1. **exact=true**
   The reference position is treated **exactly**. This is only possible, if the input signal is defined by an analytical function which can be differentiated at least twice. If this prerequisite is fulfilled, the Modelica translator will differentiate the input signal twice in order to compute the reference acceleration of the flange.
2. **exact=false**
   The reference position is **filtered** and the second derivative of the filtered curve is used to compute the reference acceleration of the flange. This second derivative is **not** computed by numerical differentiation but by an appropriate realization of the filter. For filtering, a second order Bessel filter is used. The critical frequency (also called cut-off frequency) of the filter is defined via parameter **f_crit** in [Hz]. This value should be selected in such a way that it is higher as the essential low frequencies in the signal.

The input signal can be provided from one of the signal generator blocks of the block library Modelica.Blocks.Sources.

Parameters

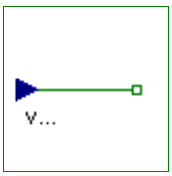
Type	Name	Default	Description
Boolean	exact	false	true/false exact treatment/filtering the input signal
Frequency	f_crit	50	if exact=false, critical frequency of filter to filter input signal [Hz]

Connectors

Type	Name	Description
Flange_b	flange_b	
input RealInput	s_ref	reference position of flange as input signal

---

Modelica.Mechanics.Translational.Speed



Forced movement of a flange according to a reference speed

Information

The input signal **v_ref** defines the **reference speed** in [m/s]. Flange **flange_b** is **forced** to move according to this reference motion. According to parameter **exact** (default = **false**), this is done in the following way:

1. **exact=true**
   The reference speed is treated **exactly**. This is only possible, if the input signal is defined by an analytical function which can be differentiated at least once. If this prerequisite is fulfilled, the Modelica translator will differentiate the input signal once in order to compute the reference acceleration of the flange.
2. **exact=false**
   The reference speed is **filtered** and the first derivative of the filtered curve is used to compute the reference acceleration of the flange. This first derivative is **not** computed by numerical differentiation but by an appropriate realization of the filter. For filtering, a first order filter is used. The critical frequency (also called cut-off frequency) of the filter is defined via parameter **f_crit** in [Hz]. This value should be selected in such a way that it is higher as the essential low frequencies in the signal.

The input signal can be provided from one of the signal generator blocks of the block library Modelica.Blocks.Sources.

Parameters

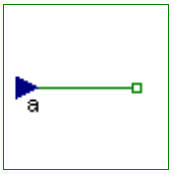
Type	Name	Default	Description
Boolean	exact	false	true/false exact treatment/filtering the input signal
Frequency	f_crit	50	if exact=false, critical frequency of filter to filter input signal [Hz]
Position	s_start	0	Start position of flange_b [m]

Connectors

Type	Name	Description
input RealInput	v_ref	reference speed of flange as input signal
Flange_b	flange_b	Flange that is forced to move according to input signals u

---

Modelica.Mechanics.Translational.Accelerate



Forced movement of a flange according to an acceleration signal

Information

The input signal **a** in [m/s2] moves the 1D translational flange connector flange_b with a predefined *acceleration*, i.e., the flange is *forced* to move with this acceleration. The velocity and the position of the flange are also predefined and are determined by integration of the acceleration.

The acceleration "a(t)" can be provided from one of the signal generator blocks of the block library Modelica.Blocks.Source.

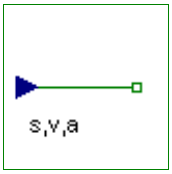
Parameters

Type	Name	Default	Description
Position	s_start	0	Start position [m]
Velocity	v_start	0	Start velocity [m/s]

Connectors

Type	Name	Description
input RealInput	a	absolute acceleration of flange as input signal
Flange_b	flange_b	

Modelica.Mechanics.Translational.Move



Forced movement of a flange according to a position, velocity and acceleration signal

Information

Flange **flange_b** is **forced** to move with a predefined motion according to the input signals:

```
u[1]: position of flange
u[2]: velocity of flange
u[3]: acceleration of flange
```

The user has to guarantee that the input signals are consistent to each other, i.e., that u[2] is the derivative of u[1] and that u[3] is the derivative of u. There are, however, also applications where by purpose these conditions do not hold. For example, if only the position dependent terms of a mechanical system shall be calculated, one may provide position = position(t) and set the velocity and the acceleration to zero.

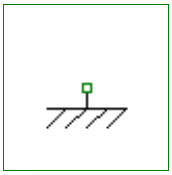
The input signals can be provided from one of the signal generator blocks of the block library Modelica.Blocks.Sources.

Connectors

Type	Name	Description
input RealInput	u[3]	position, velocity and acceleration of flange as input signals
Flange_b	flange_b	Flange that is forced to move according to input signals u

---

Modelica.Mechanics.Translational.Fixed



Fixed flange

Information

The *flange* of a 1D translational mechanical system *fixed* at an position s0 in the *housing*. May be used:

- to connect a compliant element, such as a spring or a damper, between a sliding mass and the housing.
- to fix a rigid element, such as a sliding mass, at a specific position.

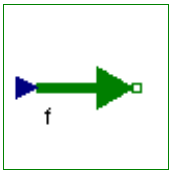
Parameters

Type	Name	Default	Description
Position	s0	0	fixed offset position of housing [m]

Connectors

Type	Name	Description
Flange_b	flange_b	

Modelica.Mechanics.Translational.Force



External force acting on a drive train element as input signal

Information

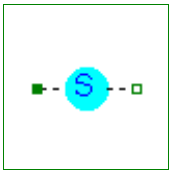
The input signal "s" in [N] characterizes an *external force* which acts (with positive sign) at a flange, i.e., the component connected to the flange is driven by force f.

Input signal s can be provided from one of the signal generator blocks of Modelica.Blocks.Source.

Connectors

Type	Name	Description
Flange_b	flange_b	
input RealInput	f	driving force as input signal

Modelica.Mechanics.Translational.RelativeStates



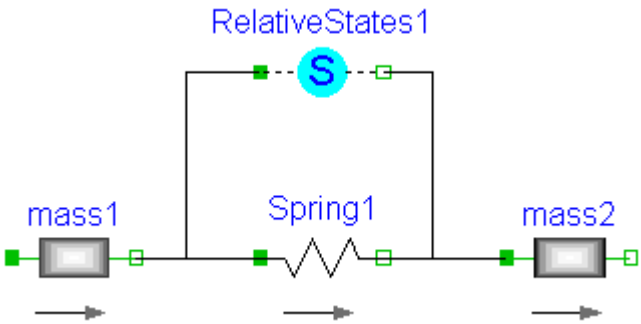
Definition of relative state variables

Information

Usually, the absolute position and the absolute velocity of Modelica.Mechanics.Translational.Inertia models are used as state variables. In some circumstances, relative quantities are better suited, e.g., because it may be easier to supply initial values. In such cases, model **RelativeStates** allows the definition of state variables in the following way:

- Connect an instance of this model between two flange connectors.
- The **relative position** and the **relative velocity** between the two connectors are used as **state variables**.

An example is given in the next figure



Here, the relative position and the relative velocity between the two masses are used as state variables. Additionally, the simulator selects either the absolute position and absolute velocity of model mass1 or of model mass2 as state variables.

Connectors

Type	Name	Description
Flange_a	flange_a	(left) driving flange (flange axis directed INTO cut plane, e. g. from left to right)
Flange_b	flange_b	(right) driven flange (flange axis directed OUT OF cut plane)

Modelica.Media

Library of media property models

Information

This library contains interface definitions for media and the following **property** models for single and multiple substance fluids with one and multiple phases:

- Ideal gases:
  1241 high precision gas models based on the NASA Glenn coefficients, plus ideal gas mixture models based on the same data.
- Water models:
  ConstantPropertyLiquidWater, WaterIF97 (high precision water model according to the IAPWS/IF97 standard)
- Air models:
  SimpleAir, DryAirNasa, and MoistAir
- Incompressible media:

TableBased incompressible fluid models (properties are defined by tables rho(T), HeatCapacity_cp(T), etc.)
- Compressible liquids:
  Simple liquid models with linear compressibility

The following parts are useful, when newly starting with this library:

- Modelica.Media.UsersGuide.
- Modelica.Media.UsersGuide.MediumUsage describes how to use a medium model in a component model.
- Modelica.Media.UsersGuide.MediumDefinition describes how a new fluid medium model has to be implemented.
- Modelica.Media.UsersGuide.ReleaseNotes summarizes the changes of the library releases.
- Modelica.Media.Examples contains examples that demonstrate the usage of this library.

Copyright © 1998-2007, Modelica Association.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
UsersGuide	User's Guide of Media Library
Examples	Demonstrate usage of property models (currently: simple tests)
Interfaces	Interfaces for media models
Common	data structures and fundamental functions for fluid properties
Air	Medium models for air
CompressibleLiquids	compressible liquid models
IdealGases	Data and models of ideal gases (single, fixed and dynamic mixtures) from NASA source
Incompressible	Medium model for T-dependent properties, defined by tables or polynomials
Water	Medium models for water

Modelica.Media.UsersGuide

User's Guide of Media Library

Library **Modelica.Media** is a **free** Modelica package providing a standardized interface to fluid media models and specific media models based on this interface. A fluid medium model defines **algebraic** equations for the intensive thermodynamic variables used in the **mass** and **energy** balance of component models. Optionally, additional medium properties can be computed such as dynamic viscosity or thermal conductivity. Medium models are defined for **single** and **multiple substance** fluids with **one** and **multiple phases**.

A large part of the library provides specific medium models that can be directly utilized. This library can be used in all types of Modelica fluid libraries that may have different connectors and design philosophies. It is particularily utilized in the Modelica_Fluid library (the Modelica_Fluid library is currently under development to provide 1D thermo-fluid flow components for single and multiple substance flow with one and multiple phases). The Modelica.Media library has the following main features:

- Balance equations and media model equations are decoupled. This means that the used medium model does usually not have an influence on how the balance equations are formulated. For

example, the same balance equations are used for media that use pressure and temperature, or pressure and specific enthalpy as independent variables, as well as for incompressible and compressible media models. A Modelica tool will have enough information to generate as efficient code as a traditional (coupled) definition. This feature is described in more detail in section Static State Selection.

- Optional variables, such as dynamic viscosity, are only computed if needed in the corresponding component.
- The independent variables of a medium model do not influence the definition of a fluid connector port. Especially, the media models are implemented in such a way that a connector may have the minimum number of independent medium variables in a connector and still get the same efficiency as if all medium variables are passed by the connector from one component to the next one (the latter approach has the restriction that a fluid port can only connect two components and not more). Note, the Modelica_Fluid library uses the first approach, i.e., having a set of independent medium variables in a connector.
- The medium models are implemented with regards to efficient dynamic simulation. For example, two phase medium models trigger state events at phase boundaries (because the medium variables are not differentiable at this point).

This User's Guide has the following main parts:

- Medium usage describes how to use a medium model from this library in a component model.
- Medium definition describes how a new fluid medium model has to be implemented.
- ReleaseNotes summarizes the changes of the library releases.
- Contact provides information about the authors of the library as well as acknowledgements.

Package Content

Name	Description
MediumUsage	Medium usage
MediumDefinition	Medium definition
ReleaseNotes	Release notes
Contact	Contact

Modelica.Media.UsersGuide.MediumUsage

Medium usage

Content:

1. Basic usage of medium model
2. Medium model for a balance volume
3. Medium model for a pressure loss
4. Optional medium properties
5. Constants provided by medium model
6. Two-phase media
7. Initialization

A good demonstration how to use the media from Modelica.Media is given in package Modelica.Media.Examples.Tests. Under Tests.Components the most basic components of a Fluid library are defined. Under Tests.MediaTestModels these basic components are used to test all media models with some very simple piping networks.

Package Content

Name	Description

BasicUsage	Basic usage
BalanceVolume	Balance volume
ShortPipe	Short pipe
OptionalProperties	Optional properties
Constants	Constants
TwoPhase	Two-phase media
Initialization	Initialization

Modelica.Media.UsersGuide.MediumUsage.BasicUsage

Basic usage

Basic usage of medium model

Media models in Modelica.Media are provided by packages, inheriting from the partial package Modelica.Media.Interfaces.PartialMedium. Every package defines:

- Medium **constants** (such as the number of chemical substances, molecular data, critical properties, etc.).
- A BaseProperties **model**, to compute the basic thermodynamic properties of the fluid;
- **setState_XXX** functions to compute the thermodynamic state record from different input arguments (such as density, temperature, and composition which would be setState_dTX);
- **Functions** to compute additional properties (such as saturation properties, viscosity, thermal conductivity, etc.).

There are - as stated above - two different basic ways of using the Media library which will be described in more details in the following section. One way is to use the model BaseProperties. Every instance of BaseProperties for any medium model provides **3+nXi equations** for the following **5+nXi variables** that are declared in the medium model (nXi is the number of independent mass fractions, see explanation below):

Variable	Unit	Description
T	K	temperature
p	Pa	absolute pressure
d	kg/m^3	density
u	J/kg	specific internal energy
h	J/kg	specific enthalpy (h = u + p/d)
Xi[nXi]	kg/kg	independent mass fractions m_i/m
X[nX]	kg/kg	All mass fractions m_i/m. X is defined in BaseProperties by: X = **if** reducedX **then** vector([Xi; 1-**sum**(Xi)]) **else** Xi

Two variables out of p, d, h, or u, as well as the **mass fractions** Xi are the **independent** variables and the medium model basically provides equations to compute the remaining variables, including the full mass fraction vector X (more details to Xi and X are given further below).

In a component, the most basic usage of a medium model is as follows

```
  model Pump
    replaceable package Medium = Modelica.Media.Interfaces.PartialMedium
                          "Medium model" annotation (choicesAllMatching = true);
    Medium.BaseProperties medium_a "Medium properties at location a (e.g.
port_a)";
    // Use medium variables (medium_a.p, medium_a.T, medium_a.h, ...)
      ...
  end Pump;
```

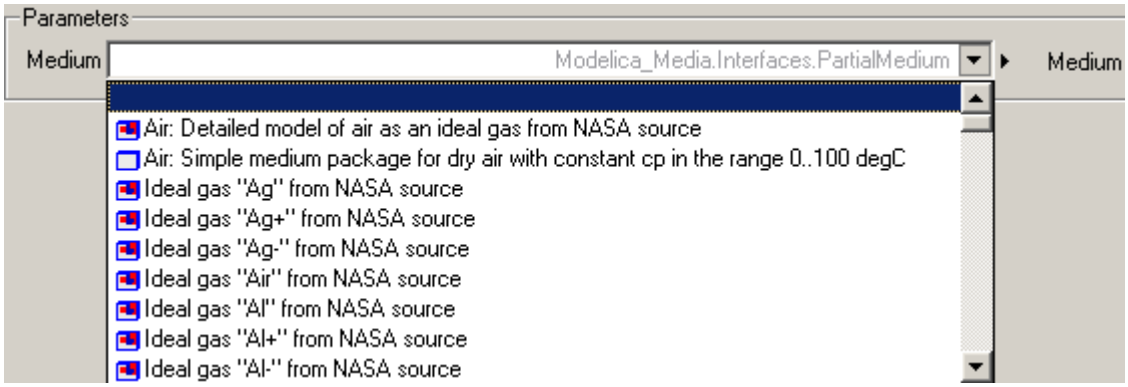
The second way is to use the setState_XXX functions to compute the thermodynamic state record from which all other thermodynamic state variables can be computed (see Basic definition of medium for further details on ThermodynamicState). The setState_XXX functions accept either X or Xi (see explanation below) and will decide internally which of these two compositions is provided by the user. The four fundamental setState_XXX functions are provided in PartialMedium

Function	Description	Short-form for single component medium
setState_dTX	computes ThermodynamicState from density, temperature, and composition X or Xi	setState_dT
setState_phX	computes ThermodynamicState from pressure, specific enthalpy, and composition X or Xi	setState_ph
setState_psX	computes ThermodynamicState from pressure, specific entropy, and composition X or Xi	setState_ps
setState_pTX	computes ThermodynamicState from pressure, temperature, and composition X or Xi	setState_pT

The simple example that explained the basic usage of BaseProperties would then become

```
  model Pump
    replaceable package Medium = Modelica.Media.Interfaces.PartialMedium
                          "Medium model" annotation (choicesAllMatching = true);
    Medium.ThermodynamicState state_a "Thermodynamic state record at location a
(e.g. port_a)";
    // Compute medium variables from thermodynamic state record
(pressure(state_a), temperature(state_a),
    // specificEnthalpy(state_a), ...)
    ...
  end Pump;
```

All media models are directly or indirectly a subpackage of package Modelica.Media.Interfaces.PartialMedium. Therefore, a medium model in a component should inherit from this partial package. Via the annotation "choicesAllMatching = true" it is defined that the tool should display a selection box with all loaded packages that inherit from PartialMedium. An example is given in the next figure:



A selected medium model leads, e.g., to the following equation:

```
  Pump pump(redeclare package Medium = Modelica.Media.Water.SimpleLiquidWater);
```

Usually, a medium model is associated with the variables of a fluid connector. Therefore, equations have to be defined in a model that relate the variables in the connector with the variables in the medium model:

```
  model Pump
    replaceable package Medium = Modelica.Media.Interfaces.PartialMedium
                          "Medium model" annotation (choicesAllMatching = true);
```

```
    Medium.BaseProperties medium_a "Medium properties of port_a";
    // definition of the fluid port port_a
     ...
  equation
    medium.p = port_a.p;
    medium.h = port_a.h;
    medium.Xi = port_a.Xi;
     ...
  end Pump;
```

in the case of using BaseProperties or

```
  model Pump
    replaceable package Medium = Modelica.Media.Interfaces.PartialMedium
                         "Medium model" annotation (choicesAllMatching = true);
    Medium.ThermodynamicState state_a "Thermodynamic state record of medium at
port_a";
    // definition of the fluid port port_a
     ...
  equation
    state_a = Medium.setState_phX(port_a.p, port_a.h, port_a.Xi) // if port_a
contains the variables
                                                                 // p, h, and Xi
     ...
  end Pump;
```

in the case of using ThermodynamicState.

If a component model shall treat both single and multiple substance fluids, equations for the mass fractions have to be present (above: medium.Xi = port_a.Xi) in the model. According to the Modelica semantics, the equations of the mass fractions are ignored, if the dimension of Xi is zero, i.e., for a single-component medium. Note, by specific techniques sketched in section "Medium definition", the independent variables in the medium model need not to be the same as the variables in the connector and still get the same efficiency, as if the same variables would be used.

If a fluid consists of a single substance, **nXi = 0** and the vector of mass fractions Xi is not present. If a fluid consists of nS substances, the medium model may define the number of independent mass fractions **nXi** to be **nS**, **nS-1**, or zero. In all cases, balance equations for nXi substances have to be given in the corresponding component (see discussion below). Note, that if nXi = nS, the constraint "sum(Xi)=1" between the mass fractions is **not** present in the model; in that case, it is necessary to provide consistent start values for Xi such that sum(Xi) = 1.

The reason for this definition of Xi is that a fluid component library can be implemented by using only the independent mass fractions Xi and then via the medium it is defined how Xi is interpreted:

- If Xi = nS, then the constraint equation sum(X) = 1 is neglected during simulation. By making sure that the initial conditions of X fulfill this constraint, it can usually be guaranteed that small errors in sum(X) = 1 remain small although this constraint equation is not explicitly used during the simulation. This approach is usually useful if components of the mixture can become very small. If such a small quantity is computed via the equation 1 - sum(X[1:nX-1]), there might be large numerical errors and it is better to compute it via the corresponding balance equation.
- If Xi = nS-1, then the true independent mass fractions are used in the fluid component and the last component of X is computed via X[nX] = 1 - sum(Xi). This is useful for, e.g., MoistAir, where the number of states should be as small as possible without introducing numerical problems.
- If Xi = 0, then the reference value of composition reference_X is assumed. This case is useful to avoid composition states in all the cases when the composition will always be constant, e.g. with circuits having fixed composition sources.

The full vector of mass fractions **X[nX]** is computed in PartialMedium.BaseProperties based on Xi, reference_X, and the information whether Xi = nS or nS-1. For single-substance media, nX = 0, so there's also no X vector. For multiple-substance media, nX = nS, and X always contains the full vector of mass

fractions. In order to reduce confusion for the user of a fluid component library, "Xi" has the annotation "Hide=true", meaning, that this variable is not shown in the plot window. Only X is shown in the plot window and this vector always contains all mass fractions.

Modelica.Media.UsersGuide.MediumUsage.BalanceVolume

Balance volume

Fluid libraries usually have balance volume components with one fluid connector port that fulfill the mass and energy balance and on a different grid components that fulfill the momentum balance. A balance volume component, called junction volume below, should be primarily implemented in the following way (see also the implementation in Modelica.Media.Examples.Tests.Components.PortVolume):

```
  model JunctionVolume
    import SI=Modelica.SIunits;
    import Modelica.Media.Examples.Tests.Components.FluidPort_a;

    parameter SI.Volume V = 1e-6 "Fixed size of junction volume";
    replaceable package Medium = Modelica.Media.Interfaces.PartialMedium
                           "Medium model" annotation (choicesAllMatching =
true);

    FluidPort_a port(redeclare package Medium = Medium);
    Medium.BaseProperties medium(preferredMediumStates = true);

    SI.Energy U               "Internal energy of junction volume";
    SI.Mass   M               "Mass of junction volume";
    SI.Mass   MX[Medium.nXi] "Independent substance masses of junction volume";
  equation
    medium.p  = port.p;
    medium.h  = port.h;
    medium.Xi = port.Xi;

    M  = V*medium.d;                  // mass of JunctionVolume
    MX = M*medium.Xi;                // mass fractions in JunctionVolume
    U  = M*medium.u;                  // internal energy in JunctionVolume

    der(M)  = port.m_flow;    // mass balance
    der(MX) = port.mX_flow;   // substance mass balance
    der(U)  = port.H_flow;    // energy balance
  end JunctionVolume;
```

Assume the Modelica.Media.Air.SimpleAir medium model is used with the JunctionVolume model above. This medium model uses pressure p and temperature T as independent variables. If the flag "preferredMediumStates" is set to **true** in the declaration of "medium", then the independent variables of this medium model get the attribute "stateSelect = StateSelect.prefer", i.e., the Modelica translator should use these variables as states, if this is possible. Basically, this means that constraints between the potential states p,T and the potential states U,M are present. A Modelica tool will therefore **automatically** differentiate medium equations and will use the following equations for code generation (note the equations related to X are removed, because SimpleAir consists of a single substance only):

```
    M  = V*medium.d;
    U  = M*medium.u;

    // balance equations
    der(M)  = port.m_flow;
    der(U)  = port.H_flow;
```

```
    // abbreviations introduced to get simpler terms
    p = medium.p;
    T = medium.T;
    d = medium.d;
    u = medium.u;
    h = medium.h;

    // medium equations
    d = fd(p,T);
    h = fh(p,T);
    u = h - p/d;

    // equations derived automatically by a Modelica tool due to index reduction
    der(U) = der(M)*u + M*der(u);
    der(M) = V*der(d);
    der(u) = der(h) - der(p)/d - p/der(d);
    der(d) = der(fd,p)*der(p) + der(fd,T)*der(T);
    der(h) = der(fh,p)*der(p) + der(fd,T)*der(T);
```

Note, that "der(y,x)" is an operator that characterizes in the example above the partial derivative of y with respect to x (this operator will be included in one of the next Modelica language releases). All media models in this library are written in such a way that at least the partial derivatives of the medium variables with respect to the independent variables are provided, either because the equations are directly given (= symbolic differentiation is possible) or because the derivative of the corresponding function (such as fd above) is provided. A Modelica tool will transform the equations above in differential equations with p and T as states, i.e., will generate equations to compute **der**(p) and **der**(T) as function of p and T.

Note, when preferredMediumStates = **false**, no differentiation will take place and the Modelica translator will use the variables appearing differentiated as states, i.e., M and U. This has the disadvantage that for many media non-linear systems of equations are present to compute the intrinsic properties p, d, T, u, h from M and U.

Modelica.Media.UsersGuide.MediumUsage.ShortPipe

Short pipe

Fluid libraries have components with two ports that store neither mass nor energy and fulfill the momentum equation between their two ports, e.g., a short pipe. In most cases this means that an equation is present relating the pressure drop between the two ports and the mass flow rate from one to the other port. Since no mass or energy is stored, no differential equations for thermodynamic variables are present. A component model of this type has therefore usually the following structure (see also the implementation in Modelica.Media.Examples.Tests.Components.ShortPipe):

```
  model ShortPipe
    import SI=Modelica.SIunits;
    import Modelica.Media.Examples.Tests.Components;

    // parameters defining the pressure drop equation

    replaceable package Medium = Modelica.Media.Interfaces.PartialMedium
                           "Medium model" annotation (choicesAllMatching =
true);

    Component.FluidPort_a port_a (redeclare package Medium = Medium);
    Component.FluidPort_b port_b (redeclare package Medium = Medium);

    SI.Pressure dp = port_a.p - port_b.p "Pressure drop";
```

```
  Medium.BaseProperties medium_a "Medium properties in port_a";
  Medium.BasePropreties medium_b "Medium properties in port_b";
equation
  // define media models of the ports
  medium_a.p   = port_a.p;
  medium_a.h   = port_a.h;
  medium_a.Xi = port_a.Xi;

  medium_b.p   = port_b.p;
  medium_b.h   = port_b.h;
  medium_b.Xi = port_b.Xi;

  // Handle reverse and zero flow (semiLinear is a built-in Modelica operator)
  port_a.H_flow   = semiLinear(port_a.m_flow, port_a.h, port_b.h);
  port_a.mXi_flow = semiLinear(port_a.m_flow, port_a.Xi, port_b.Xi);

  // Energy, mass and substance mass balance
  port_a.H_flow + port_b.H_flow = 0;
  port_a.m_flow + port_b.m_flow = 0;
  port_a.mXi_flow + port_b.mXi_flow = zeros(Medium.nXi);

  // Provide equation: port_a.m_flow = f(dp)
end ShortPipe;
```

The **semiLinear**(..) operator is basically defined as:

```
semiLinear(m_flow, ha, hb) = if m_flow ≥ 0 then m_flow*ha else m_flow*hb;
```

that is, it computes the enthalpy flow rate either from the port_a or from the port_b properties, depending on flow direction. The exact details of this operator are given in ModelicaReference.Operators.SemiLinear. Especially, rules are defined in the Modelica specification that m_flow = 0 can be treated in a "meaningful way". Especially, if n fluid components (such as pipes) are connected together and the fluid connector from above is used, a linear system of equations appear between medium1.h, medium2.h, medium3.h, ..., port1.h, port2.h, port3.h, ..., port1.H_flow, port2.H_flow, port3.H_flow, ... The rules for the semiLinear(..) operator allow the following solution of this linear system of equations:

- n = 2 (two components are connected):
  The linear system of equations can be analytically solved with the result

  ```
  medium1.h = medium2.h = port1.h = port2.h
  0 = port1.H_flow + port2.H_flow
  ```

  Therefore, no problems with zero mass flow rate are present.
- n > 2 (more than two components are connected together):
  The linear system of equations is solved numerically during simulation. For m_flow = 0, the linear system becomes singular and has an infinite number of solutions. The simulator could use the solution t that is closest to the solution in the previous time step ("least squares solution"). Physically, the solution is determined by diffusion which is usually neglected. If diffusion is included, the linear system is regular.

Modelica.Media.UsersGuide.MediumUsage.OptionalProperties

Optional properties

In some cases additional medium properties are needed. A component that needs these optional properties has to call one of the functions listed in the following table. They are defined as partial functions within package PartialMedium, and then (optionally) implemented

in actual medium packages. If a component calls such an optional function and the medium package does not provide a new implementation for this function, an error message is printed at translation time, since the function is "partial", i.e., incomplete. The argument of all functions is the **state** record, automatically defined by the BaseProperties model or specifically computed using the setState_XXX functions, which contains the minimum number of thermodynamic variables needed to compute all the additional properties. In the table it is assumed that there is a declaration of the form:

```
replaceable package Medium = Modelica.Media.Interfaces.PartialMedium;
Medium.ThermodynamicState state;
```

Function call	Unit	Description
Medium.dynamicViscosity(state)	Pa.s	dynamic viscosity
Medium.thermalConductivity(state)	W/(m.K)	thermal conductivity
Medium.prandtlNumber(state)	1	Prandtl number
Medium.specificEntropy(state)	J/(kg.K)	specific entropy
Medium.specificHeatCapacityCp(state)	J/(kg.K)	specific heat capacity at constant pressure
Medium.specificHeatCapacityCv(state)	J/(kg.K)	specific heat capacity at constant density
Medium.isentropicExponent(state)	1	isentropic exponent
Medium.isentropicEnthatlpy(pressure, state)	J/kg	isentropic enthalpy
Medium.velocityOfSound(state)	m/s	velocity of sound
Medium.isobaricExpansionCoefficient(state)	1/K	isobaric expansion coefficient
Medium.isothermalCompressibility(state)	1/Pa	isothermal compressibility
Medium.density_derp_h(state)	kg/(m3.Pa)	derivative of density by pressure at constant enthalpy
Medium.density_derh_p(state)	kg2/(m3.J)	derivative of density by enthalpy at constant pressure
Medium.density_derp_T(state)	kg/(m3.Pa)	derivative of density by pressure at constant temperature
Medium.density_derT_p(state)	kg/(m3.K)	derivative of density by temperature at constant pressure
Medium.density_derX(state)	kg/m3	derivative of density by mass fraction
Medium.molarMass(state)	kg/mol	molar mass

There are also some short forms provided for user convenience that allow the computation of certain thermodynamic state variables without using the ThermodynamicState record explicitly. Those short forms are for example useful to compute consistent start values in the initial equation section. Let's consider the function temperature_phX(p,h,X) as an exmaple. This function computes the temperature from pressure, specific enthalpy, and composition X (or Xi) and is a short form for writing

```
temperature(setState_phX(p,h,X))
```

The following functions are predefined in PartialMedium (other functions can be added in the actual medium implementation package if they are useful)

Medium.specificEnthalpy_pTX(p,T,X)	J/kg	Specific enthalpy at p, T, X
Medium.temperature_phX(p,h,X)	K	Temperature at p, h, X
Medium.density_phX(p,h,X)	kg/m³	Density at p, h, X
Medium.temperature_psX(p,s,X)	K	Temperature at p, s, X
Medium.specificEnthalpy_psX(p,s,X)	J/(kg.K)	Specific entropy at p, s, X

Assume for example that the dynamic viscosity eta is needed in the pressure drop equation of a short pipe.

Then, the model of a short pipe has to be changed to:

```
model ShortPipe
    ...
  Medium.BaseProperties medium_a "Medium properties in port_a";
  Medium.BaseProperties medium_b "Medium properties in port_b";
    ...
  Medium.DynamicViscosity eta;
    ...
  eta = if port_a.m_flow > 0 then
            Medium.dynamicViscosity(medium_a.state)
        else
            Medium.dynamicViscosity(medium_b.state);
  // use eta in the pressure drop equation: port_a.m_flow = f(dp, eta)
end ShortPipe;
```

Note, "Medium.DynamicViscosity" is a type defined in Modelica.Interfaces.PartialMedium as

```
import SI = Modelica.SIunits;
type DynamicViscosity = SI.DynamicViscosity (
                                    min=0,
                                    max=1.e8,
                                    nominal=1.e-3,
                                    start=1.e-3);
```

Every medium model may modify the attributes, to provide, e.g., min, max, nominal, and start values adapted to the medium. Also, other types, such as AbsolutePressure, Density, MassFlowRate, etc. are defined in PartialMedium. Whenever possible, these medium specific types should be used in a model in order that medium information, e.g., about nominal or start values, are automatically utilized.

Modelica.Media.UsersGuide.MediumUsage.Constants

Constants

Every medium model provides the following **constants**. For example, if a medium is declared as:

```
replaceable package Medium = Modelica.Media.Interfaces.PartialMedium;
```

then constants "Medium.mediumName", "Medium.nX", etc. are defined:

Type	Name	Description
String	mediumName	Unique name of the medium (is usually used to check whether the media in different components connected together are the same, by providing Medium.mediumName as quantity attribute of the mass flow rate in the connector)
String	substanceNames[nS]	Names of the substances that make up the medium. If only one substance is present, substanceNames = {mediumName}.
String	extraPropertiesNames[nC]	Names of the extra transported substances, outside of mass and energy balances.
Boolean	singleState	= **true**, if u and d are not a function of pressure, and thus only a function of a single thermal variable (temperature or enthalpy) and of Xi for a multiple substance medium. Usually, this flag is **true** for incompressible media. It is used in a model to determine whether 1+nXi (singleState=**true**) or 2+nXi (singleState=**false**) initial conditions have to be provided for a volume element that contains mass and energy balance.

AbsolutePressure	reference_p	Reference pressure for the medium
MassFraction	reference_X[nX]	Reference composition for the medium
AbsolutePressure	p_default	Default value for pressure of medium (for initialization)
Temperature	T_default	Default value for temperature of medium (for initialization)
SpecificEnthalpy	h_default	Default value for specific enthalpy of medium (for initialization)
MassFraction	X_default[nX]	Default value for mass fractions of medium (for initialization)
Integer	nS	number of substances contained in the medium.
Integer	nX	Size of the full mass fraction vector X. If there is a single substance, then nX = 0, else nX=nS.
Integer	nXi	Number of independent mass fractions. If there is a single substance, then nXi = 0.
Boolean	reducedX	= **true**, if the medium has a single substance, or if the medium model has multiple substances and contains the equation sum(X) = 1. In both cases, nXi = nS - 1 (unless fixedX = true). = **false**, if the medium has multiple substances and does not contain the equation sum(X)=1, i.e., nXi = nX = nS (unless fixedX = true).
Boolean	fixedX	= **false**: the composition of the medium can vary, and is determined by nXi independent mass fractions (see reducedX above). = **true**: the composition of the medium is always reference_X, and nXi = 0.
FluidConstants	fluidConstants[nS]	Critical, triple, molecular and other standard data that are provided for every substance of a medium.

The record FluidConstants that is defined in PartialMedium contains the following elements

Type	**Name**	**Description**
String	iupacName	complete IUPAC name
String	casRegistryNumber	chemical abstracts sequencing number
String	chemicalFormula	Chemical formula, (brutto, nomenclature according to Hill)
String	structureFormula	Chemical structure formula
MolarMass	molarMass	molar mass

This record is extended in the partial packages further down the hierarchy (such as PartialTwoPhaseMedium or PartialMixtureMedium) and may contain some or all of the following elements

Temperature	criticalTemperature	critical temperature
AbsolutePressure	criticalPressure	critical pressure
MolarVolume	criticalMolarVolume	critical molar Volume
Real	acentricFactor	Pitzer acentric factor
Temperature	triplePointTemperature	triple point temperature
AbsolutePressure	triplePointPressure	triple point pressure
Temperature	meltingPoint	melting point at 101325 Pa
Temperature	normalBoilingPoint	normal boiling point (at 101325 Pa)
DipoleMoment	dipoleMoment	dipole moment of molecule in Debye (1 debye = 3.33564e10-30 C.m)
Boolean	hasIdealGasHeatCapacity	true if ideal gas heat capacity is available
Boolean	hasCriticalData	true if critical data are known
Boolean	hasDipoleMoment	true if a dipole moment known
Boolean	hasFundamentalEquation	true if a fundamental equation
Boolean	hasLiquidHeatCapacity	true if liquid heat capacity is available

Boolean	hasSolidHeatCapacity	true if solid heat capacity is available
Boolean	hasAccurateViscosityData	true if accurate data for a viscosity function is available
Boolean	hasAccurateConductivityData	true if accurate data for thermal conductivity is available
Boolean	hasVapourPressureCurve	true if vapour pressure data, e.g. Antoine coefficents are known
Boolean	hasAcentricFactor	true if Pitzer accentric factor is known
SpecificEnthalpy	HCRIT0	Critical specific enthalpy of the fundamental equation
SpecificEntropy	SCRIT0	Critical specific entropy of the fundamental equation
SpecificEnthalpy	deltah	Difference between specific enthalpy model (h_m) and f.eq. (h_f) (h_m - h_f)
SpecificEntropy	deltas	Difference between specific enthalpy model (s_m) and f.eq. (s_f) (s_m - s_f)

Modelica.Media.UsersGuide.MediumUsage.TwoPhase

Two-phase media

Models for media which can exist in one-phase or two-phase conditions inherit from Modelica.Media.Interfaces.PartialTwoPhaseMedium (which inherits from PartialMedium). The basic usage of these media models is the same as described in the previous sections. However, additional functionalities are provided, which apply only to potentially two-phase media.

The following additional medium **constants** are provided:

Type	Name	Description
Boolean	smoothModel	If this flag is false (default value), then events are triggered whenever the saturation boundary is crossed; otherwise, no events are generated.
Boolean	onePhase	If this flag is true, then the medium model assumes it will be never called in the two-phase region. This can be useful to speed up the computations in a two-phase medium, when the user is sure it will always work in the one-phase region. Default value: false.

The setState_ph(), setState_ps(), setState_dT() and setState_pT() functions have one extra input, named *phase*. If the phase input is not specified, or if it is given a value of zero, then the setState function will determine the phase, based on the other input values. An input phase = 1 will force the setState function to return a state vector corresponding to a one-phase state, while phase = 2 will force the setState value to return a state vector corresponding to a two-phase state, as shown in the following example;

```
  replaceable package Medium = Modelica.Media.Interfaces.PartialTwoPhaseMedium;
  Medium.ThermodynamicState state, state1, state2;
equation
  // Set the state, given the pressure and the specific enthalpy
  // the phase is determined by the (p, h) values, and can be retrieved
  // from the state record
  state = Medium.setState_ph(p, h);
  phase = state1.phase;

  // Force the computation of the state with one-phase
  // equations of state, irrespective of the (p, h) values
  state1 = Medium.setState_ph(p, h, 1);

  // Force the computation of the state with 2-phase
  // equations of state, irrespective of the (p, h) values
  state2 = Medium.setState_ph(p, h, 2);
```

This feature can be used for the following purposes:

- saving computational time, if one knows in advance the phase of the medium;
- unambiguously determine the phase, when the two inputs correspond to a point on the saturation boundary (the derivative functions have substantially different values on either side);
- get the properties of metastable states, like superheated water or subcooled vapour.

Many additional optional functions are defined to compute properties of saturated media, either liquid (bubble point) or vapour (dew point). The argument to such functions is a SaturationProperties record, which can be set starting from either the saturation pressure or the saturation temperature, as shown in the following example.

```
  replaceable package Medium = Modelica.Media.Interfaces.PartialTwoPhaseMedium;
  Medium.SaturationProperties sat_p;
  Medium.SaturationProperties sat_T;
equation
  // Set sat_p to saturation properties at pressure p
  sat_p = Medium.setSat_p(p);

  // Compute saturation properties at pressure p
  saturationTemperature_p = Medium.saturationTemperature_sat(sat_p);
  bubble_density_p =        Medium.bubbleDensity(sat_p);
  dew_enthalpy_p   =        Medium.dewEnthalpy(sat_p);

  // Set sat_T to saturation properties at temperature T
  sat_T = Medium.setSat_T(T);

  // Compute saturation properties at temperature T
  saturationTemperature_T = Medium.saturationPressure_sat(sat_T);
  bubble_density_T =        Medium.bubbleDensity(sat_T);
  dew_enthalpy_T =          Medium.dewEnthalpy(sat_T);
```

With reference to a model defining a pressure p, a temperature T, and a SaturationProperties record sat, the following functions are provided:

Function call	Unit	Description
Medium.saturationPressure(T)	Pa	Saturation pressure at temperature T
Medium.saturationTemperature(p)	K	Saturation temperature at pressure p
Medium.saturationTemperature_derp(p)	K/Pa	Derivative of saturation temperature with respect to pressure
Medium.saturationTemperature_sat(sat)	K	Saturation temperature
Medium.saturationPressure_sat(sat)	Pa	Saturation pressure
Medium.bubbleEnthalpy(sat)	J/kg	Specific enthalpy at bubble point
Medium.dewEnthalpy(sat)	J/kg	Specific enthalpy at dew point
Medium.bubbleEntropy(sat)	J/(kg.K)	Specific entropy at bubble point
Medium.dewEntropy(sat)	J/(kg.K)	Specific entropy at dew point
Medium.bubbleDensity(sat)	kg/m3	Density at bubble point
Medium.dewDensity(sat)	kg/m3	Density at dew point
Medium.saturationTemperature_derp_sat(sat)	K/Pa	Derivative of saturation temperature with respect to pressure
Medium.dBubbleDensity_dPressure(sat)	kg/(m3.Pa)	Derivative of density at bubble point with respect to pressure
Medium.dDewDensity_dPressure(sat)	kg/(m3.Pa)	Derivative of density at dew point with respect to pressure
Medium.dBubbleEnthalpy_dPressure(sat)	J/(kg.Pa)	Derivative of specific enthalpy at bubble point with respect to pressure
Medium.dDewEnthalpy_dPressure(sat)	J/(kg.Pa)	Derivative of specific enthalpy at dew point with

		respect to pressure
Medium.surfaceTension(sat)	N/m	Surface tension between liquid and vapour phase

Sometimes it can be necessary to compute fluid properties in the thermodynamic plane, just inside or outside the saturation dome. In this case, it is possible to obtain an instance of a ThermodynamicState state vector, and then use it to call the additional functions already defined for one-phase media.

Function call	Description
Medium.setBubbleState(sat, phase)	Obtain the thermodynamic state vector corresponding to the bubble point. If phase==1 (default), the state is on the one-phase side; if phase==2, the state is on the two-phase side
Medium.setDewState(sat, phase)	Obtain the thermodynamic state vector corresponding to the dew point. If phase==1 (default), the state is on the one-phase side; if phase==2, the state is on the two-phase side

Here are some examples:

```
  replaceable package Medium = Modelica.Media.Interfaces.PartialTwoPhaseMedium;
  Medium.SaturationProperties sat;
  Medium.ThermodynamicState   dew_1;    // dew point, one-phase side
  Medium.ThermodynamicState   bubble_2; // bubble point, two phase side
equation
  // Set sat to saturation properties at pressure p
  sat = setSat_p(p);

  // Compute dew point properties, (default) one-phase side
  dew_1 = setDewState(sat);
  cpDew = Medium.specificHeatCapacityCp(dew_1);
  drho_dp_h_1 = Medium.density_derp_h(dew_1);

  // Compute bubble point properties, two-phase side
  bubble_2    = setBubbleState(sat, 2);
  drho_dp_h_2 = Medium.density_derp_h(bubble_2);
```

Modelica.Media.UsersGuide.MediumUsage.Initialization

Initialization

When a medium model is used in a balance volume, differential equations for the independent medium variables are present and therefore initial conditions have to be provided. The following possibilities exist:

Steady state initialization

Modelica has currently no language element to define steady state initialization. In the Modelica simulation environment Dymola, the option

```
  Advanced.DefaultSteadyStateInitialization = true
```

can be set before translation. Then, missing initial conditions are provided by automamtically setting appropriate state derivatives to zero.

Explicit start values or initial equations

Explicit start values can be defined with the "start" and "fixed" attributes. The number of independent variables nx need to be known which can be deduced from the medium constants (nx = nXi + **if** singleState **then** 1 **else** 2). Then, start values or initial equations can be defined for nx variables (= p, T, d, u, h, Xi) from Medium.BaseProperties, e.g., in the form:

```
    replaceable package Medium = Medium.Interfaces.PartialMedium;
    Medium.BaseProperties medium1 (p(start=1e5, fixed=not Medium.singleState),
                                   T(start=300, fixed=true));
    Medium.BaseProperties medium2;
initial equation
    if not Medium.singleState then
       medium2.p = 1e5;
    end if;
    medium2.T = 300;
equation
```

If initial conditions are not provided for the independent medium variables, non-linear systems of equations may occur to compute the initial values of the independent medium variables from the provided initial conditions.

Guess values

If non-linear systems of equations occur during initialization, e.g., in case of steady state initialization, guess values for the iteration variables of the non-linear system of equations have to be provided via the "start" attribute (and fixed=false). Unfortunately, it is usually not known in advance which variables are selected as iteration variables of a non-linear system of equations. One of the following possibilities exist:

- Do not supply start values and hope that the medium specific types have meaningful start values, such as in "Medium.AbsolutePressure"
- Supply start values on all variables of the BaseProperties model, i.e., on p, T, d, u, h, Xi.
- Determine the iteration variables of the non-linear systems of equations and provide start values for these variables. In the Modelica simulation environment Dymola, the iteration variables can be determined by setting the command

  ```
  Advanced.OutputModelicaCode = true
  ```

  and by inspection of the file "dsmodel.mof" that is generated when this option is set (search for "nonlinear").

Modelica.Media.UsersGuide.MediumDefinition

Medium definition

If a new medium model shall be introduced, copy package Modelica.Media.Interfaces.TemplateMedium to the desired location, remove the "partial" keyword from the package and provide the information that is requested in the comments of the Modelica source. A more detailed description for the different parts of the TemplateMedium package is given here:

1. Basic structure of medium interface
2. Basic definition of medium model
3. Multiple Substances
4. Specific enthalpy as function
5. Static State Selection
6. Test of medium model

Package Content

Name	Description
BasicStructure	Basic structure

BasicDefinition	Basic definition
MultipleSubstances	Multiple Substances
SpecificEnthalpyAsFunction	Specific enthalpy as function
StaticStateSelection	Static State Selection
TestOfMedium	Test of medium

Modelica.Media.UsersGuide.MediumDefinition.BasicStructure

Basic structure

A medium model of Modelica.Media is essentially a **package** that contains the following definitions:

- Definition of **constants**, such as the medium name.
- A **model** in the package that contains the 3 basic thermodynamic equations that relate the 5+nXi primary medium variables.
- **Optional functions** to compute medium properties that are only needed in certain circumstances, such as dynamic viscosity. These optional functions need not be provided by every medium model.
- **Type** definitions, which are adapted to the particular medium. For example, a type **Temperature** is defined where the attributes **min** and **max** define the validity region of the medium, and a suitable default start value is given. In a device model, it is advisable to use these type definitions, e.g., for parameters, in order that medium limits are checked as early as possible, and that iteration variables of non-linear systems of equations get reasonable start values.

Note, although we use the term **medium model**, it is actually a Modelica **package** that contains all the constants and definitions required for a complete **medium model**. The basic interface to a medium is defined by Modelica.Media.Interfaces.PartialMedium that has the following structure:

```
partial package PartialMedium
  import SI = Modelica.SIunits;
  constant String           mediumName = "";
  constant String           substanceNames[:] = {mediumName};
  constant String           extraPropertiesNames[:] = fill("",0);
  constant Boolean          singleState = false;
  constant Boolean          reducedX = true;
  constant Boolean          fixedX = false;
  constant AbsolutePressure reference_p = 101325;
  constant MassFraction     reference_X[nX]=fill(1/nX,nX);
  constant AbsolutePressure p_default = 101325;
  constant Temperature      T_default =
Modelica.SIunits.Conversions.from_degC(20);
  constant SpecificEnthalpy h_default =
                            specificEnthalpy_pTX(p_default, T_default,
X_default);
  constant MassFraction     X_default[nX]=reference_X;
  final constant Integer    nS  = size(substanceNames,1);
  final constant Integer    nX  = if nS==1 then 0 else nS;
  final constant Integer    nXi = if fixedX then 0
                                  else if reducedX then nS-1 else nS;
  final constant Integer    nC  = size(extraPropertiesNames,1);
  constant FluidConstants[nS] fluidConstants;

  replaceable record BasePropertiesRecord
    AbsolutePressure p;
    Density d;
    Temperature T;
```

```
    SpecificEnthalpy h;
    SpecificInternalEnergy u;
    MassFraction[nX] X;
    MassFraction[nXi] Xi;
    SpecificHeatCapacity R;
    MolarMass MM;
  end BasePropertiesRecord;

  replaceable partial model BaseProperties
    extends BasePropertiesRecord;
    ThermodynamicState state;
    parameter Boolean preferredMediumStates=false;
    SI.Conversions.NonSIunits.Temperature_degC T_degC =
       Modelica.SIunits.Conversions.to_degC(T)
    SI.Conversions.NonSIunits.Pressure_bar p_bar =
       Modelica.SIunits.Conversions.to_bar(p)
  equation
    Xi = X[1:nXi];
    if nX > 1 then
       if fixedX then
          X = reference_X;
       elseif reducedX then
          X[nX] = 1 - sum(Xi);
       end if;
    end if;
    // equations such as
    //    d = d(p,T);
    //    u = u(p,T);
    //    h = u + p/d;
    //    state.p = p;
    //    state.T = T;
    // will go here in actual media implementations, but are not present
    // in the base class since the ThermodynamicState record is still empty
   end BaseProperties

  replaceable record ThermodynamicState
     // there are no "standard" thermodynamic variables in the base class
     // but they will be defined here in actual media extending PartialMedium
     // Example:
     //    AbsolutePressure p "Absolute pressure of medium";
     //    Temperature      T "Temperature of medium";
  end ThermodynamicState;

  // optional medium properties
  replaceable partial function dynamicViscosity
    input  ThermodynamicState state;
    output DynamicViscosity eta;
  end dynamicViscosity;

  // other optional functions

  // medium specific types
  type AbsolutePressure = SI.AbsolutePressure (
                              min     = 0,
                              max     = 1.e8,
                              nominal = 1.e5,
                              start   = 1.e5);
  type DynamicViscosity = ...;
  // other type definitions
```

```
end PartialMedium;
```

We will discuss all parts of this package in the following paragraphs. An actual medium model should extend from PartialMedium and has to provide implementations of the various parts.

Some of the constants at the beginning of the package do not have a value yet (this is valid in Modelica), but a value has to be provided when extending from package PartialMedium. A given value can be modified until the model is translated or the **final** prefix is set. The reason to use constants instead of parameters in the model BaseProperties is that some of these constants are used in a context where parameters are not allowed. For example, in connector definitions the number of independent mass fractions nXi is used as dimension of a vector Xi. When defining the connector, only *constants* in packages can be accessed, but not *parameters* in a model, because a connector cannot contain an instance of BaseProperties.

The record BasePropertiesRecord contains the variables primarily used in balance equations. Three equations for these variables have to be provided by every medium in model BaseProperties, plus two equations for the gas constant and the molar mass.

Optional medium properties are defined by functions, such as the function dynamicViscosity (see code Section above) to compute the dynamic viscosity. The argument of those functions is the ThermodynamicState record, defined in BaseProperties, which contains the minimum number of thermodynamic variables needed as an input to compute all the optional properties. This construction simplifies the usage considerably as demonstrated in the following code fragment:

```
replaceable package Medium = Modelica.Media.Interfaces.PartialMedium;
Medium.BaseProperties   medium;
Medium.DynamicViscosity eta;
...
U   = m*medium.u; //Internal energy
eta = Medium.dynamicViscosity(medium.state);
```

Medium is the medium package that satisfies the requirements of a PartialMedium (when using the model above, a value for Medium has to be provided by a redeclaration). The medium component is an instance of the model Medium.BaseProperties and contains the core medium equations. Variables in this model can be accessed just by dot-notation, such as medium.u or medium.T. If an optional medium variable has to be computed, the corresponding function from the actual Medium package is called, such as Medium.dynamicViscosity. The medium.state vector can be given as input argument to this function, and its fields are kept consistent to those of BaseProperties by suitable equations, contained in BaseProperties itself (see above).

If a medium model does not provide implementations of all optional functions and one of these functions is called in a model, an error occurs during translation since the optional functions which have not been redeclared have the *partial* attribute. For example, if function dynamicViscosity is not provided in the medium model when it is used, only simple pressure drop loss models without a reference to the viscosity can be used and not the sophisticated ones.

At the bottom of the PartialMedium package type declarations are present, that are used in all other parts of the PartialMedium package and that should be used in all models and connectors where a medium model is accessed. The reason is that minimum, maximum, nominal, and start values are defined and these values can be adapted to the particular medium at hand. For example, the nominal value of AbsolutePressure is 10^5 Pa. If a simple model of water steam is used that is only valid above 100 °C, then the minimum value in the Temperature type should be set to this value. The minimum and maximum values are also important for parameters in order to get an early message if data outside of the validity region is given. The nominal attribute is important as a scaling value if the variable is used as a state in a differential equation or as an iteration variable in a non-linear system of equations. The start attribute can be very useful to provide a meaningful default start or guess value if the variable is used, e.g., as iteration variable in a non-linear system of equations. Note, that all these attributes can be set specifically for a medium in the following way:

```
package MyMedium
  extends Modelica.Media.Interfaces.PartialMedium(
    ...
    Temperature(min=373));
```

```
end MyMedium;
```

The type PartialMedium.MassFlowRate is defined as

```
type MassFlowRate = Modelica.SIunits.MassFlowRate
     (quantity = "MassFlowRate." + mediumName);
```

Note that the constant mediumName, that has to be defined in every medium model, is used in the quantity attribute. For example, if mediumName = SimpleLiquidWater, then the quantity attribute has the value MassFlowRate.SimpleLiquidWater. This type should be used in a connector definition of a fluid library:

```
connector FluidPort
  replaceable package Medium = Modelica.Media.Interfaces.PartialMedium;
  flow Medium.MassFlowRate m_flow;
  ...
end FluidPort;
```

In the model where this connector is used, the actual Medium has to be defined. Connectors can only be connected together, if the corresponding attributes are either not defined or have identical values. Since mediumName is part of the quantity attribute of MassFlowRate, it is not possible to connect connectors with different media models together. In Dymola this is already checked when models are connected together in the diagram layer of the graphical user interface.

Modelica.Media.UsersGuide.MediumDefinition.BasicDefinition

Basic definition

Let's now walk through the definition of a new medium model. Please refer to Modelica.Media.Interfaces.TemplateMedium to obtain a template of the new medium model code. For the moment being, consider a single-substance medium model.

The new medium model is obtained by extending Modelica.Media.Interfaces.PartialMedium, and setting the following package constants:

- mediumName is a String containing the name of the medium.
- substancesNames is a vector of strings containing the names of the substances that make up the medium. In this case, it will contain only mediumName.
- singleState can be set to true if u and d in BaseProperties do not depend on pressure. In other words, density does not depend on pressure (incompressible fluid), and it is assumed that also u does not depend on pressure. This setting can be useful for fluids having high density and low compressibility (e.g., liquids at moderate pressure); fast states resulting from the low compressibility effects are automatically avoided.
- reducedX = true for single-substance media, which do not need mass fractions at all.

It is also possible to change the default min, max, nominal, and start attributes of Medium-defined types (see TemplateMedium).

All other package constants, such as nX, nXi, nS, are automatically set by the declarations of the base package Interfaces.PartialMedium.

The second step is to provide an implementation to the BaseProperties model, partially defined in the base class Interfaces.PartialMedium. In the case of single-substance media, two independent state variables must be selected among p, T, d, u, h, and three equations must be written to provide the values of the remaining variables. Two equations must then be added to compute the molar mass MM and the gas constant R.

The third step is to consider the optional functions that are going to be implemented, among the partial functions defined by the base class PartialMedium. A minimal set of state variables that could be provided as an input to *all* those functions must be selected, and included in the redeclaration of the ThermodynamicState record. Subsequently, equations must be added to BaseProperties in order that the instance of that record inside BaseProperties (named "state") is kept updated. For example, assume that all

additional properties can be computed as a function of p and T. Then, ThermodynamicState should be redclared as follows:

```
redeclare replaceable record ThermodynamicState
  AbsolutePressure p "Absolute pressure of medium";
  Temperature T "Temperature of medium";
end ThermodynamicState;
```

and the following equations should be added to BaseProperties:

```
state.p = p;
state.T = T;
```

The additional functions can now be implemented by redeclaring the functions defined in the base class and adding their algorithms, e.g.:

```
redeclare function extends dynamicViscosity "Return dynamic viscosity"
algorithm
  eta := 10 - state.T*0.3 + state.p*0.2;
end dynamicViscosity;
```

Modelica.Media.UsersGuide.MediumDefinition.MultipleSubstances

Multiple Substances

When writing the model of a multiple-substance medium, a fundamental issue concerns how to consider the mass fractions of the fluid. If there are nS substances, there are also nS mass fractions; however, one of them is redundant, as sum(X) = 1. Therefore there are basically two options, concerning the number of independent mass fractions nXi:

- *Reduced-state models*: reducedX = **true** and nXi = nS - 1. In this case, the number of independent mass fractions nXi is the minimum possible. The full state vector X is provided by equations declared in the base class Interfaces.PartialMedium.BaseProperties: the first nXi elements are equal to Xi, and the last one is 1 - sum(Xi).
- *Full-state models*: reducedX = **false** and nXi = nS. In this case, Xi = X, i.e., all the elements of the composition vector are considered as independent variables, and the constraint sum(X) = 1 is never written explicitly. Although this kind of model is heavier, as it provides one extra state variable, it can be less prone to numerical and/or symbolic problems, which can be caused by that constraint.
- *Fixed-composition models*: fixedX = **true** and nXi = 0. In this case X = reference_X, i.e. all the elements of the composition vector are fixed.

The medium implementor can declare the value reducedX as **final**. In this way only one implementation must be given. For instance, Modelica.Media.IdealGases models declare **final** reducedX = **false**, so that the implementation can always assume nXi = nX. The same is true for Air.MoistAir, which declares **final** reducedX = **true**, and always assumes nXi = nX - 1 = 1.

It is also possible to leave reducedX modifiable. In this case, the BaseProperties model and all additional functions should check for the actual value of reducedX, and provide the corresponding implementation.

If fixedX is left modifiable, then the implementation should also handle the case fixedX = true properly.

Fluid connectors should always use composition vectors of size Xi, such as in the Modelica_Fluid library:

```
connector FluidPort
  replaceable package Medium = Modelica.Media.Interfaces.PartialMedium;
  Medium.AbsolutePressure      p;
  flow Medium.MassFlowRate     m_flow;

  Medium.SpecificEnthalpy      h;
  flow Medium.EnthalpyFlowRate H_flow;
```

```
  Medium.MassFraction           Xi     [Medium.nXi];
  flow Medium.MassFlowRate      mX_flow[Medium.nXi];
end FluidPort;
```

For further details, refer to the implementation of MixtureGasNasa model and MoistAir model.

Modelica.Media.UsersGuide.MediumDefinition.SpecificEnthalpyAsFunction

Specific enthalpy as function

If pressure p and specific enthalpy h are **not** used as independent medium variables, the specific enthalpy should be computed by a Modelica function that has as input arguments only the independent medium variables. It should **not** be computed by an equation. For example, if p and T are used as independent medium variables, a function h_pT(p,T) should be defined that is called to compute h:

```
h = h_pT(p,T);
```

The reason for this rule requires a longer explanation. In short, if h is not a computed by a Modelica function and this function is non-linear in the independent medium variables, then non-linear systems of equations will occur at every connection point, if the FluidPort connectors from the Modelica_Fluid library are used (these are the same as in Modelica.Media.Examples.Tests.Components.FluidPort). Only, if the above rule is fulfilled, a tool is able to remove these non-linear system of equations in most cases.

The basic idea of the FluidPort connector is that 2 or more components can be connected together at a point and that automatically the mass and energy balance is fulfilled in the connection point, i.e., the ideal mixing equations are generated. Note, the momentum balance is only correct for straight line connections. If "ideal mixing" is not sufficient, a special component to define the mixing equations must be introduced.

The mass and momentum balance equations in a component are derived from the partial differential equations along the flow direction of a pipe:

$$\frac{\partial(\rho A)}{\partial t} + \frac{\partial(\rho A v)}{\partial x} = 0$$

$$\frac{\partial(\rho v A)}{\partial t} + \frac{\partial(\rho v^2 A)}{\partial x} = -A\frac{\partial p}{\partial x} - F_F - A\rho g\frac{\partial z}{\partial x}$$

$$F_F = \frac{1}{2}\rho v|v| fS$$

Note, F_F is the fanning friction factor. The energy balance can be given in different forms. Usually, it is given as:

$$\frac{\partial(\rho(u+\frac{v^2}{2})A)}{\partial t} + \frac{\partial(\rho v(u+\frac{p}{\rho}+\frac{v^2}{2})A)}{\partial x} = -A\rho v g\frac{\partial z}{\partial x} + \frac{\partial}{\partial x}(kA\frac{\partial T}{\partial x}) + \dot{Q}_s$$

This form describes the change of the internal energy, kinetic energy and potential energy of a volume as function of the in and out flowing fluid. Multiplying the momentum balance with the flow velocity v and subtracting it from the energy balance above, results in the following alternative form of the energy balance:

$$\frac{\partial(\rho u A)}{\partial t} + \frac{\partial(\rho v (u + \frac{p}{\rho}) A)}{\partial x} = vA\frac{\partial p}{\partial x} + vF_F + \frac{\partial}{\partial x}(kA\frac{\partial T}{\partial x}) + \dot{Q}_e$$

This form has the advantage that the kinetic and potential energy is no longer part of the energy balance and therefore the energy balance is substantially simpler (e.g., additional non-linear systems of equations occur in the first form since the velocity is present in the energy balance; in the second form this is not the case and it is still valid also for high speeds).

Assume now that the second form of the energy balance above is used in all components and that the following FluidPort connector is used in all components:

```
connector FluidPort
  replaceable package Medium = Modelica.Media.Interfaces.PartialMedium;
  Medium.AbsolutePressure      p;
  flow Medium.MassFlowRate     m_flow;

  Medium.SpecificEnthalpy      h;
  flow Medium.EnthalpyFlowRate H_flow;

  Medium.MassFraction          Xi    [Medium.nXi];
  flow Medium.MassFlowRate     mX_flow[Medium.nXi];
end FluidPort;
```

As an example, assume that 3 components are connected together and that the medium is a single substance fluid. This will result in the following connection equations:

```
p1=p2=p3;
h1=h2=h3;
0 = m_flow1 + m_flow2 + m_flow3;
0 = H_flow1 + H_flow2 + H_flow3;
```

These are the mass balance and the energy balance (form 2) of an infinitesimal volume in the connection point under the assumption that no mass or energy is stored in this volume. In other words, the connection equations are the equations that describe ideal mixing. Under the assumption that the velocity vectors of the 3 flows are identical (especially, they are parallel), also the momentum balance is fulfilled:

```
0 = m_flow1*v1 + m_flow2*v2 + m_flow3*v3;
  = v*(m_flow1 + m_flow2 + m_flow3);
  = 0;
```

With the above connector it is therefore possible to connect components together in a nearly arbitrary fashion, because every connection fulfills automatically the balance equations. This approach has, however, one drawback: If two components are connected together, then the medium variables on both sides of the connector are identical. However, due to the connector, only the two equations

```
p1 = p2;
h1 = h2;
```

are present. Assume, that p,T are the independent medium variables and that the medium properties are computed at one side of the connections. This means, the following equations are basically present:

```
h1 = h(p1,T1);
h2 = h(p2,T2);
p1 = p2;
h1 = h2;
```

These equations can be solved in the following way:

```
h1 := h(p1,T1)
p2 := p1;
h2 := h1;
0  := h2 - h(p2,T2);   // non-linear system of equations for T2
```

This means that T2 is computed by solving a non-linear system of equations. If h1 and h2 are provided as Modelica functions, a Modelica translator, such as Dymola, can replace this non-linear system of equations by the equation:

```
T2 := T1;
```

because after alias substition there are two function calls

```
h1 := h(p1,T1);
h1 := h(p1,T2);
```

Since the left hand side of the function call and the first argument are the same, the second arguments T1 and T2 must also be identical and therefore T2 := T1. This type of analysis seems to be only possible, if the specific enthalpy is defined as a function of the independent medium variables.

Modelica.Media.UsersGuide.MediumDefinition.StaticStateSelection

Static State Selection

Without pre-caution when implementing a medium model, it is very easy that non-linear algebraic systems of equations occur when using the medium model. In this section it is explained how to avoid non-linear systems of equations that result from unnecessary dynamic state selections.

A medium model should be implemented in such a way that a tool is able to select states of a medium in a balance volume statically (during translation). This is only possible if the medium equations are written in a specific way. Otherwise, a tool has to dynamically select states during simulation. Since medium equations are usually non-linear, this means that non-linear algebraic systems of equations would occur in every balance volume.

It is assumed that medium equations in a balance volume are defined in the following way:

```
  package Medium = Modelica.Media.Interfaces.PartialMedium;
  Medium.BaseProperties medium;
equation
  // mass balance
    der(M)  = port_a.m_flow + port_b.m_flow;
    der(MX) = port_a_mX_flow + port_b_mX_flow;
          M = V*medium.d;
         MX = M*medium.X;

  // Energy balance
  U = M*medium.u;
  der(U) = port_a.H_flow+port_b.H_flow;
```

Single Substance Media

A medium consisting of a single substance has to define two of "p,T,d,u,h" with stateSelect=StateSelect.prefer if BaseProperties.preferredMediumstates = **true** and has to provide the other three variables as function of these states. This results in:

- static state selection (no dynamic choices).
- a linear system of equations in the two state derivatives.

Example for a single substance medium

p, T are preferred states (i.e. StateSelect.prefer is set) and there are three equations written in the form:

```
d = fd(p,T)
u = fu(p,T)
h = fh(p,T)
```

Index reduction leads to the equations:

```
der(M) = V*der(d)
der(U) = der(M)*u + M*der(u)
der(d) = der(fd,p)*der(p) + der(fd,T)*der(T)
der(u) = der(fu,p)*der(p) + der(fu,T)*der(T)
```

Note, that **der**(y,x) is the partial derivative of y with respect to x and that this operator will be introduced in a future version of the Modelica language. The above equations imply, that if p,T are provided from the integrator as states, all functions, such as fd(p,T) or **der**(fd,p) can be evaluated as function of the states. The overall system results in a linear system of equations in **der**(p) and **der**(T) after eliminating **der**(M), **der**(U), **der**(d), **der**(u) via tearing.

Counter Example for a single substance medium

An ideal gas with one substance is written in the form

```
  redeclare model extends BaseProperties(
     T(stateSelect=if preferredMediumStates then StateSelect.prefer else
StateSelect.default),
     p(stateSelect=if preferredMediumStates then StateSelect.prefer else
StateSelect.default)
  equation
     h = h(T);
     u = h - R*T;
     p = d*R*T;
      ...
  end BaseProperties;
```

If p, T are preferred states, these equations are **not** written in the recommended form, because d is not a function of p and T. If p,T would be states, it would be necessary to solve for the density:

```
d = p/(R*T)
```

If T or R are zero, this results in a division by zero. A tool does not know that R or T cannot become zero. Therefore, a tool must assume that p, T **cannot** always be selected as states and has to either use another static state selection or use dynamic state selection. The only other choice for static state selection is d,T, because h,u,p are given as functions of d,T. However, as potential states only variables appearing differentiated and variables declared with StateSelect.prefer or StateSelect.always are used. Since "d" does not appear differentiated and has StateSelect.default, it cannot be selected as a state. As a result, the tool has to select states dynamically during simulation. Since the equations above are non-linear and they are utilized in the dynamic state selection, a non-linear system of equations is present in every balance volume.

To summarize, for single substance ideal gas media there are the following two possibilities to get static state selection and linear systems of equations:

1. Use p,T as preferred states and write the equation for d in the form: d = p/(T*R)
2. Use d,T as preferred states and write the equation for p in the form: p = d*T*R

All other settings (other/no preferred states etc.) lead to dynamic state selection and non-linear systems of equations for a balance volume.

Multiple Substance Media

A medium consisting of multiple substance has to define two of "p,T,d,u,h" as well as the mass fractions Xi with stateSelect=StateSelect.prefer (if BaseProperties.preferredMediumStates = **true**) and has to provide the other three variables as functions of these states. Only then, static selection is possible for a tool.

Example for a multiple substance medium:

p, T and Xi are defined as preferred states and the equations are written in the form:

```
d = fp(p,T,Xi);
u = fu(p,T,Xi);
h = fh(p,T,Xi);
```

Since the balance equations are written in the form:

```
  M = V*medium.d;
MXi = M*medium.Xi;
```

The variables M and MXi appearing differentiated in the balance equations are provided as functions of d and Xi and since d is given as a function of p, T and Xi, it is possible to compute M and MXi directly from the desired states. This means that static state selection is possible.

Modelica.Media.UsersGuide.MediumDefinition.TestOfMedium

Test of medium

After implementation of a new medium model, it should be tested. A basic test is already provided with model Modelica.Media.Examples.Tests.Components.PartialTestModel which might be used in the following way:

```
model TestOfMyMedium
   extends Modelica.Media.Examples.Tests.Components.PartialTestModel(
             redeclare package Medium = MyMedium);
end TestOfMyMedium;
```

It might be necessary to adapt or change initial values depending on the validity range of the medium. The model above should translate and simulate. If the medium model is written according to the suggestions given in the previous sections (and the Modelica translator has appropriate algorithms implemented), there should be only static state selection everywhere and no non-linear system of equations, provided h is an independent medium variable or is only a function of T. If h is a function of, say h=h(p,T), one non-linear system of equations occurs that cannot be avoided.

The test model above can be used to test the most basic properties. Of course, more tests should be performed.

Modelica.Media.UsersGuide.ReleaseNotes

Release notes

Version 1.0, 2005-03-01

Many improvements in the library, e.g., providing mixtures of the ideal gases, table based media, test suite for all media, improved and updated User's Guide.

Version 0.9, 2004-10-18

- Changed the redeclaration/extends within packages from the experimental feature to the language keywords introduced in Modelica 2.1.
- Re-introduced package "Water.SaltWater" in order to test substance mixtures (this medium model does not describe real mixing of water and salt).
- Started to improve the documentation in Modelica.Media.UsersGuide.MediumDefinition.BasicStructure

Version 0.792, 2003-10-28

This is the first version made available for the public for the Modelica'2003 conference (for evaluation).

Modelica.Media.UsersGuide.Contact

Contact

Main author and maintainer:
Hubertus Tummescheit
Modelon AB
Ideon Science Park
SE-22730 Lund, Sweden
email: Hubertus.Tummescheit@Modelon.se

Acknowledgements:

The development of this library has been a collaborative effort and many have contributed:

- The essential parts of the media models have been implemented in the ThermoFluid library by Hubertus Tummescheit with help from Jonas Eborn and Falko Jens Wagner. These media models have been converted to the Modelica.Media interface definition and have been improved by Hubertus Tummescheit.
- The effort for the development of the Modelica.Media library has been organized by Martin Otter who also contributed to the design, implemented part of the generic models, contributed to the User's Guide and provided the generic test suite Modelica.Media.Examples.Tests.
- The basic idea for the medium model interface based on packages is from Michael Tiller who also contributed to the design.
- The first design of the medium model interface is from Hilding Elmqvist. The design and the implementation has been further improved at the Modelica design meetings in
  Dearborn, Nov. 20-22, 2002
  Dearborn, Sept. 2-4, 2003
  Lund Jan. 28-30, 2004
  Munich, May 26-28, 2004
  Lund, Aug. 30-31, 2004
  Dearborn, Nov. 15-17, 2004
  Cremona Jan. 31 - Feb. 2, 2005.
- Hans Olsson, Sven Erik Mattsson and Hilding Elmqvist developed symbolic transformation algorithms and implemented them in Dymola to improve the efficiency considerably (e.g., to avoid non-linear systems of equations).
- Katrin Pröß implemented the moist air model
- Rüdiger Franke performed the first realistic tests of the Modelica.Media and Modelica_Fluid libraries and gave valuable feedback.
- Francesco Casella has been the most relentless bug-hunter and tester of the water and ideal gas properties. He also contributed to the User's Guide.
- John Batteh, Daniel Bouskela, Jonas Eborn, Andreas Idebrant, Charles Newman, Gerhart Schmitz, and the users of the ThermoFluid library provided many useful comments and feedback.

Modelica.Media.Examples

Demonstrate usage of property models (currently: simple tests)

Information

Examples

Physical properties for fluids are needed in so many different variants that a library can only provide models for the most common situations. With the following examples we are going to demonstrate how to use the existing packages and functions in Modelica.Media to customize these models for advanced applications. The high level functions try to abstract as much as possible form the fact that different media are based on different variables, e.g. ideal gases need pressure and temperature, while many refrigerants are based on Helmholtz functions of density and temperature, and many water proeprties are based on pressure and specific enthalpy. Medium properties are needed in control volumes in the dynamic state equations and in many thermodynamic state locations that are independent of the dynamic states of a control volume, e.g. at a wall temperature, an isentropic reference state or at a phase boundary. The general structure of the library is such that:

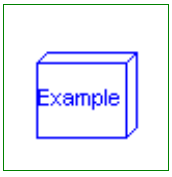
- Each medium has a model called BaseProperties. BaseProperties contains the minimum set of medium properties needed in a dynamic control volume model.
- Each instance of BaseProperties contains a "state" record that is an input to all the functions to compute properties. If these functions need further inputs, like e.g. the molarMass, these are accessible as constants in the package.
- The simplest way to compute properties at any other reference point is to declare an instance of ThermodynamicState and use that as input to arbitrary property functions.

A small library of generic volume, pipe, pump and ambient models is provided in Modelica.Media.Examples.Tests.Components to demonstrate how fluid components should be implemented that are using Modelica.Media models. This library is also used to test all media models in Modelica.Media.Examples.Tests.MediaTestModels.

Package Content

Name	Description
SimpleLiquidWater	Example for Water.SimpleLiquidWater medium model
IdealGasH2O	IdealGas H20 medium model
WaterIF97	WaterIF97 medium model
MixtureGases	Test gas mixtures
MoistAir	Ideal gas flue gas model
TwoPhaseWater	extension of the StandardWater package
TestOnly	examples for testing purposes: move for final version
Tests	Library to test that all media models simulate and fulfill the expected structural properties
SolveOneNonlinearEquation	Demonstrate how to solve one non-linear algebraic equation in one unknown

Modelica.Media.Examples.SimpleLiquidWater



Example for Water.SimpleLiquidWater medium model

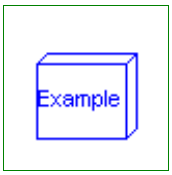
Information

Parameters

Type	Name	Default	Description

Volume	V	1	Volume [m3]
EnthalpyFlowRate	H_flow_ext	1.e6	Constant enthalpy flow rate into the volume [W]

Modelica.Media.Examples.IdealGasH2O

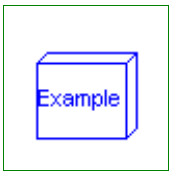


IdealGas H20 medium model

Information

An example for using ideal gas properties and how to compute isentropic enthalpy changes. The function that is implemented is approximate, but usually very good: the second medium record medium2 is given to compare the approximation.

Modelica.Media.Examples.WaterIF97



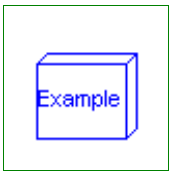
WaterIF97 medium model

Information

Parameters

Type	Name	Default	Description
VolumeFlowRate	dV	0.0	[m3/s]
MassFlowRate	m_flow_ext	0	[kg/s]
EnthalpyFlowRate	H_flow_ext	10000	[W]

Modelica.Media.Examples.MixtureGases



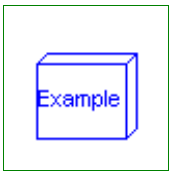
Test gas mixtures

Information

Parameters

Type	Name	Default	Description
Real	V	1	
Real	m_flow_ext	0.01	
Real	H_flow_ext	5000	

Modelica.Media.Examples.MoistAir



Ideal gas flue gas model

Information

An example for using ideal gas properties and how to compute isentropic enthalpy changes. The function that is implemented is approximate, but usually very good: the second medium record medium2 is given to compare the approximation.

Parameters

Type	Name	Default	Description
Real	MMx[2]	{Medium.dryair.MM,Medium.ste...	

Modelica.Media.Examples.TwoPhaseWater

extension of the StandardWater package

Information

Example: TwoPhaseWater

The TwoPhaseWater package demonstrates how to extend the parsimonius BaseProperties with a minimal set of properties from the standard water package with most properties that are needed in two-phase situations. The model also demonstrates how to compute additional properties for the medium model. In this scenario, that builds a new medium model with many more properties than the default, the standard BaseProperties is used as a basis. For additional properties, a user has to:

1. Declare a new variable of the wanted type, e.g. "DynamicViscosity eta".
2. Compute that variable by calling the function form the package, e.g. eta = dynamicViscosity(state). Note that the instance of ThermodynamicState is used as an input to the function. This instance "state" is declared in PartialMedium and thus available in every medium model. A user does not have to know what actual variables are required to compute the dynamic viscosity, because the state instance is guaranteed to contain what is needed.
3. Attention: Many properties are not well defined in the two phase region and the functions might return undesired values if called there. It is the user's responsibility to take care of such ituations. The example uses one of several possible models to compute an averaged viscosity for two-phase flows.

In two phase models, properties are often needed on the phase boundary just outside the two phase dome, right on the border.. To compute the thermodynamic state there, two auxiliary functions are provided: **setDewState(sat)** and **setBubbleState(sat)**. They take an instance of SaturationProperties as input. By default they are in one-phase, but with the optional phase argument set to 2, the output is forced to be just inside the phase boundary. This is only needed when derivatives like cv are computed with are different on both sides of the boundaries. The ususal steps to compute properties on the phase boundary are:

1. Declare an instance of ThermodynamicState, e.g. "ThermodynamicState dew".
2. Compute the state, using an instance of SaturationProperties, e.g. dew = setDewState(sat)
3. Compute properties on the phase boundary to your full desire, e.g. "cp_d = specificHeatCapacityCp(dew)".

The sample model TestTwoPhaseStates test the extended properties

The same procedure can be used to compute properties at other state points, e.g. when an isentropic reference state is computed.

Package Content

Name	Description
ExtendedProperties	plenty of two-phase properties
TestTwoPhaseStates	Test the above model
Inherited	
ThermodynamicState	thermodynamic state
ph_explicit	true if explicit in pressure and specific enthalpy

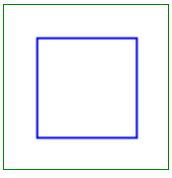
dT_explicit	true if explicit in density and temperature
pT_explicit	true if explicit in pressure and temperature
BaseProperties	Base properties of water
density_ph	Computes density as a function of pressure and specific enthalpy
temperature_ph	Computes temperature as a function of pressure and specific enthalpy
temperature_ps	Compute temperature from pressure and specific enthalpy
density_ps	Computes density as a function of pressure and specific enthalpy
pressure_dT	Computes pressure as a function of density and temperature
specificEnthalpy_dT	Computes specific enthalpy as a function of density and temperature
specificEnthalpy_pT	Computes specific enthalpy as a function of pressure and temperature
specificEnthalpy_ps	Computes specific enthalpy as a function of pressure and temperature
density_pT	Computes density as a function of pressure and temperature
setDewState	set the thermodynamic state on the dew line
setBubbleState	set the thermodynamic state on the bubble line
dynamicViscosity	Dynamic viscosity of water
thermalConductivity	Thermal conductivity of water
surfaceTension	Surface tension in two phase region of water
pressure	return pressure of ideal gas
temperature	return temperature of ideal gas
density	return density of ideal gas
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificEntropy	specific entropy of water
specificHeatCapacityCp	specific heat capacity at constant pressure of water
specificHeatCapacityCv	specific heat capacity at constant volume of water
isentropicExponent	Return isentropic exponent
isothermalCompressibility	Isothermal compressibility of water
isobaricExpansionCoefficient	isobaric expansion coefficient of water
velocityOfSound	
isentropicEnthalpy	compute h(p,s)
density_derh_p	density derivative by specific enthalpy
density_derp_h	density derivative by pressure

bubbleEnthalpy	boiling curve specific enthalpy of water
dewEnthalpy	dew curve specific enthalpy of water
bubbleEntropy	boiling curve specific entropy of water
dewEntropy	dew curve specific entropy of water
bubbleDensity	boiling curve specific density of water
dewDensity	dew curve specific density of water
saturationTemperature	saturation temperature of water
saturationTemperature_derp	derivative of saturation temperature w.r.t. pressure
saturationPressure	saturation pressure of water
dBubbleDensity_dPressure	bubble point density derivative
dDewDensity_dPressure	dew point density derivative
dBubbleEnthalpy_dPressure	bubble point specific enthalpy derivative
dDewEnthalpy_dPressure	dew point specific enthalpy derivative
setState_dTX	
setState_phX	
setState_psX	
setState_pTX	
smoothModel	true if the (derived) model should not generate state events
onePhase	true if the (derived) model should never be called with two-phase inputs
FluidLimits	validity limits for fluid model
FluidConstants	extended fluid constants
fluidConstants	constant data for the fluid
SaturationProperties	Saturation properties of two phase medium
FixedPhase	phase of the fluid: 1 for 1-phase, 2 for two-phase, 0 for not known, e.g. interactive use
setSat_T	Return saturation property record from temperature
setSat_p	Return saturation property record from pressure
saturationPressure_sat	Return saturation temperature
saturationTemperature_sat	Return saturation temperature
saturationTemperature_derp_sat	Return derivative of saturation temperature w.r.t. pressure
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from pressure, temperature and mass fraction
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p, s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi

setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
setState_px	Return thermodynamic state from pressure and vapour quality
setState_Tx	Return thermodynamic state from temperature and vapour quality
vapourQuality	Return vapour quality
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
density_pTX	Return density from p, T, and X or Xi

AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Modelica.Media.Examples.TwoPhaseWater.ExtendedProperties



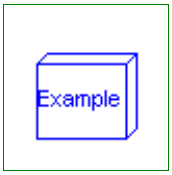
plenty of two-phase properties

Parameters

Type	Name	Default	Description

Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Initialization			
Integer	phase.start	1	2 for two-phase, 1 for one-phase, 0 if not known
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

Modelica.Media.Examples.TwoPhaseWater.TestTwoPhaseStates



Test the above model

Information

For details see the documentation of the example package TwoPhaseWater

Parameters

Type	Name	Default	Description
Real	dh	80000.0	80 kJ/second
Real	dp	1.0e6	10 bars per second

Modelica.Media.Examples.TestOnly

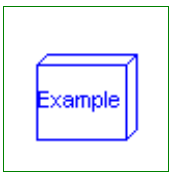
examples for testing purposes: move for final version

Information

Package Content

Name	Description
MixIdealGasAir	Ideal gas air medium model
FlueGas	Ideal gas flue gas model
IdealGasN2	Test IdealGas.SingleMedia.N2 medium model
TestMedia	

Modelica.Media.Examples.TestOnly.MixIdealGasAir

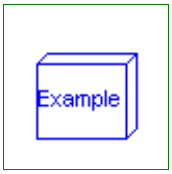


Ideal gas air medium model

Information

An example for using ideal gas properties and how to compute isentropic enthalpy changes. The function that is implemented is approximate, but usually very good: the second medium record medium2 is given to compare the approximation.

Modelica.Media.Examples.TestOnly.FlueGas



Ideal gas flue gas model

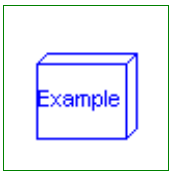
Information

An example for using ideal gas properties and how to compute isentropic enthalpy changes. The function that is implemented is approximate, but usually very good: the second medium record medium2 is given to compare the approximation.

Parameters

Type	Name	Default	Description
Real	MMx[4]	Medium.data.MM	

Modelica.Media.Examples.TestOnly.IdealGasN2



Test IdealGas.SingleMedia.N2 medium model

Information

Parameters

Type	Name	Default	Description
Real	V	1	
Real	m_flow_ext	0.01	
Real	H_flow_ext	5000	

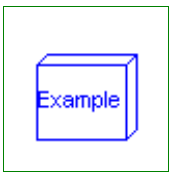
Modelica.Media.Examples.TestOnly.TestMedia

Information

Package Content

Name	Description
TemplateMedium	Test Interfaces.TemplateMedium

Modelica.Media.Examples.TestOnly.TestMedia.TemplateMedium



Test Interfaces.TemplateMedium

Information

Modelica.Media.Examples.Tests

Library to test that all media models simulate and fulfill the expected structural properties

Information

Package Content

Name	Description
Components	Functions, connectors and models needed for the media model tests
MediaTestModels	Test models to test all media

Modelica.Media.Examples.Tests.Components

Functions, connectors and models needed for the media model tests

Information

Package Content

Name	Description
FluidPort	Interface for quasi one-dimensional fluid flow in a piping network (incompressible or compressible, one or more phases, one or more substances)
FluidPort_a	Fluid connector with filled icon
FluidPort_b	Fluid connector with outlined icon
PortVolume	Fixed volume associated with a port by the finite volume method
FixedMassFlowRate	Ideal pump that produces a constant mass flow rate from a large reservoir at fixed temperature and mass fraction
FixedAmbient	Ambient pressure, temperature and mass fraction source
ShortPipe	Simple pressure loss in pipe
PartialTestModel	Basic test model to test a medium
PartialTestModel2	slightly larger test model to test a medium

---

Modelica.Media.Examples.Tests.Components.FluidPort

Interface for quasi one-dimensional fluid flow in a piping network (incompressible or compressible, one or more phases, one or more substances)

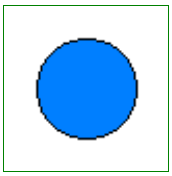
Information

Contents

Type	Name	Description
AbsolutePressure	p	Pressure in the connection point [Pa]
flow MassFlowRate	m_flow	Mass flow rate from the connection point into the component [kg/s]
SpecificEnthalpy	h	Specific mixture enthalpy in the connection point [J/kg]
flow EnthalpyFlowRate	H_flow	Enthalpy flow rate into the component (if m_flow > 0, H_flow = m_flow*h) [W]
MassFraction	Xi[Medium.nXi]	Independent mixture mass fractions m_i/m in the connection

		point [kg/kg]
flow MassFlowRate	mXi_flow[Medium.nXi]	Mass flow rates of the independent substances from the connection point into the component (if m_flow > 0, mX_flow = m_flow*X) [kg/s]
ExtraProperty	C[Medium.nC]	properties c_i/m in the connection point
flow ExtraPropertyFlowRate	mC_flow[Medium.nC]	Flow rates of auxiliary properties from the connection point into the component (if m_flow > 0, mC_flow = m_flow*C)

Modelica.Media.Examples.Tests.Components.FluidPort_a



Fluid connector with filled icon

Information

Modelica.Media.Examples.Tests.Components.FluidPort_a

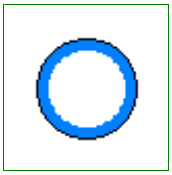
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model

Contents

Type	Name	Description
AbsolutePressure	p	Pressure in the connection point [Pa]
flow MassFlowRate	m_flow	Mass flow rate from the connection point into the component [kg/s]
SpecificEnthalpy	h	Specific mixture enthalpy in the connection point [J/kg]
flow EnthalpyFlowRate	H_flow	Enthalpy flow rate into the component (if m_flow > 0, H_flow = m_flow*h) [W]
MassFraction	Xi[Medium.nXi]	Independent mixture mass fractions m_i/m in the connection point [kg/kg]
flow MassFlowRate	mXi_flow[Medium.nXi]	Mass flow rates of the independent substances from the connection point into the component (if m_flow > 0, mX_flow = m_flow*X) [kg/s]
ExtraProperty	C[Medium.nC]	properties c_i/m in the connection point
flow ExtraPropertyFlowRate	mC_flow[Medium.nC]	Flow rates of auxiliary properties from the connection point into the component (if m_flow > 0, mC_flow = m_flow*C)

Modelica.Media.Examples.Tests.Components.FluidPort_b



Fluid connector with outlined icon

Information

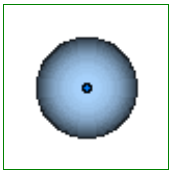
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model

Contents

Type	Name	Description
AbsolutePressure	p	Pressure in the connection point [Pa]
flow MassFlowRate	m_flow	Mass flow rate from the connection point into the component [kg/s]
SpecificEnthalpy	h	Specific mixture enthalpy in the connection point [J/kg]
flow EnthalpyFlowRate	H_flow	Enthalpy flow rate into the component (if m_flow > 0, H_flow = m_flow*h) [W]
MassFraction	Xi[Medium.nXi]	Independent mixture mass fractions m_i/m in the connection point [kg/kg]
flow MassFlowRate	mXi_flow[Medium.nXi]	Mass flow rates of the independent substances from the connection point into the component (if m_flow > 0, mX_flow = m_flow*X) [kg/s]
ExtraProperty	C[Medium.nC]	properties c_i/m in the connection point
flow ExtraPropertyFlowRate	mC_flow[Medium.nC]	Flow rates of auxiliary properties from the connection point into the component (if m_flow > 0, mC_flow = m_flow*C)

Modelica.Media.Examples.Tests.Components.PortVolume



Fixed volume associated with a port by the finite volume method

Information

This component models the **volume** of **fixed size** that is associated with the **fluid port** to which it is connected. This means that all medium properties inside the volume, are identical to the port medium properties. In particular, the specific enthalpy inside the volume (= medium.h) is always identical to the specific enthalpy in the port (port.h = medium.h). Usually, this model is used when discretizing a component according to the finite volume method into volumes in internal ports that only store energy and mass and into transport elements that just transport energy, mass and momentum between the internal ports without storing these quantities during the transport.

Parameters

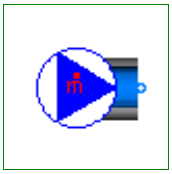
Type	Name	Default	Description
Volume	V	1e-6	Fixed size of junction volume [m3]
Initial pressure or initial density			
Boolean	use_p_start	true	select p_start or d_start
AbsolutePressure	p_start	101325	Initial pressure [Pa]
Density	d_start	1	Initial density [kg/m3]
Initial temperature or initial specific enthalpy			
Boolean	use_T_start	true	select T_start or h_start
Temperature	T_start	Modelica.SIunits.Conversions...	Initial temperature [K]
SpecificEnthalpy	h_start	1.e4	Initial specific enthalpy [J/kg]
Only for multi-substance flow			
MassFraction	X_start[Medium.nX]		Initial mass fractions m_i/m [kg/kg]

Connectors

Type	Name	Description

FluidPort_a	port	

Modelica.Media.Examples.Tests.Components.FixedMassFlowRate



Ideal pump that produces a constant mass flow rate from a large reservoir at fixed temperature and mass fraction

Information

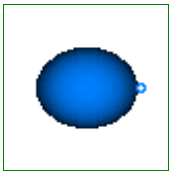
Parameters

Type	Name	Default	Description
MassFlowRate	m_flow		Fixed mass flow rate from an infinite reservoir to the fluid port [kg/s]
MassFraction	X_ambient[Medium.nX]		Ambient mass fractions m_i/m of reservoir [kg/kg]
Ambient temperature or ambient specific enthalpy			
Boolean	use_T_ambient	true	select T_ambient or h_ambient
Temperature	T_ambient	Modelica.SIunits.Conversions..	Ambient temperature [K]
SpecificEnthalpy	h_ambient	1.e4	Ambient specific enthalpy [J/kg]

Connectors

Type	Name	Description
FluidPort_b	port	

Modelica.Media.Examples.Tests.Components.FixedAmbient



Ambient pressure, temperature and mass fraction source

Information

Model **FixedAmbient_pt** defines constant values for ambient conditions:

- Ambient pressure.
- Ambient temperature.
- Ambient mass fractions (only for multi-substance flow).

Note, that ambient temperature and mass fractions have only an effect if the mass flow is from the ambient into the port. If mass is flowing from the port into the ambient, the ambient definitions, with exception of ambient pressure, do not have an effect.

Parameters

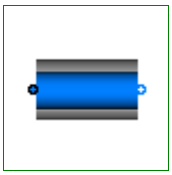
Type	Name	Default	Description
Ambient pressure or ambient density			
Boolean	use_p_ambient	true	select p_ambient or d_ambient
AbsolutePressure	p_ambient	101325	Ambient pressure [Pa]
Density	d_ambient	1	Ambient density [kg/m3]
Ambient temperature or ambient specific enthalpy			

Boolean	use_T_ambient	true	select T_ambient or h_ambient
Temperature	T_ambient	Modelica.SIunits.Conversions...	Ambient temperature [K]
SpecificEnthalpy	h_ambient	1.e4	Ambient specific enthalpy [J/kg]
Only for multi-substance flow			
MassFraction	X_ambient[Medium.nX]		Ambient mass fractions m_i/m [kg/kg]

Connectors

Type	Name	Description
FluidPort_b	port	

Modelica.Media.Examples.Tests.Components.ShortPipe



Simple pressure loss in pipe

Information

Model **ShortPipe** defines a simple pipe model with pressure loss due to friction. It is assumed that no mass or energy is stored in the pipe. The details of the pipe friction model are described here.

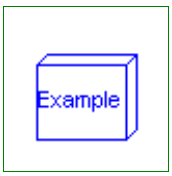
Parameters

Type	Name	Default	Description
AbsolutePressure	dp_nominal		Nominal pressure drop [Pa]
MassFlowRate	m_flow_nominal		Nominal mass flow rate at nominal pressure drop [kg/s]

Connectors

Type	Name	Description
FluidPort_a	port_a	
FluidPort_b	port_b	

Modelica.Media.Examples.Tests.Components.PartialTestModel



Basic test model to test a medium

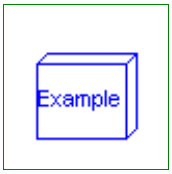
Information

Parameters

Type	Name	Default	Description
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

---

Modelica.Media.Examples.Tests.Components.PartialTestModel2



slightly larger test model to test a medium

Information

Parameters

Type	Name	Default	Description
AbsolutePressure	p_start	1.0e5	Initial value of pressure [Pa]
Temperature	T_start	300	Initial value of temperature [K]
SpecificEnthalpy	h_start	1	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.reference_X	Initial value of mass fractions

Modelica.Media.Examples.Tests.MediaTestModels

Test models to test all media

Information

Package Content

Name	Description
Air	Test models of library Modelica.Media.Air
IdealGases	Test models of library Modelica.Media.IdealGases
Incompressible	Test models of library Modelica.Media.Incompressible
Water	Test models of library Modelica.Media.Water
LinearFluid	Test models of library Modelica.Media.Incompressible

Modelica.Media.Examples.Tests.MediaTestModels.Air

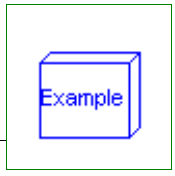
Test models of library Modelica.Media.Air

Information

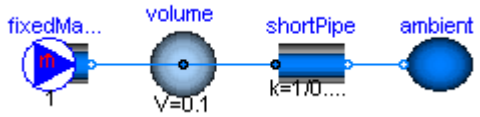
Package Content

Name	Description
SimpleAir	Test Modelica.Media.Air.SimpleAir
DryAirNasa	Test Modelica.Media.Air.DryAirNasa
MoistAir	Test Modelica.Media.Air.MoistAir

Modelica.Media.Examples.Tests.MediaTestModels.Air.SimpleAir



Test Modelica.Media.Air.SimpleAir



Information

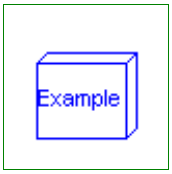
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

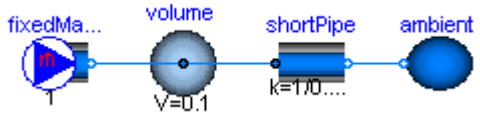
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Air.DryAirNasa



Test Modelica.Media.Air.DryAirNasa



Information

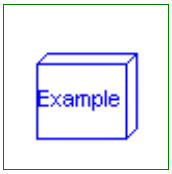
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

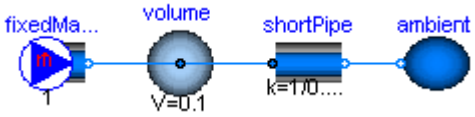
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Air.MoistAir



Test Modelica.Media.Air.MoistAir



Information

Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.IdealGases

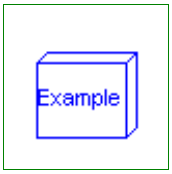
Test models of library Modelica.Media.IdealGases

Package Content

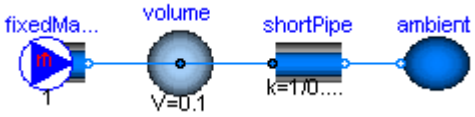
Name	Description
Air	Test single gas Modelica.Media.IdealGases.SingleGases.Air
Nitrogen	Test single gas Modelica.Media.IdealGases.SingleGases.N2
SimpleNaturalGas	Test mixture gas Modelica.Media.IdealGases.MixtureGases.SimpleNaturalGas
SimpleNaturalGasFixedComposition	Test mixture gas Modelica.Media.IdealGases.MixtureGases.SimpleNaturalGas

---

Modelica.Media.Examples.Tests.MediaTestModels.IdealGases.Air



Test single gas Modelica.Media.IdealGases.SingleGases.Air



Information

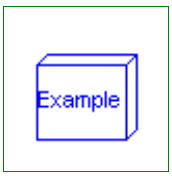
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

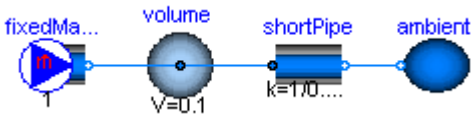
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.IdealGases.Nitrogen



Test single gas Modelica.Media.IdealGases.SingleGases.N2



Information

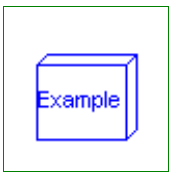
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

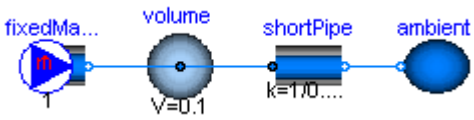
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.IdealGases.SimpleNaturalGas



Test mixture gas Modelica.Media.IdealGases.MixtureGases.SimpleNaturalGas



Information

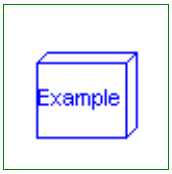
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

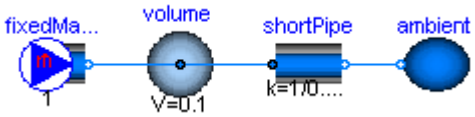
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.IdealGases.SimpleNaturalGasFixedComposition



Test mixture gas Modelica.Media.IdealGases.MixtureGases.SimpleNaturalGas



Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Incompressible

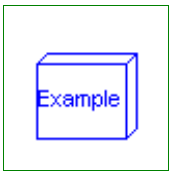
Test models of library Modelica.Media.Incompressible

Information

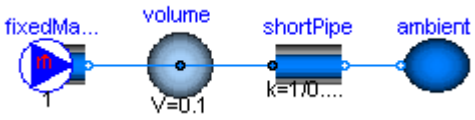
Package Content

Name	Description
Glycol47	Test Modelica.Media.Incompressible.Examples.Glycol47
Essotherm650	Test Modelica.Media.Incompressible.Examples.Essotherm65

Modelica.Media.Examples.Tests.MediaTestModels.Incompressible.Glycol47



Test Modelica.Media.Incompressible.Examples.Glycol47



Information

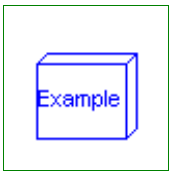
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

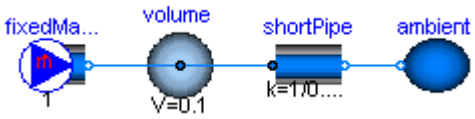
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Incompressible.Essotherm650



Test Modelica.Media.Incompressible.Examples.Essotherm65



Information

Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]

Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Water

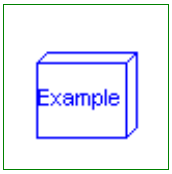
Test models of library Modelica.Media.Water

Information

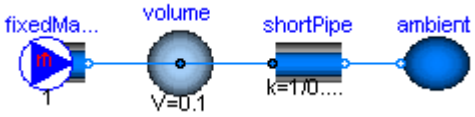
Package Content

Name	Description
ConstantPropertyLiquidWater	Test Modelica.Media.Water.ConstantPropertyLiquidWater
IdealSteam	Test Modelica.Media.Water.IdealSteam
WaterIF97OnePhase_ph	Test Modelica.Media.Water.WaterIF97OnePhase_ph
WaterIF97_pT	Test Modelica.Media.Water.WaterIF97_pT
WaterIF97_ph	Test Modelica.Media.Water.WaterIF97_ph

Modelica.Media.Examples.Tests.MediaTestModels.Water.ConstantPropertyLiquidWater



Test Modelica.Media.Water.ConstantPropertyLiquidWater



Information

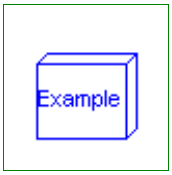
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

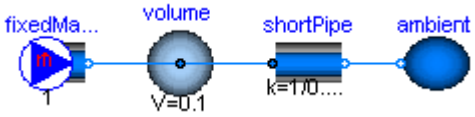
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Water.IdealSteam



Test Modelica.Media.Water.IdealSteam



Information

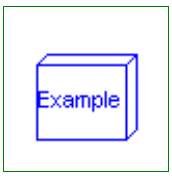
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

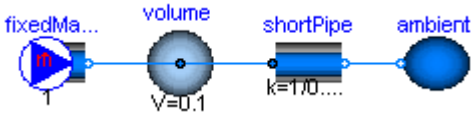
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Water.WaterIF97OnePhase_ph



Test Modelica.Media.Water.WaterIF97OnePhase_ph



Information

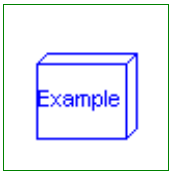
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

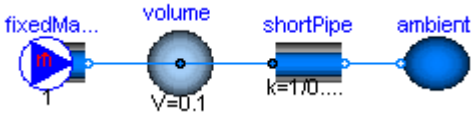
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Water.WaterIF97_pT



Test Modelica.Media.Water.WaterIF97_pT



Information

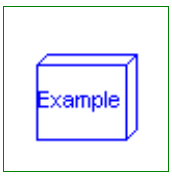
Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

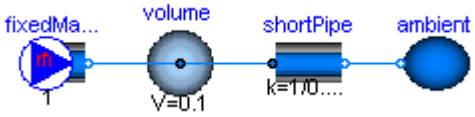
Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.Tests.MediaTestModels.Water.WaterIF97_ph



Test Modelica.Media.Water.WaterIF97_ph



Information

Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

Connectors

Type	Name	Description
replaceable package	Medium	Medium model

Modelica.Media.Examples.Tests.MediaTestModels.LinearFluid

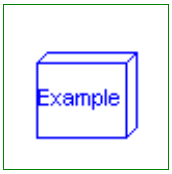
Test models of library Modelica.Media.Incompressible

Information

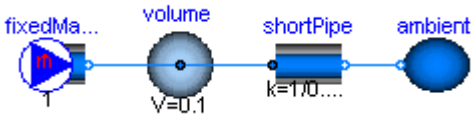
Package Content

Name	Description
LinearColdWater	Test Modelica.Media.Incompressible.Examples.Glycol47
LinearWater_pT	Test Modelica.Media.Incompressible.Examples.Essotherm65

Modelica.Media.Examples.Tests.MediaTestModels.LinearFluid.LinearColdWater



Test Modelica.Media.Incompressible.Examples.Glycol47



Information

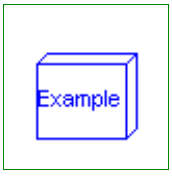
Parameters

Type	Name	Default	Description
replaceable package	Medium	PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

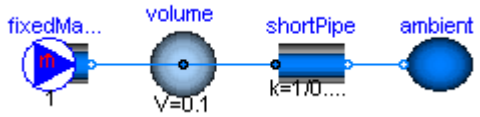
Connectors

Type	Name	Description
replaceable package	Medium	Medium model

Modelica.Media.Examples.Tests.MediaTestModels.LinearFluid.LinearWater_pT



Test Modelica.Media.Incompressible.Examples.Essotherm65



Information

Parameters

Type	Name	Default	Description
replaceable package Medium		PartialMedium	Medium model
AbsolutePressure	p_start	Medium.p_default	Initial value of pressure [Pa]
Temperature	T_start	Medium.T_default	Initial value of temperature [K]
SpecificEnthalpy	h_start	Medium.h_default	Initial value of specific enthalpy [J/kg]
Real	X_start[Medium.nX]	Medium.X_default	Initial value of mass fractions

Connectors

Type	Name	Description
replaceable package Medium		Medium model

Modelica.Media.Examples.SolveOneNonlinearEquation

Demonstrate how to solve one non-linear algebraic equation in one unknown

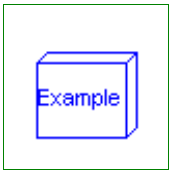
Information

This package demonstrates how to solve one non-linear algebraic equation in one unknown with function Modelica.Media.Common.OneNonLinearEquation.

Package Content

Name	Description
Inverse_sine	Solve y = A*sin(w*x) for x, given y
Inverse_sh_T	Solve h = h_T(T), s = s_T(T) for T, if h or s is given for ideal gas NASA
InverseIncompressible_sh_T	inverse computation for incmpressible media
Inverse_sh_TX	Solve h = h_TX(TX) for T, if h is given for ideal gas NASA

Modelica.Media.Examples.SolveOneNonlinearEquation.Inverse_sine



Solve y = A*sin(w*x) for x, given y

Information

This models solves the following non-linear equation

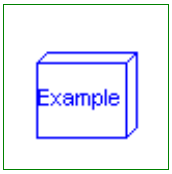
```
y = A*sin(w*x); -> determine x for given y
```

Translate model "Inverse_sine" and simulate for 0 sec. The result is printed to the output window.

Parameters

Type	Name	Default	Description
Real	y_zero	0.5	Desired value of A*sin(w*x)
Real	x_min	-1.7	Minimum value of x_zero
Real	x_max	1.7	Maximum value of x_zero
Real	A	1	
Real	w	1	
f_nonlinear_Data	data	Inverse_sine_definition.f_no...	

Modelica.Media.Examples.SolveOneNonlinearEquation.Inverse_sh_T



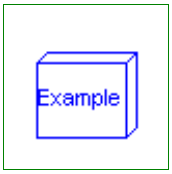
Solve h = h_T(T), s = s_T(T) for T, if h or s is given for ideal gas NASA

Information

Parameters

Type	Name	Default	Description
Temperature	T_min	300	[K]
Temperature	T_max	500	[K]
Pressure	p	1.0e5	[Pa]
SpecificEnthalpy	h_min	Medium.h_T(Medium.data, T_min)	[J/kg]
SpecificEnthalpy	h_max	Medium.h_T(Medium.data, T_max)	[J/kg]
SpecificEntropy	s_min	Medium.specificEntropy(Mediu...	[J/(kg.K)]
SpecificEntropy	s_max	Medium.specificEntropy(Mediu...	[J/(kg.K)]

Modelica.Media.Examples.SolveOneNonlinearEquation.InverseIncompressible_sh_T



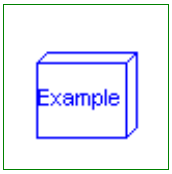
inverse computation for incmpressible media

Information

Parameters

Type	Name	Default	Description
Temperature	T_min	Medium.T_min	[K]
Temperature	T_max	Medium.T_max	[K]
Pressure	p	1.0e5	[Pa]
SpecificEnthalpy	h_min	Medium.h_T(Medium.T_min)	[J/kg]
SpecificEnthalpy	h_max	Medium.h_T(Medium.T_max)	[J/kg]
SpecificEntropy	s_min	Medium.specificEntropy(Mediu...	[J/(kg.K)]
SpecificEntropy	s_max	Medium.specificEntropy(Mediu...	[J/(kg.K)]

Modelica.Media.Examples.SolveOneNonlinearEquation.Inverse_sh_TX



Solve h = h_TX(TX) for T, if h is given for ideal gas NASA

Information

Parameters

Type	Name	Default	Description
Temperature	T_min	300	[K]
Temperature	T_max	500	[K]
Pressure	p	1.0e5	[Pa]
SpecificEntropy	s_min	Medium.specificEntropy(Mediu...	[J/(kg.K)]
SpecificEntropy	s_max	Medium.specificEntropy(Mediu...	[J/(kg.K)]

Modelica.Media.Interfaces

Interfaces for media models

Information

This package provides basic interfaces definitions of media models for different kind of media.

Package Content

Name	Description
TemplateMedium	Template for media models
PartialMedium	Partial medium properties (base package of all media packages)
PartialPureSubstance	base class for pure substances of one chemical substance
PartialLinearFluid	Generic pure liquid model with constant cp, compressibility and thermal expansion coefficients
PartialMixtureMedium	base class for pure substances of several chemical substances
PartialCondensingGases	Base class for mixtures of condensing and non-condensing gases
PartialTwoPhaseMedium	
PartialSimpleMedium	Medium model with linear dependency of u, h from temperature. All other quantities, especially density, are constant.
PartialSimpleIdealGasMedium	Medium model of Ideal gas with constant cp and cv. All other quantities, e.g. transport properties, are constant.

Modelica.Media.Interfaces.TemplateMedium

Template for media models

Information

This package is a **template** for **new medium** models. For a new medium model just make a copy of this package, remove the "partial" keyword from the package and provide the information that is requested in the

comments of the Modelica source.

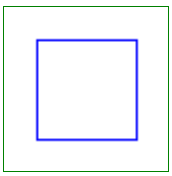
Package Content

Name	Description
BaseProperties	Base properties of medium
ThermodynamicState	a selction of variables that uniquely defines the thermodynamic state
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
specificEntropy	Return specific entropy
specificHeatCapacityCp	Return specific heat capacity at constant pressure
specificHeatCapacityCv	Return specific heat capacity at constant volume
isentropicExponent	Return isentropic exponent
velocityOfSound	Return velocity of sound
Inherited	
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
BasePropertiesRecord	Variables contained in every instance of BaseProperties
setState_pTX	Return thermodynamic state as function of p, T and composition X or Xi
setState_phX	Return thermodynamic state as function of p, h and

	composition X or Xi
setState_psX	Return thermodynamic state as function of p, s and composition X or Xi
setState_dTX	Return thermodynamic state as function of d, T and composition X or Xi
prandtlNumber	Return the Prandtl number
pressure	Return pressure
temperature	Return temperature
density	Return density
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
isentropicEnthalpy	Return isentropic enthalpy
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes

MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Modelica.Media.Interfaces.TemplateMedium.BaseProperties



Base properties of medium

Information

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent

			property variables of the medium

Modelica.Media.Interfaces.TemplateMedium.ThermodynamicState

a selction of variables that uniquely defines the thermodynamic state

Information

Modelica definition

```
redeclare replaceable record ThermodynamicState
  "a selction of variables that uniquely defines the thermodynamic state"
  AbsolutePressure p "Absolute pressure of medium";
  Temperature T "Temperature of medium";
end ThermodynamicState;
```

Modelica.Media.Interfaces.TemplateMedium.dynamicViscosity

Return dynamic viscosity

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Interfaces.TemplateMedium.thermalConductivity

Return thermal conductivity

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Interfaces.TemplateMedium.specificEntropy

Return specific entropy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.Interfaces.TemplateMedium.specificHeatCapacityCp

Return specific heat capacity at constant pressure

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Interfaces.TemplateMedium.specificHeatCapacityCv

Return specific heat capacity at constant volume

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Interfaces.TemplateMedium.isentropicExponent

Return isentropic exponent

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.Interfaces.TemplateMedium.velocityOfSound

Return velocity of sound

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.Interfaces.PartialMedium

Partial medium properties (base package of all media packages)

Information

PartialMedium is a package and contains all **declarations** for a medium. This means that constants, models, and functions are defined that every medium is supposed to support (some of them are optional). A medium package inherits from **PartialMedium** and provides the equations for the medium. The details of this package are described in Modelica.Media.UsersGuide.

Package Content

Name	Description
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused

singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
ThermodynamicState	Minimal variable set that is available as input argument to every medium function
BasePropertiesRecord	Variables contained in every instance of BaseProperties
BaseProperties	Base properties (p, d, T, h, u, R, MM and, if applicable, X) of a medium
setState_pTX	Return thermodynamic state as function of p, T and composition X or Xi
setState_phX	Return thermodynamic state as function of p, h and composition X or Xi
setState_psX	Return thermodynamic state as function of p, s and composition X or Xi
setState_dTX	Return thermodynamic state as function of d, T and composition X or Xi
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
prandtlNumber	Return the Prandtl number
pressure	Return pressure
temperature	Return temperature
density	Return density
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy

specificHeatCapacityCp	Return specific heat capacity at constant pressure
heatCapacity_cp	alias for deprecated name
specificHeatCapacityCv	Return specific heat capacity at constant volume
heatCapacity_cv	alias for deprecated name
isentropicExponent	Return isentropic exponent
isentropicEnthalpy	Return isentropic enthalpy
velocityOfSound	Return velocity of sound
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes

SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
  constant String mediumName = "unusablePartialMedium" "Name of the medium";

  constant String substanceNames[:]={mediumName}
  "Names of the mixture substances. Set substanceNames={mediumName} if only one
substance.";

  constant String extraPropertiesNames[:]=fill("", 0)
  "Names of the additional (extra) transported properties. Set
extraPropertiesNames=fill(\"\",0) if unused";

  constant Boolean singleState
  "= true, if u and d are not a function of pressure";

  constant Boolean reducedX=true
  "= true if medium contains the equation sum(X) = 1.0; set reducedX=true if
only one substance (see docu for details)";

  constant Boolean fixedX=false
  "= true if medium contains the equation X = reference_X";

  constant AbsolutePressure reference_p=101325
  "Reference pressure of Medium: default 1 atmosphere";

  constant Temperature reference_T=298.15
```

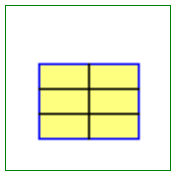
```
  "Reference temperature of Medium: default 25 deg Celsius";

  constant MassFraction reference_X[nX]= if nX == 0 then fill(0,nX) else
fill(1/nX, nX)
  "Default mass fractions of medium";

  constant AbsolutePressure p_default=101325
  "Default value for pressure of medium (for initialization)";

  constant Temperature T_default = Modelica.SIunits.Conversions.from_degC(20)
  "Default value for temperature of medium (for initialization)";

  constant SpecificEnthalpy h_default = specificEnthalpy_pTX(p_default, T_default,
X_default)
  "Default value for specific enthalpy of medium (for initialization)";

  constant MassFraction X_default[nX]=reference_X
  "Default value for mass fractions of medium (for initialization)";

  final constant Integer nS=size(substanceNames, 1) "Number of substances";

  constant Integer nX=if nS == 1 then 0 else nS
  "Number of mass fractions (= 0, if only one substance)";

  constant Integer nXi=if fixedX then 0 else if reducedX then nS - 1 else nX
  "Number of structurally independent mass fractions (see docu for details)";

  final constant Integer nC=size(extraPropertiesNames, 1)
  "Number of extra (outside of standard mass-balance) transported properties";

  type AbsolutePressure = SI.AbsolutePressure (
      min=0,
      max=1.e8,
      nominal=1.e5,
      start=1.e5) "Type for absolute pressure with medium specific attributes";

  type Density = SI.Density (
      min=0,
      max=1.e5,
      nominal=1,
      start=1) "Type for density with medium specific attributes";

  type DynamicViscosity = SI.DynamicViscosity (
      min=0,
      max=1.e8,
      nominal=1.e-3,
      start=1.e-3) "Type for dynamic viscosity with medium specific attributes";

  type EnthalpyFlowRate = SI.EnthalpyFlowRate (
      nominal=1000.0,
      min=-1.0e8,
      max=1.e8) "Type for enthalpy flow rate with medium specific attributes";

  type MassFlowRate = SI.MassFlowRate (
```

```
    quantity="MassFlowRate." + mediumName,
    min=-1.0e5,
    max=1.e5) "Type for mass flow rate with medium specific attributes";

  type MassFraction = Real (
    quantity="MassFraction",
    final unit="kg/kg",
    min=0,
    max=1,
    nominal=0.1) "Type for mass fraction with medium specific attributes";

  type MoleFraction = Real (
    quantity="MoleFraction",
    final unit="mol/mol",
    min=0,
    max=1,
    nominal=0.1) "Type for mole fraction with medium specific attributes";

  type MolarMass = SI.MolarMass (
    min=0.001,
    max=0.25,
    nominal=0.032) "Type for molar mass with medium specific attributes";

  type MolarVolume = SI.MolarVolume (
    min=1e-6,
    max=1.0e6,
    nominal=1.0) "Type for molar volume with medium specific attributes";

  type IsentropicExponent = SI.RatioOfSpecificHeatCapacities (
    min=1,
    max=500000,
    nominal=1.2,
    start=1.2) "Type for isentropic exponent with medium specific attributes";

  type SpecificEnergy = SI.SpecificEnergy (
    min=-1.0e8,
    max=1.e8,
    nominal=1.e6) "Type for specific energy with medium specific attributes";

  type SpecificInternalEnergy = SpecificEnergy
  "Type for specific internal energy with medium specific attributes";

  type SpecificEnthalpy = SI.SpecificEnthalpy (
    min=-1.0e8,
    max=1.e8,
    nominal=1.e6)
  "Type for specific enthalpy with medium specific attributes";

  type SpecificEntropy = SI.SpecificEntropy (
    min=-1.e6,
    max=1.e6,
    nominal=1.e3) "Type for specific entropy with medium specific attributes";

  type SpecificHeatCapacity = SI.SpecificHeatCapacity (
    min=0,
```

```
      max=1.e6,
      nominal=1.e3,
      start=1.e3)
  "Type for specific heat capacity with medium specific attributes";

  type SurfaceTension = SI.SurfaceTension
  "Type for surface tension with medium specific attributes";

  type Temperature = SI.Temperature (
      min=1,
      max=1.e4,
      nominal=300,
      start=300) "Type for temperature with medium specific attributes";

  type ThermalConductivity = SI.ThermalConductivity (
      min=0,
      max=500,
      nominal=1,
      start=1) "Type for thermal conductivity with medium specific attributes";

  type PrandtlNumber = SI.PrandtlNumber (
      min=1e-3,
      max=1e5,
      nominal=1.0) "Type for Prandtl number with medium specific attributes";

  type VelocityOfSound = SI.Velocity (
      min=0,
      max=1.e5,
      nominal=1000,
      start=1000) "Type for velocity of sound with medium specific attributes";

  type ExtraProperty = Real (min=0.0, start=1.0)
  "Type for unspecified, mass-specific property transported by flow";

  type CumulativeExtraProperty = Real (min=0.0, start=1.0)
  "Type for conserved integral of unspecified, mass specific property";

  type ExtraPropertyFlowRate = Real
  "Type for flow rate of unspecified, mass-specific property";

  type IsobaricExpansionCoefficient = Real (
      min=1e-8,
      max=1.0e8,
      unit="1/K")
  "Type for isobaric expansion coefficient with medium specific attributes";

  type DipoleMoment = Real (
      min=0.0,
      max=2.0,
      unit="debye",
      quantity="ElectricDipoleMoment")
  "Type for dipole moment with medium specific attributes";

  type DerDensityByPressure = SI.DerDensityByPressure
  "Type for partial derivative of density with resect to pressure with medium
```

```
specific attributes";

  type DerDensityByEnthalpy = SI.DerDensityByEnthalpy
  "Type for partial derivative of density with resect to enthalpy with medium
specific attributes";

  type DerEnthalpyByPressure = SI.DerEnthalpyByPressure
  "Type for partial derivative of enthalpy with resect to pressure with medium
specific attributes";

  type DerDensityByTemperature = SI.DerDensityByTemperature
  "Type for partial derivative of density with resect to temperature with medium
specific attributes";
```

Modelica.Media.Interfaces.PartialMedium.FluidConstants



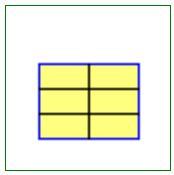
critical, triple, molecular and other standard data of fluid

Information

Modelica definition

```
replaceable record FluidConstants
  "critical, triple, molecular and other standard data of fluid"
  extends Modelica.Icons.Record;
  String iupacName "complete IUPAC name (or common name, if non-existent)";
  String casRegistryNumber
    "chemical abstracts sequencing number (if it exists)";
  String chemicalFormula
    "Chemical formula, (brutto, nomenclature according to Hill";
  String structureFormula "Chemical structure formula";
  MolarMass molarMass "molar mass";
end FluidConstants;
```

Modelica.Media.Interfaces.PartialMedium.ThermodynamicState



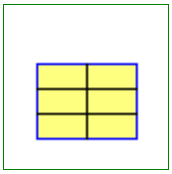
Minimal variable set that is available as input argument to every medium function

Information

Modelica definition

```
replaceable record ThermodynamicState
  "Minimal variable set that is available as input argument to every medium
function"
  extends Modelica.Icons.Record;
end ThermodynamicState;
```

Modelica.Media.Interfaces.PartialMedium.BasePropertiesRecord



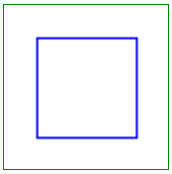
Variables contained in every instance of BaseProperties

Information

Modelica definition

```
replaceable record BasePropertiesRecord
  "Variables contained in every instance of BaseProperties"
  extends Modelica.Icons.Record;
  AbsolutePressure p "Absolute pressure of medium";
  Density d "Density of medium";
  Temperature T "Temperature of medium";
  MassFraction[nX] X(start=reference_X)
    "Mass fractions (= (component mass)/total mass  m_i/m)";
  MassFraction[nXi] Xi(start=reference_X[1:nXi])
    "Structurally independent mass fractions";
  SpecificEnthalpy h "Specific enthalpy of medium";
  SpecificInternalEnergy u "Specific internal energy of medium";
  SpecificHeatCapacity R "Gas constant (of mixture if applicable)";
  MolarMass MM "Molar mass (of mixture or single fluid)";
end BasePropertiesRecord;
```

Modelica.Media.Interfaces.PartialMedium.BaseProperties



Base properties (p, d, T, h, u, R, MM and, if applicable, X) of a medium

Information

Model **BaseProperties** is a model within package **PartialMedium** and contains the **declarations** of the minimum number of variables that every medium model is supposed to support. A specific medium inherits from model **BaseProperties** and provides the equations for the basic properties. Note, that in package PartialMedium the following constants are defined:

Type	Name	Description
String	mediumName	Unique name of the medium (used to check whether two media in a model are the same)
String	substanceNames	Names of the mixture substances that are treated as independent. If medium consists of a single substance, set substanceNames=fill("",0). If medium consists of n substances, provide either n-1 or n substance names, depending whether mass fractions PartialMedium.BaseProperties.X shall have dimension PartialMedium.nX = n-1 or PartialMedium.nX = n
Boolean	incompressible	= true, if density is constant; otherwise set it to false

In every medium **3+nX equations** have to be defined that provide relations between the following **5+nX variables**, declared in model BaseProperties, where nX is the number of independent mass fractions defined in package PartialMedium:

Variable	Unit	Description
T	K	temperature
p	Pa	absolute pressure
d	kg/m^3	density
h	J/kg	specific enthalpy

u	J/kg	specific internal energy
X[nX]	kg/kg	independent mass fractions m_i/m

In some components, such as "Ambient", explicit equations for medium variables are provided as "boundary conditions". For example, the "Ambient" component may define a temperature T_ambient.

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

---

Modelica.Media.Interfaces.PartialMedium.setState_pTX

Return thermodynamic state as function of p, T and composition X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialMedium.setState_phX

Return thermodynamic state as function of p, h and composition X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialMedium.setState_psX

Return thermodynamic state as function of p, s and composition X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialMedium.setState_dTX

Return thermodynamic state as function of d, T and composition X or Xi

Information

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialMedium.dynamicViscosity

Return dynamic viscosity

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Interfaces.PartialMedium.thermalConductivity

Return thermal conductivity

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Interfaces.PartialMedium.prandtlNumber

Return the Prandtl number

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
PrandtlNumber	Pr	Prandtl number [1]

Modelica.Media.Interfaces.PartialMedium.pressure

Return pressure

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Interfaces.PartialMedium.temperature

Return temperature

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialMedium.density

Return density

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialMedium.specificEnthalpy

Return specific enthalpy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialMedium.specificInternalEnergy

Return specific internal energy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	u	Specific internal energy [J/kg]

Modelica.Media.Interfaces.PartialMedium.specificEntropy

Return specific entropy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.Interfaces.PartialMedium.specificGibbsEnergy

Return specific Gibbs energy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	g	Specific Gibbs energy [J/kg]

Modelica.Media.Interfaces.PartialMedium.specificHelmholtzEnergy

Return specific Helmholtz energy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	f	Specific Helmholtz energy [J/kg]

Modelica.Media.Interfaces.PartialMedium.specificHeatCapacityCp

Return specific heat capacity at constant pressure

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Interfaces.PartialMedium.heatCapacity_cp

alias for deprecated name

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Interfaces.PartialMedium.specificHeatCapacityCv

Return specific heat capacity at constant volume

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Interfaces.PartialMedium.heatCapacity_cv

alias for deprecated name

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Interfaces.PartialMedium.isentropicExponent

Return isentropic exponent

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.Interfaces.PartialMedium.isentropicEnthalpy

Return isentropic enthalpy

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p_downstream		downstream pressure [Pa]
ThermodynamicState	refState		reference state for entropy

Outputs

Type	Name	Description
SpecificEnthalpy	h_is	Isentropic enthalpy [J/kg]

Modelica.Media.Interfaces.PartialMedium.velocityOfSound

Return velocity of sound

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.Interfaces.PartialMedium.isobaricExpansionCoefficient

Return overall the isobaric expansion coefficient beta

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsobaricExpansionCoefficient	beta	Isobaric expansion coefficient [1/K]

Modelica.Media.Interfaces.PartialMedium.beta

alias for isobaricExpansionCoefficient for user convenience

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsobaricExpansionCoefficient	beta	Isobaric expansion coefficient [1/K]

Modelica.Media.Interfaces.PartialMedium.isothermalCompressibility

Return overall the isothermal compressibility factor

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	Isothermal compressibility [1/Pa]

Modelica.Media.Interfaces.PartialMedium.kappa

alias of isothermalCompressibility for user convenience

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	Isothermal compressibility [1/Pa]

Modelica.Media.Interfaces.PartialMedium.density_derp_h

Return density derivative wrt pressure at const specific enthalpy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByPressure	ddph	Density derivative wrt pressure [s2/m2]

Modelica.Media.Interfaces.PartialMedium.density_derh_p

Return density derivative wrt specific enthalpy at constant pressure

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByEnthalpy	ddhp	Density derivative wrt specific enthalpy [kg.s2/m5]

Modelica.Media.Interfaces.PartialMedium.density_derp_T

Return density derivative wrt pressure at const temperature

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByPressure	ddpT	Density derivative wrt pressure [s2/m2]

Modelica.Media.Interfaces.PartialMedium.density_derT_p

Return density derivative wrt temperature at constant pressure

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByTemperature	ddTp	Density derivative wrt temperature [kg/(m3.K)]

Modelica.Media.Interfaces.PartialMedium.density_derX

Return density derivative wrt mass fraction

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	dddX[nX]	Derivative of density wrt mass fraction [kg/m3]

Modelica.Media.Interfaces.PartialMedium.molarMass

Return the molar mass of the medium

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
MolarMass	MM	Mixture molar mass [kg/mol]

Modelica.Media.Interfaces.PartialMedium.specificEnthalpy_pTX

Return specific enthalpy from p, T, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialMedium.density_pTX

Return density from p, T, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]		Mass fractions [kg/kg]

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialMedium.temperature_phX

Return temperature from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialMedium.density_phX

Return density from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialMedium.temperature_psX

Return temperature from p,s, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialMedium.density_psX

Return density from p, s, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialMedium.specificEnthalpy_psX

Return specific enthalpy from p, s, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialMedium.Choices

Types, constants to define menu choices

Package Content

Name	Description
Init	Type, constants and menu choices to define initialization, as temporary solution until enumerations are available
ReferenceEnthalpy	Type, constants and menu choices to define reference enthalpy, as temporary solution until enumerations are available
ReferenceEntropy	Type, constants and menu choices to define reference entropy, as temporary solution until enumerations are available
pd	Type, constants and menu choices to define whether p or d are known, as temporary solution until enumerations are available
Th	Type, constants and menu choices to define whether T or h are known, as temporary solution until enumerations are available
Explicit	Type, constants and menu choices to define the explicitly given state variable inputs

---

Modelica.Media.Interfaces.PartialMedium.Choices.Init

Type, constants and menu choices to define initialization, as temporary solution until enumerations are available

Package Content

Name	Description
NoInit=1	
InitialStates=2	
SteadyState=3	
SteadyMass=4	

Temp	Temporary type with choices for menus (until enumerations are available)

Types and constants

```
constant Integer NoInit=1;

constant Integer InitialStates=2;

constant Integer SteadyState=3;

constant Integer SteadyMass=4;

type Temp
"Temporary type with choices for menus (until enumerations are available)"

  extends Integer;
end Temp;
```

Modelica.Media.Interfaces.PartialMedium.Choices.ReferenceEnthalpy

Type, constants and menu choices to define reference enthalpy, as temporary solution until enumerations are available

Package Content

Name	Description
ZeroAt0K=1	
ZeroAt25C=2	
UserDefined=3	
Temp	Temporary type with choices for menus (until enumerations are available)

Types and constants

```
constant Integer ZeroAt0K=1;

constant Integer ZeroAt25C=2;

constant Integer UserDefined=3;

type Temp
"Temporary type with choices for menus (until enumerations are available)"

  extends Integer;

end Temp;
```

Modelica.Media.Interfaces.PartialMedium.Choices.ReferenceEntropy

Type, constants and menu choices to define reference entropy, as temporary solution until enumerations are available

Package Content

Name	Description
ZeroAt0K=1	
ZeroAt0C=2	
UserDefined=3	
Temp	Temporary type with choices for menus (until enumerations are available)

Types and constants

```
constant Integer ZeroAt0K=1;

constant Integer ZeroAt0C=2;

constant Integer UserDefined=3;

type Temp
"Temporary type with choices for menus (until enumerations are available)"

  extends Integer;

end Temp;
```

Modelica.Media.Interfaces.PartialMedium.Choices.pd

Type, constants and menu choices to define whether p or d are known, as temporary solution until enumerations are available

Package Content

Name	Description
default=1	
p_known=2	
d_known=3	
Temp	Temporary type with choices for menus (until enumerations are available)

Types and constants

```
constant Integer default=1;

constant Integer p_known=2;

constant Integer d_known=3;

type Temp
"Temporary type with choices for menus (until enumerations are available)"

  extends Integer;
end Temp;
```

Modelica.Media.Interfaces.PartialMedium.Choices.Th

Type, constants and menu choices to define whether T or h are known, as temporary solution until enumerations are available

Package Content

Name	Description
default=1	
T_known=2	
h_known=3	
Temp	Temporary type with choices for menus (until enumerations are available)

Types and constants

```
constant Integer default=1;

constant Integer T_known=2;

constant Integer h_known=3;

type Temp
"Temporary type with choices for menus (until enumerations are available)"

  extends Integer;
end Temp;
```

Modelica.Media.Interfaces.PartialMedium.Choices.Explicit

Type, constants and menu choices to define the explicitly given state variable inputs

Package Content

Name	Description
dT_explicit=0	explicit in density and temperature
ph_explicit=1	explicit in pressure and specific enthalpy
ps_explicit=2	explicit in pressure and specific entropy
pT_explicit=3	explicit in pressure and temperature
Temp	Temporary type with choices for menus (until enumerations are available)

Types and constants

```
constant Integer dT_explicit=0 "explicit in density and temperature";

constant Integer ph_explicit=1 "explicit in pressure and specific enthalpy";

constant Integer ps_explicit=2 "explicit in pressure and specific entropy";

constant Integer pT_explicit=3 "explicit in pressure and temperature";

type Temp
"Temporary type with choices for menus (until enumerations are available)"
```

```
    extends Integer(min=0,max=3);
  end Temp;
```

Modelica.Media.Interfaces.PartialPureSubstance

base class for pure substances of one chemical substance

Package Content

Name	Description
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
Inherited	
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances

nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
ThermodynamicState	Minimal variable set that is available as input argument to every medium function
BasePropertiesRecord	Variables contained in every instance of BaseProperties
BaseProperties	Base properties (p, d, T, h, u, R, MM and, if applicable, X) of a medium
setState_pTX	Return thermodynamic state as function of p, T and composition X or Xi
setState_phX	Return thermodynamic state as function of p, h and composition X or Xi
setState_psX	Return thermodynamic state as function of p, s and composition X or Xi
setState_dTX	Return thermodynamic state as function of d, T and composition X or Xi
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
prandtlNumber	Return the Prandtl number
pressure	Return pressure
temperature	Return temperature
density	Return density
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificHeatCapacityCp	Return specific heat capacity at constant pressure
heatCapacity_cp	alias for deprecated name
specificHeatCapacityCv	Return specific heat capacity at constant volume
heatCapacity_cv	alias for deprecated name
isentropicExponent	Return isentropic exponent
isentropicEnthalpy	Return isentropic enthalpy
velocityOfSound	Return velocity of sound
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience

density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes

DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Modelica.Media.Interfaces.PartialPureSubstance.setState_pT

Return thermodynamic state from p and T

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialPureSubstance.setState_ph

Return thermodynamic state from p and h

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialPureSubstance.setState_ps

Return thermodynamic state from p and s

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]

SpecificEntropy	s		Specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialPureSubstance.setState_dT

Return thermodynamic state from d and T

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialPureSubstance.density_ph

Return density from p and h

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialPureSubstance.temperature_ph

Return temperature from p and h

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialPureSubstance.pressure_dT

Return pressure from d and T

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Interfaces.PartialPureSubstance.specificEnthalpy_dT

Return specific enthalpy from d and T

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialPureSubstance.specificEnthalpy_ps

Return specific enthalpy from p and s

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialPureSubstance.temperature_ps

Return temperature from p and s

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialPureSubstance.density_ps

Return density from p and s

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialPureSubstance.specificEnthalpy_pT

Return specific enthalpy from p and T

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialPureSubstance.density_pT

Return density from p and T

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialLinearFluid

Generic pure liquid model with constant cp, compressibility and thermal expansion coefficients

Information

Linear Compressibility Fluid Model

This linear compressibility fluid model is based on the assumptions that:

- The specific heat capacity at constant pressure (cp) is constant
- The isobaric expansion coefficient (beta) is constant
- The isothermal compressibility (kappa) is constant
- Pressure and temperature are used as states

That means that the density is a linear function in temperature and in pressure. In order to define the complete model, a number of constant reference values are needed which are computed at the reference values of the states pressure p and temperature T. The model can be interprested as a linearization of a full non-linear fluid model (but it is not linear in all thermodynamic coordinates). Reference values are needed for

1. the density (reference_d),
2. the specific enthalpy (reference_h),
3. the specific entropy (reference_s).

Apart from that, a user needs to define the molar mass, MM_const. Note that it is possible to define a fluid by computing the reference values from a full non-linear fluid model by computing the package constants using the standard functions defined in a fluid package (see example in liquids package).

Efficiency considerations

One of the main reasons to use a simple, linear fluid model is to achieve high performance in simulations. There are a number of possible compromises and possibilities to improve performance. Some of them can be influenced by a flag. The following rules where used in this model:

- All forward evaluations (using the ThermodynamicState record as input) are exactly following the assumptions above.
- If the flag **constantJacobian** is set to true in the package, all functions that typically appear in thermodynamic jacobians (specificHeatCapacityCv, density_derp_h, density_derh_p, density_derp_T, density_derT_p) are evaluated at reference conditions (that means using the reference density) instead of the density of the current pressure and temperature. This makes it possible to evaluate the thermodynamic jacobian at compile time.
- For inverse functions using other inputs than the states (e.g pressure p and specific enthalpy h), the inversion is using the reference state whenever that is necessary to achieve a symbolic inversion.
- If **constantJacobian** is set to false, the above list of functions is computed exactly according to the above list of assumptions

Authors:

Francesco Casella
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5
I-20133 Milano, Italy
email: casella@elet.polimi.it

and
Hubertus Tummescheit
Modelon AB
Ideon Science Park
SE-22730 Lund, Sweden
email: Hubertus.Tummescheit@Modelon.se

Package Content

Name	Description
cp_const	Specific heat capacity at constant pressure
beta_const	Thermal expansion coefficient at constant pressure
kappa_const	Isothermal compressibility
MM_const	Molar mass
reference_d	Density in reference conditions
reference_h	Specific enthalpy in reference conditions
reference_s	Specific enthalpy in reference conditions
constantJacobian	if true, entries in thermodynamic Jacobian are constant, taken at reference conditions
ThermodynamicState	a selction of variables that uniquely defines the thermodynamic state
BaseProperties	Base properties of medium
setState_pTX	set the thermodynamic state record from p and T (X not needed)
setState_phX	set the thermodynamic state record from p and h (X not needed)
setState_dTX	set the thermodynamic state record from d and T (X not needed)
setState_psX	set the thermodynamic state record from p and s (X not needed)
pressure	Return the pressure from the thermodynamic state
temperature	Return the temperature from the thermodynamic state
density	Return the density from the thermodynamic state
specificEnthalpy	Return the specific enthalpy from the thermodynamic state
specificEntropy	Return the specific entropy from the thermodynamic state
specificInternalEnergy	Return the specific internal energy from the thermodynamic state
specificGibbsEnergy	Return specific Gibbs energy from the thermodynamic state
specificHelmholtzEnergy	Return specific Helmholtz energy from the thermodynamic state
velocityOfSound	Return velocity of sound from the thermodynamic state
isentropicExponent	Return isentropic exponent from the thermodynamic state
isentropicEnthalpy	Return isentropic enthalpy
specificHeatCapacityCp	Return specific heat capacity at constant volume
specificHeatCapacityCv	Return specific heat capacity at constant volume from the thermodynamic state

isothermalCompressibility	Return the iso-thermal compressibility kappa
isobaricExpansionCoefficient	Return the iso-baric expansion coefficient
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
molarMass	Return molar mass
T_ph	Return temperature from pressure and specific enthalpy
T_ps	Return temperature from pressure and specific entropy
Inherited	
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default,	Default value for specific enthalpy of medium (for

T_default, X_default)	initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
BasePropertiesRecord	Variables contained in every instance of BaseProperties
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derX	Return density derivative wrt mass fraction
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes

SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
    constant SpecificHeatCapacity cp_const
  "Specific heat capacity at constant pressure";

    constant IsobaricExpansionCoefficient beta_const
  "Thermal expansion coefficient at constant pressure";

    constant SI.IsothermalCompressibility kappa_const
  "Isothermal compressibility";

    constant MolarMass MM_const "Molar mass";

    constant Density reference_d "Density in reference conditions";

    constant SpecificEnthalpy reference_h
  "Specific enthalpy in reference conditions";

    constant SpecificEntropy reference_s
  "Specific enthalpy in reference conditions";

    constant Boolean constantJacobian
  "if true, entries in thermodynamic Jacobian are constant, taken at reference
conditions";
```

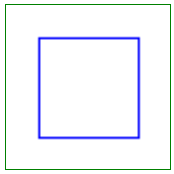
Modelica.Media.Interfaces.PartialLinearFluid.ThermodynamicState

a selction of variables that uniquely defines the thermodynamic state

Modelica definition

```
redeclare replaceable record ThermodynamicState
  "a selction of variables that uniquely defines the thermodynamic state"
  AbsolutePressure p "Absolute pressure of medium";
  Temperature T "Temperature of medium";
end ThermodynamicState;
```

Modelica.Media.Interfaces.PartialLinearFluid.BaseProperties



Base properties of medium

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

---

Modelica.Media.Interfaces.PartialLinearFluid.setState_pTX

set the thermodynamic state record from p and T (X not needed)

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialLinearFluid.setState_phX

set the thermodynamic state record from p and h (X not needed)

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]

SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialLinearFluid.setState_dTX

set the thermodynamic state record from d and T (X not needed)

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialLinearFluid.setState_psX

set the thermodynamic state record from p and s (X not needed)

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialLinearFluid.pressure

Return the pressure from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Interfaces.PartialLinearFluid.temperature

Return the temperature from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialLinearFluid.density

Return the density from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialLinearFluid.specificEnthalpy

Return the specific enthalpy from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialLinearFluid.specificEntropy

Return the specific entropy from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.Interfaces.PartialLinearFluid.specificInternalEnergy

Return the specific internal energy from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	u	Specific internal energy [J/kg]

Modelica.Media.Interfaces.PartialLinearFluid.specificGibbsEnergy

Return specific Gibbs energy from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	g	Specific Gibbs energy [J/kg]

Modelica.Media.Interfaces.PartialLinearFluid.specificHelmholtzEnergy

Return specific Helmholtz energy from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	f	Specific Helmholtz energy [J/kg]

Modelica.Media.Interfaces.PartialLinearFluid.velocityOfSound

Return velocity of sound from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.Interfaces.PartialLinearFluid.isentropicExponent

Return isentropic exponent from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.Interfaces.PartialLinearFluid.isentropicEnthalpy

Return isentropic enthalpy

Information

A minor approximation is used: the reference density is used instead of the real one, which would require a numeric solution.

Inputs

Type	Name	Default	Description
AbsolutePressure	p_downstream		downstream pressure [Pa]
ThermodynamicState	refState		reference state for entropy

Outputs

Type	Name	Description
SpecificEnthalpy	h_is	Isentropic enthalpy [J/kg]

Modelica.Media.Interfaces.PartialLinearFluid.specificHeatCapacityCp

Return specific heat capacity at constant volume

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Interfaces.PartialLinearFluid.specificHeatCapacityCv

Return specific heat capacity at constant volume from the thermodynamic state

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Interfaces.PartialLinearFluid.isothermalCompressibility

Return the iso-thermal compressibility kappa

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	Isothermal compressibility [1/Pa]

Modelica.Media.Interfaces.PartialLinearFluid.isobaricExpansionCoefficient

Return the iso-baric expansion coefficient

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsobaricExpansionCoefficient	beta	Isobaric expansion coefficient [1/K]

Modelica.Media.Interfaces.PartialLinearFluid.density_derp_h

Return density derivative wrt pressure at const specific enthalpy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByPressure	ddph	Density derivative wrt pressure [s2/m2]

Modelica.Media.Interfaces.PartialLinearFluid.density_derh_p

Return density derivative wrt specific enthalpy at constant pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByEnthalpy	ddhp	Density derivative wrt specific enthalpy [kg.s2/m5]

Modelica.Media.Interfaces.PartialLinearFluid.density_derp_T

Return density derivative wrt pressure at const temperature

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByPressure	ddpT	Density derivative wrt pressure [s2/m2]

Modelica.Media.Interfaces.PartialLinearFluid.density_derT_p

Return density derivative wrt temperature at constant pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByTemperature	ddTp	Density derivative wrt temperature [kg/(m3.K)]

Modelica.Media.Interfaces.PartialLinearFluid.molarMass

Return molar mass

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
MolarMass	MM	Mixture molar mass [kg/mol]

Modelica.Media.Interfaces.PartialLinearFluid.T_ph

Return temperature from pressure and specific enthalpy

Inputs

Type	Name	Default	Description
SpecificEnthalpy	h		Specific enthalpy [J/kg]
AbsolutePressure	p		pressure [Pa]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialLinearFluid.T_ps

Return temperature from pressure and specific entropy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialMixtureMedium

base class for pure substances of several chemical substances

Package Content

Name	Description
ThermodynamicState	thermodynamic state variables
FluidConstants	extended fluid constants
fluidConstants	constant data for the fluid
gasConstant	Return the gas constant of the mixture (also for liquids)
moleToMassFractions	Return mass fractions X from mole fractions
massToMoleFractions	Return mole fractions from mass fractions X
Inherited	
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
BaseProperties	Base properties (p, d, T, h, u, R, MM and, if applicable, X) of a medium

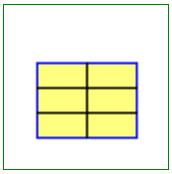
setState_pTX	Return thermodynamic state as function of p, T and composition X or Xi
setState_phX	Return thermodynamic state as function of p, h and composition X or Xi
setState_psX	Return thermodynamic state as function of p, s and composition X or Xi
setState_dTX	Return thermodynamic state as function of d, T and composition X or Xi
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
prandtlNumber	Return the Prandtl number
pressure	Return pressure
temperature	Return temperature
density	Return density
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificHeatCapacityCp	Return specific heat capacity at constant pressure
heatCapacity_cp	alias for deprecated name
specificHeatCapacityCv	Return specific heat capacity at constant volume
heatCapacity_cv	alias for deprecated name
isentropicExponent	Return isentropic exponent
isentropicEnthalpy	Return isentropic enthalpy
velocityOfSound	Return velocity of sound
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi

temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
  constant FluidConstants[nS] fluidConstants "constant data for the fluid";
```

Modelica.Media.Interfaces.PartialMixtureMedium.ThermodynamicState

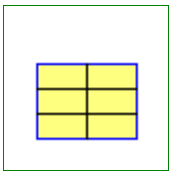


thermodynamic state variables

Modelica definition

```
redeclare replaceable record extends ThermodynamicState
  "thermodynamic state variables"
  AbsolutePressure p "Absolute pressure of medium";
  Temperature T "Temperature of medium";
  MassFraction X[nX] "Mass fractions (= (component mass)/total mass  m_i/m)";
end ThermodynamicState;
```

Modelica.Media.Interfaces.PartialMixtureMedium.FluidConstants



extended fluid constants

Modelica definition

```
redeclare replaceable record extends FluidConstants
  "extended fluid constants"
  Temperature criticalTemperature "critical temperature";
  AbsolutePressure criticalPressure "critical pressure";
  MolarVolume criticalMolarVolume "critical molar Volume";
  Real acentricFactor "Pitzer acentric factor";
  Temperature triplePointTemperature "triple point temperature";
  AbsolutePressure triplePointPressure "triple point pressure";
  Temperature meltingPoint "melting point at 101325 Pa";
  Temperature normalBoilingPoint "normal boiling point (at 101325 Pa)";
  DipoleMoment dipoleMoment
    "dipole moment of molecule in Debye (1 debye = 3.33564e10-30 C.m)";
  Boolean hasIdealGasHeatCapacity=false
    "true if ideal gas heat capacity is available";
  Boolean hasCriticalData=false "true if critical data are known";
  Boolean hasDipoleMoment=false "true if a dipole moment known";
  Boolean hasFundamentalEquation=false "true if a fundamental equation";
  Boolean hasLiquidHeatCapacity=false
    "true if liquid heat capacity is available";
  Boolean hasSolidHeatCapacity=false "true if solid heat capacity is available";
  Boolean hasAccurateViscosityData=false
    "true if accurate data for a viscosity function is available";
  Boolean hasAccurateConductivityData=false
    "true if accurate data for thermal conductivity is available";
  Boolean hasVapourPressureCurve=false
    "true if vapour pressure data, e.g. Antoine coefficents are known";
  Boolean hasAcentricFactor=false "true if Pitzer accentric factor is known";
  SpecificEnthalpy HCRIT0=0.0
    "Critical specific enthalpy of the fundamental equation";
  SpecificEntropy SCRIT0=0.0
```

```
    "Critical specific entropy of the fundamental equation";
  SpecificEnthalpy deltah=0.0
    "Difference between specific enthalpy model (h_m) and f.eq. (h_f) (h_m -
h_f)";
  SpecificEntropy deltas=0.0
    "Difference between specific enthalpy model (s_m) and f.eq. (s_f) (s_m -
s_f)";
end FluidConstants;
```

Modelica.Media.Interfaces.PartialMixtureMedium.gasConstant

Return the gas constant of the mixture (also for liquids)

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state

Outputs

Type	Name	Description
SpecificHeatCapacity	R	mixture gas constant [J/(kg.K)]

Modelica.Media.Interfaces.PartialMixtureMedium.moleToMassFractions

Return mass fractions X from mole fractions

Inputs

Type	Name	Default	Description
MoleFraction	moleFractions[:]		Mole fractions of mixture [1]
MolarMass	MMX[:]		molar masses of components [kg/mol]

Outputs

Type	Name	Description
MassFraction	X[size(moleFractions, 1)]	Mass fractions of gas mixture [1]

Modelica.Media.Interfaces.PartialMixtureMedium.massToMoleFractions

Return mole fractions from mass fractions X

Inputs

Type	Name	Default	Description
MassFraction	X[:]		Mass fractions of mixture [1]
MolarMass	MMX[:]		molar masses of components [kg/mol]

Outputs

Type	Name	Description
MoleFraction	moleFractions[size(X, 1)]	Mole fractions of gas mixture [1]

Modelica.Media.Interfaces.PartialCondensingGases

Base class for mixtures of condensing and non-condensing gases

Package Content

Name	Description
saturationPressure	Return saturation pressure of condensing fluid
enthalpyOfVaporization	Return vaporization enthalpy of condensing fluid
enthalpyOfLiquid	Return liquid enthalpy of condensing fluid
enthalpyOfGas	Return enthalpy of non-condensing gas mixture
enthalpyOfCondensingGas	Return enthalpy of condensing gas (most often steam)
Inherited	
ThermodynamicState	thermodynamic state variables
FluidConstants	extended fluid constants
fluidConstants	constant data for the fluid
gasConstant	Return the gas constant of the mixture (also for liquids)
moleToMassFractions	Return mass fractions X from mole fractions
massToMoleFractions	Return mole fractions from mass fractions X
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)

nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
BaseProperties	Base properties (p, d, T, h, u, R, MM and, if applicable, X) of a medium
setState_pTX	Return thermodynamic state as function of p, T and composition X or Xi
setState_phX	Return thermodynamic state as function of p, h and composition X or Xi
setState_psX	Return thermodynamic state as function of p, s and composition X or Xi
setState_dTX	Return thermodynamic state as function of d, T and composition X or Xi
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
prandtlNumber	Return the Prandtl number
pressure	Return pressure
temperature	Return temperature
density	Return density
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificHeatCapacityCp	Return specific heat capacity at constant pressure
heatCapacity_cp	alias for deprecated name
specificHeatCapacityCv	Return specific heat capacity at constant volume
heatCapacity_cv	alias for deprecated name
isentropicExponent	Return isentropic exponent
isentropicEnthalpy	Return isentropic enthalpy
velocityOfSound	Return velocity of sound
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature

density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes

DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Modelica.Media.Interfaces.PartialCondensingGases.saturationPressure

Return saturation pressure of condensing fluid

Inputs

Type	Name	Default	Description
Temperature	Tsat		saturation temperature [K]

Outputs

Type	Name	Description
AbsolutePressure	psat	saturation pressure [Pa]

Modelica.Media.Interfaces.PartialCondensingGases.enthalpyOfVaporization

Return vaporization enthalpy of condensing fluid

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	r0	vaporization enthalpy [J/kg]

Modelica.Media.Interfaces.PartialCondensingGases.enthalpyOfLiquid

Return liquid enthalpy of condensing fluid

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	liquid enthalpy [J/kg]

Modelica.Media.Interfaces.PartialCondensingGases.enthalpyOfGas

Return enthalpy of non-condensing gas mixture

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]
MassFraction	X[:]		vector of mass fractions [kg/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	liquid enthalpy [J/kg]

Modelica.Media.Interfaces.PartialCondensingGases.enthalpyOfCondensingGas

Return enthalpy of condensing gas (most often steam)

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	liquid enthalpy [J/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium

Package Content

Name	Description
smoothModel	true if the (derived) model should not generate state events
onePhase	true if the (derived) model should never be called with two-phase inputs
FluidLimits	validity limits for fluid model
FluidConstants	extended fluid constants
fluidConstants	constant data for the fluid
ThermodynamicState	Thermodynamic state of two phase medium
SaturationProperties	Saturation properties of two phase medium
FixedPhase	phase of the fluid: 1 for 1-phase, 2 for two-phase, 0 for not known, e.g. interactive use
BaseProperties	Base properties (p, d, T, h, u, R, MM, sat) of two phase medium
setDewState	Return the thermodynamic state on the dew line
setBubbleState	Return the thermodynamic state on the bubble line

setState_dTX	Return thermodynamic state as function of d, T and composition X or Xi
setState_phX	Return thermodynamic state as function of p, h and composition X or Xi
setState_psX	Return thermodynamic state as function of p, s and composition X or Xi
setState_pTX	Return thermodynamic state as function of p, T and composition X or Xi
setSat_T	Return saturation property record from temperature
setSat_p	Return saturation property record from pressure
bubbleEnthalpy	Return bubble point specific enthalpy
dewEnthalpy	Return dew point specific enthalpy
bubbleEntropy	Return bubble point specific entropy
dewEntropy	Return dew point specific entropy
bubbleDensity	Return bubble point density
dewDensity	Return dew point density
saturationPressure	Return saturation pressure
saturationTemperature	Return saturation temperature
saturationPressure_sat	Return saturation temperature
saturationTemperature_sat	Return saturation temperature
saturationTemperature_derp	Return derivative of saturation temperature w.r.t. pressure
saturationTemperature_derp_sat	Return derivative of saturation temperature w.r.t. pressure
surfaceTension	Return surface tension sigma in the two phase region
molarMass	Return the molar mass of the medium
dBubbleDensity_dPressure	Return bubble point density derivative
dDewDensity_dPressure	Return dew point density derivative
dBubbleEnthalpy_dPressure	Return bubble point specific enthalpy derivative
dDewEnthalpy_dPressure	Return dew point specific enthalpy derivative
specificEnthalpy_pTX	Return specific enthalpy from pressure, temperature and mass fraction
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p, s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
setState_px	Return thermodynamic state from pressure and vapour

	quality
setState_Tx	Return thermodynamic state from temperature and vapour quality
vapourQuality	Return vapour quality
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
Inherited	
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
prandtlNumber	Return the Prandtl number
pressure	Return pressure

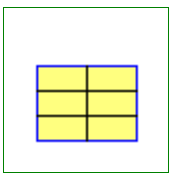
temperature	Return temperature
density	Return density
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificHeatCapacityCp	Return specific heat capacity at constant pressure
heatCapacity_cp	alias for deprecated name
specificHeatCapacityCv	Return specific heat capacity at constant volume
heatCapacity_cv	alias for deprecated name
isentropicExponent	Return isentropic exponent
isentropicEnthalpy	Return isentropic enthalpy
velocityOfSound	Return velocity of sound
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
density_pTX	Return density from p, T, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes

SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
  constant Boolean smoothModel
  "true if the (derived) model should not generate state events";

  constant Boolean onePhase
  "true if the (derived) model should never be called with two-phase inputs";

constant FluidConstants[nS] fluidConstants "constant data for the fluid";

  type FixedPhase = Integer(min=0,max=2)
  "phase of the fluid: 1 for 1-phase, 2 for two-phase, 0 for not known, e.g.
interactive use";
```

Modelica.Media.Interfaces.PartialTwoPhaseMedium.FluidLimits



validity limits for fluid model

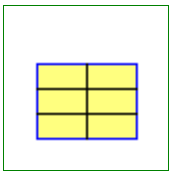
Information

The minimum pressure mostly applies to the liquid state only. The minimum density is also arbitrary, but is reasonable for techical applications to limit iterations in non-linear systems. The limits in enthalpy and entropy are used as safeguards in inverse iterations.

Modelica definition

```
record FluidLimits "validity limits for fluid model"
  extends Modelica.Icons.Record;
  Temperature TMIN "minimum temperature";
  Temperature TMAX "maximum temperature";
  Density DMIN "minimum density";
  Density DMAX "maximum density";
  AbsolutePressure PMIN "minimum pressure";
  AbsolutePressure PMAX "maximum pressure";
  SpecificEnthalpy HMIN "minimum enthalpy";
  SpecificEnthalpy HMAX "maximum enthalpy";
  SpecificEntropy SMIN "minimum entropy";
  SpecificEntropy SMAX "maximum entropy";
end FluidLimits;
```

Modelica.Media.Interfaces.PartialTwoPhaseMedium.FluidConstants



extended fluid constants

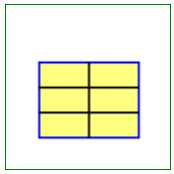
Information

Modelica definition

```
redeclare replaceable record extends FluidConstants
  "extended fluid constants"
  Temperature criticalTemperature "critical temperature";
  AbsolutePressure criticalPressure "critical pressure";
  MolarVolume criticalMolarVolume "critical molar Volume";
  Real acentricFactor "Pitzer acentric factor";
  Temperature triplePointTemperature "triple point temperature";
  AbsolutePressure triplePointPressure "triple point pressure";
  Temperature meltingPoint "melting point at 101325 Pa";
  Temperature normalBoilingPoint "normal boiling point (at 101325 Pa)";
  DipoleMoment dipoleMoment
    "dipole moment of molecule in Debye (1 debye = 3.33564e10-30 C.m)";
  Boolean hasIdealGasHeatCapacity=false
    "true if ideal gas heat capacity is available";
  Boolean hasCriticalData=false "true if critical data are known";
  Boolean hasDipoleMoment=false "true if a dipole moment known";
  Boolean hasFundamentalEquation=false "true if a fundamental equation";
  Boolean hasLiquidHeatCapacity=false
    "true if liquid heat capacity is available";
  Boolean hasSolidHeatCapacity=false "true if solid heat capacity is available";
  Boolean hasAccurateViscosityData=false
    "true if accurate data for a viscosity function is available";
  Boolean hasAccurateConductivityData=false
    "true if accurate data for thermal conductivity is available";
  Boolean hasVapourPressureCurve=false
    "true if vapour pressure data, e.g. Antoine coefficents are known";
  Boolean hasAcentricFactor=false "true if Pitzer accentric factor is known";
  SpecificEnthalpy HCRIT0=0.0
    "Critical specific enthalpy of the fundamental equation";
  SpecificEntropy SCRIT0=0.0
```

```
    "Critical specific entropy of the fundamental equation";
  SpecificEnthalpy deltah=0.0
    "Difference between specific enthalpy model (h_m) and f.eq. (h_f) (h_m -
h_f)";
  SpecificEntropy deltas=0.0
    "Difference between specific enthalpy model (s_m) and f.eq. (s_f) (s_m -
s_f)";
end FluidConstants;
```

Modelica.Media.Interfaces.PartialTwoPhaseMedium.ThermodynamicState



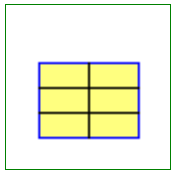
Thermodynamic state of two phase medium

Information

Modelica definition

```
redeclare replaceable record extends ThermodynamicState
  "Thermodynamic state of two phase medium"
    FixedPhase phase(min=0, max=2)
    "phase of the fluid: 1 for 1-phase, 2 for two-phase, 0 for not known, e.g.
interactive use";
end ThermodynamicState;
```

Modelica.Media.Interfaces.PartialTwoPhaseMedium.SaturationProperties



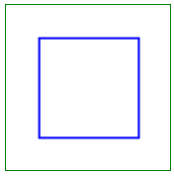
Saturation properties of two phase medium

Information

Modelica definition

```
replaceable record SaturationProperties
  "Saturation properties of two phase medium"
  extends Modelica.Icons.Record;
  AbsolutePressure psat "saturation pressure";
  Temperature Tsat "saturation temperature";
end SaturationProperties;
```

Modelica.Media.Interfaces.PartialTwoPhaseMedium.BaseProperties



Base properties (p, d, T, h, u, R, MM, sat) of two phase medium

Information

Parameters

Type	Name	Default	Description

Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setDewState

Return the thermodynamic state on the dew line

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation point
FixedPhase	phase	1	phase: default is one phase

Outputs

Type	Name	Description
ThermodynamicState	state	complete thermodynamic state info

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setBubbleState

Return the thermodynamic state on the bubble line

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation point
FixedPhase	phase	1	phase: default is one phase

Outputs

Type	Name	Description
ThermodynamicState	state	complete thermodynamic state info

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_dTX

Return thermodynamic state as function of d, T and composition X or Xi

Information

Inputs

Type	Name	Default	Description

FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_phX

Return thermodynamic state as function of p, h and composition X or Xi

Information

Inputs

Type	Name	Default	Description
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_psX

Return thermodynamic state as function of p, s and composition X or Xi

Information

Inputs

Type	Name	Default	Description
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_pTX

Return thermodynamic state as function of p, T and composition X or Xi

Information

Inputs

Type	Name	Default	Description
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setSat_T

Return saturation property record from temperature

Information

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
SaturationProperties	sat	saturation property record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setSat_p

Return saturation property record from pressure

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]

Outputs

Type	Name	Description
SaturationProperties	sat	saturation property record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.bubbleEnthalpy

Return bubble point specific enthalpy

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEnthalpy	hl	boiling curve specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.dewEnthalpy

Return dew point specific enthalpy

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEnthalpy	hv	dew curve specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.bubbleEntropy

Return bubble point specific entropy

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEntropy	sl	boiling curve specific entropy [J/(kg.K)]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.dewEntropy

Return dew point specific entropy

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEntropy	sv	dew curve specific entropy [J/(kg.K)]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.bubbleDensity

Return bubble point density

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
Density	dl	boiling curve density [kg/m3]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.dewDensity

Return dew point density

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
Density	dv	dew curve density [kg/m3]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationPressure

Return saturation pressure

Information

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
AbsolutePressure	p	saturation pressure [Pa]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationTemperature

Return saturation temperature

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]

Outputs

Type	Name	Description
Temperature	T	saturation temperature [K]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationPressure_sat

Return saturation temperature

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
AbsolutePressure	p	saturation pressure [Pa]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationTemperature_sat

Return saturation temperature

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
Temperature	T	saturation temperature [K]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationTemperature_derp

Return derivative of saturation temperature w.r.t. pressure

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]

Outputs

Type	Name	Description
Real	dTp	derivative of saturation temperature w.r.t. pressure

---

Modelica.Media.Interfaces.PartialTwoPhaseMedium.saturationTemperature_derp_sat

Return derivative of saturation temperature w.r.t. pressure

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
Real	dTp	derivative of saturation temperature w.r.t. pressure

---

Modelica.Media.Interfaces.PartialTwoPhaseMedium.surfaceTension

Return surface tension sigma in the two phase region

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SurfaceTension	sigma	Surface tension sigma in the two phase region [N/m]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.molarMass

Return the molar mass of the medium

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
MolarMass	MM	Mixture molar mass [kg/mol]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.dBubbleDensity_dPressure

Return bubble point density derivative

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerDensityByPressure	ddldp	boiling curve density derivative [s2/m2]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.dDewDensity_dPressure

Return dew point density derivative

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerDensityByPressure	ddvdp	saturated steam density derivative [s2/m2]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.dBubbleEnthalpy_dPressure

Return bubble point specific enthalpy derivative

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerEnthalpyByPressure	dhldp	boiling curve specific enthalpy derivative [J.m.s2/kg2]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.dDewEnthalpy_dPressure

Return dew point specific enthalpy derivative

Information

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerEnthalpyByPressure	dhvdp	saturated steam specific enthalpy derivative [J.m.s2/kg2]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_pTX

Return specific enthalpy from pressure, temperature and mass fraction

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[nX]		Mass fractions [kg/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy at p, T, X [J/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.temperature_phX

Return temperature from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[nX]		Mass fractions [kg/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_phX

Return density from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[nX]		Mass fractions [kg/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.temperature_psX

Return temperature from p, s, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[nX]		Mass fractions [kg/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_psX

Return density from p, s, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[nX]		Mass fractions [kg/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_psX

Return specific enthalpy from p, s, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[nX]		Mass fractions [kg/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_pT

Return thermodynamic state from p and T

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_ph

Return thermodynamic state from p and h

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_ps

Return thermodynamic state from p and s

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_dT

Return thermodynamic state from d and T

Information

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_px

Return thermodynamic state from pressure and vapour quality

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
MassFraction	x		Vapour quality [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	Thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.setState_Tx

Return thermodynamic state from temperature and vapour quality

Information

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]
MassFraction	x		Vapour quality [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialTwoPhaseMedium.vapourQuality

Return vapour quality

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		Thermodynamic state record

Outputs

Type	Name	Description
MassFraction	x	Vapour quality [kg/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_ph

Return density from p and h

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]

FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.temperature_ph

Return temperature from p and h

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.pressure_dT

Return pressure from d and T

Information

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_dT

Return specific enthalpy from d and T

Information

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_ps

Return specific enthalpy from p and s

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.temperature_ps

Return temperature from p and s

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_ps

Return density from p and s

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.specificEnthalpy_pT

Return specific enthalpy from p and T

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialTwoPhaseMedium.density_pT

Return density from p and T

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialSimpleMedium

Medium model with linear dependency of u, h from temperature. All other quantities, especially density, are constant.

Package Content

Name	Description
cp_const	Constant specific heat capacity at constant pressure
cv_const	Constant specific heat capacity at constant volume
d_const	Constant density
eta_const	Constant dynamic viscosity
lambda_const	Constant thermal conductivity
a_const	Constant velocity of sound
T_min	Minimum temperature valid for medium model
T_max	Maximum temperature valid for medium model
T0=reference_T	Zero enthalpy temperature
MM_const	Molar mass
fluidConstants	fluid constants
ThermodynamicState	Thermodynamic state
BaseProperties	Base properties
setState_pTX	Return thermodynamic state from p, T, and X or Xi
setState_phX	Return thermodynamic state from p, h, and X or Xi
setState_psX	Return thermodynamic state from p, s, and X or Xi
setState_dTX	Return thermodynamic state from d, T, and X or Xi
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
specificHeatCapacityCp	Return specific heat capacity at constant pressure
specificHeatCapacityCv	Return specific heat capacity at constant volume
isentropicExponent	Return isentropic exponent
velocityOfSound	Return velocity of sound
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
Inherited	
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s

setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
pressure	Return pressure
temperature	Return temperature
density	Return density
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy

specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
isentropicEnthalpy	Return isentropic enthalpy
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
density_pTX	Return density from p, T, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes

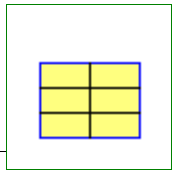
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
constant SpecificHeatCapacity cp_const
"Constant specific heat capacity at constant pressure";

constant SpecificHeatCapacity cv_const
"Constant specific heat capacity at constant volume";

constant Density d_const "Constant density";

constant DynamicViscosity eta_const "Constant dynamic viscosity";

constant ThermalConductivity lambda_const "Constant thermal conductivity";

constant VelocityOfSound a_const "Constant velocity of sound";

constant Temperature T_min "Minimum temperature valid for medium model";

constant Temperature T_max "Maximum temperature valid for medium model";

constant Temperature T0=reference_T "Zero enthalpy temperature";

constant MolarMass MM_const "Molar mass";

constant FluidConstants[nS] fluidConstants "fluid constants";
```

Modelica.Media.Interfaces.PartialSimpleMedium.ThermodynamicState

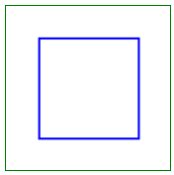
Thermodynamic state



Modelica definition

```
redeclare replaceable record extends ThermodynamicState
  "Thermodynamic state"
  AbsolutePressure p "Absolute pressure of medium";
  Temperature T "Temperature of medium";
end ThermodynamicState;
```

Modelica.Media.Interfaces.PartialSimpleMedium.BaseProperties



Base properties

Information

This is the most simple incompressible medium model, where specific enthalpy h and specific internal energy u are only a function of temperature T and all other provided medium quantities are assumed to be constant.

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

---

Modelica.Media.Interfaces.PartialSimpleMedium.setState_pTX

Return thermodynamic state from p, T, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialSimpleMedium.setState_phX

Return thermodynamic state from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialSimpleMedium.setState_psX

Return thermodynamic state from p, s, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialSimpleMedium.setState_dTX

Return thermodynamic state from d, T, and X or Xi

Information

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialSimpleMedium.dynamicViscosity

Return dynamic viscosity

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Interfaces.PartialSimpleMedium.thermalConductivity

Return thermal conductivity

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Interfaces.PartialSimpleMedium.specificHeatCapacityCp

Return specific heat capacity at constant pressure

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Interfaces.PartialSimpleMedium.specificHeatCapacityCv

Return specific heat capacity at constant volume

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Interfaces.PartialSimpleMedium.isentropicExponent

Return isentropic exponent

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.Interfaces.PartialSimpleMedium.velocityOfSound

Return velocity of sound

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.Interfaces.PartialSimpleMedium.specificEnthalpy_pTX

Return specific enthalpy from p, T, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[nX]		Mass fractions [kg/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialSimpleMedium.temperature_phX

Return temperature from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[nX]		Mass fractions [kg/kg]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialSimpleMedium.density_phX

Return density from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[nX]		Mass fractions [kg/kg]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium

Medium model of Ideal gas with constant cp and cv. All other quantities, e.g. transport properties, are constant.

Package Content

Name	Description
cp_const	Constant specific heat capacity at constant pressure
cv_const=cp_const - R_gas	Constant specific heat capacity at constant volume
R_gas	medium specific gas constant
MM_const	Molar mass
eta_const	Constant dynamic viscosity
lambda_const	Constant thermal conductivity
T_min	Minimum temperature valid for medium model
T_max	Maximum temperature valid for medium model
T0=reference_T	Zero enthalpy temperature
ThermodynamicState	Thermodynamic state of ideal gas
BaseProperties	Base properties of ideal gas
setState_pTX	Return thermodynamic state from p, T, and X or Xi
setState_phX	Return thermodynamic state from p, h, and X or Xi
setState_psX	Return thermodynamic state from p, s, and X or Xi
setState_dTX	Return thermodynamic state from d, T, and X or Xi
pressure	Return pressure of ideal gas
temperature	Return temperature of ideal gas
density	Return density of ideal gas
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
specificHeatCapacityCp	Return specific heat capacity at constant pressure
specificHeatCapacityCv	Return specific heat capacity at constant volume
isentropicExponent	Return isentropic exponent
velocityOfSound	Return velocity of sound

specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
Inherited	
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid

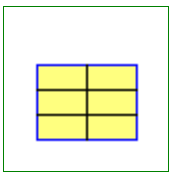
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
isentropicEnthalpy	Return isentropic enthalpy
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
density_pTX	Return density from p, T, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes

VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
constant SpecificHeatCapacity cp_const
"Constant specific heat capacity at constant pressure";

constant SpecificHeatCapacity cv_const= cp_const - R_gas
"Constant specific heat capacity at constant volume";

constant SpecificHeatCapacity R_gas "medium specific gas constant";

constant MolarMass MM_const "Molar mass";

constant DynamicViscosity eta_const "Constant dynamic viscosity";

constant ThermalConductivity lambda_const "Constant thermal conductivity";

constant Temperature T_min "Minimum temperature valid for medium model";

constant Temperature T_max "Maximum temperature valid for medium model";

constant Temperature T0= reference_T "Zero enthalpy temperature";
```

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.ThermodynamicState



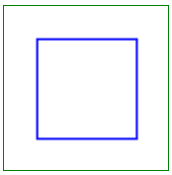
Thermodynamic state of ideal gas

Modelica definition

```
redeclare replaceable record extends ThermodynamicState
  "Thermodynamic state of ideal gas"
  AbsolutePressure p "Absolute pressure of medium";
```

```
  Temperature T "Temperature of medium";
end ThermodynamicState;
```

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.BaseProperties



Base properties of ideal gas

Information

This is the most simple incompressible medium model, where specific enthalpy h and specific internal energy u are only a function of temperature T and all other provided medium quantities are assumed to be constant.

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.setState_pTX

Return thermodynamic state from p, T, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.setState_phX

Return thermodynamic state from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description

AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.setState_psX

Return thermodynamic state from p, s, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.setState_dTX

Return thermodynamic state from d, T, and X or Xi

Information

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.pressure

Return pressure of ideal gas

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.temperature

Return temperature of ideal gas

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.density

Return density of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificEnthalpy

Return specific enthalpy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificInternalEnergy

Return specific internal energy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	u	Specific internal energy [J/kg]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificEntropy

Return specific entropy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificGibbsEnergy

Return specific Gibbs energy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	g	Specific Gibbs energy [J/kg]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificHelmholtzEnergy

Return specific Helmholtz energy

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	f	Specific Helmholtz energy [J/kg]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.dynamicViscosity

Return dynamic viscosity

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.thermalConductivity

Return thermal conductivity

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificHeatCapacityCp

Return specific heat capacity at constant pressure

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificHeatCapacityCv

Return specific heat capacity at constant volume

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.isentropicExponent

Return isentropic exponent

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.velocityOfSound

Return velocity of sound

Information

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.specificEnthalpy_pTX

Return specific enthalpy from p, T, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[nX]		Mass fractions [kg/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy at p, T, X [J/kg]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.temperature_phX

Return temperature from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[nX]		Mass fractions [kg/kg]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Interfaces.PartialSimpleIdealGasMedium.density_phX

Return density from p, h, and X or Xi

Information

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[nX]		Mass fractions [kg/kg]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Common

data structures and fundamental functions for fluid properties

Information

Package description

Package Modelica.Media.Common provides records and functions shared by many of the property sub-packages. High accuracy fluid property models share a lot of common structure, even if the actual models are different. Common data structures and computations shared by these property models are collected in this library.

Package Content

Name	Description
SaturationProperties	properties in the two phase region
SaturationBoundaryProperties	properties on both phase boundaries, including some derivatives

IF97BaseTwoPhase	Intermediate property data record for IF 97
IF97PhaseBoundaryProperties	thermodynamic base properties on the phase boundary for IF97 steam tables
GibbsDerivs	derivatives of dimensionless Gibbs-function w.r.t dimensionless pressure and temperature
HelmholtzDerivs	derivatives of dimensionless Helmholtz-function w.r.t dimensionless pressuredensity and temperature
TwoPhaseTransportProps	defines properties on both phase boundaries, needed in the two phase region
PhaseBoundaryProperties	thermodynamic base properties on the phase boundary
NewtonDerivatives_ph	derivatives for fast inverse calculations of Helmholtz functions: p & h
NewtonDerivatives_ps	derivatives for fast inverse calculation of Helmholtz functions: p & s
NewtonDerivatives_pT	derivatives for fast inverse calculations of Helmholtz functions:p & T
ExtraDerivatives	additional thermodynamic derivatives
BridgmansTables	Calculates all entries in Bridgmans tables if first seven variables given
gibbsToBridgmansTables	calculates base coefficients for bridgemans tables from gibbs enthalpy
helmholtzToBridgmansTables	calculates base coefficients for Bridgmans tables from helmholtz energy
gibbsToBoundaryProps	calulate phase boundary property record from dimensionless Gibbs function
helmholtzToBoundaryProps	calulate phase boundary property record from dimensionless Helmholtz function
cv2Phase	compute isochoric specific heat capacity inside the two-phase region
cvdpT2Phase	compute isochoric specific heat capacity inside the two-phase region and derivative of pressure w.r.t. temperature
gibbsToExtraDerivs	compute additional thermodynamic derivatives from dimensionless Gibbs function
helmholtzToExtraDerivs	compute additional thermodynamic derivatives from dimensionless Helmholtz function
Helmholtz_ph	function to calculate analytic derivatives for computing d and t given p and h
Helmholtz_pT	function to calculate analytic derivatives for computing d and t given p and t
Helmholtz_ps	function to calculate analytic derivatives for computing d and t given p and s
OneNonLinearEquation	Determine solution of a non-linear algebraic equation in one unknown without derivatives in a reliable and efficient way

Types and constants

```
  type Rate = Real (final quantity="Rate", final unit="s-1");

  type MolarFlowRate = Real (final quantity="MolarFlowRate", final
unit="mol/s");

  type MolarReactionRate = Real (final quantity="MolarReactionRate", final unit
      =  "mol/(m3.s)");

  type MolarEnthalpy = Real (final quantity="MolarEnthalpy", final
```

```
unit="J/mol");

  type DerDensityByEntropy = Real (final quantity="DerDensityByEntropy", final
unit
      =      "kg2.K/(m3.J)");

  type DerEnergyByPressure = Real (final quantity="DerEnergyByPressure", final
unit
      =      "J/Pa");

  type DerEnergyByMoles = Real (final quantity="DerEnergyByMoles", final unit=
         "J/mol");

  type DerEntropyByTemperature = Real (final quantity="DerEntropyByTemperature",
        final unit="J/K2");

  type DerEntropyByPressure = Real (final quantity="DerEntropyByPressure",
       final unit="J/(K.Pa)");

  type DerEntropyByMoles = Real (final quantity="DerEntropyByMoles", final unit
     =  "J/(mol.K)");

  type DerPressureByDensity = Real (final quantity="DerPressureByDensity",
       final unit="Pa.m3/kg");

  type DerPressureBySpecificVolume = Real (final quantity=
         "DerPressureBySpecificVolume", final unit="Pa.kg/m3");

  type DerPressureByTemperature = Real (final quantity=
         "DerPressureByTemperature", final unit="Pa/K");

  type DerVolumeByTemperature = Real (final quantity="DerVolumeByTemperature",
       final unit="m3/K");

  type DerVolumeByPressure = Real (final quantity="DerVolumeByPressure", final
unit
      =      "m3/Pa");

  type DerVolumeByMoles = Real (final quantity="DerVolumeByMoles", final unit=
         "m3/mol");

  type IsenthalpicExponent = Real (final quantity="IsenthalpicExponent", unit=
         "1");

  type IsentropicExponent = Real (final quantity="IsentropicExponent",
unit="1");

  type IsobaricVolumeExpansionCoefficient = Real (final quantity=
         "IsobaricVolumeExpansionCoefficient", unit="1/K");

  type IsochoricPressureCoefficient = Real (final quantity=
         "IsochoricPressureCoefficient", unit="1/K");

  type IsothermalCompressibility = Real (final quantity=
         "IsothermalCompressibility", unit="1/Pa");
```

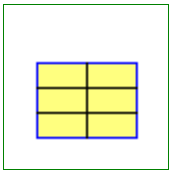
```
  type JouleThomsonCoefficient = Real (final quantity="JouleThomsonCoefficient",
         unit="K/Pa");

  constant Real MINPOS=1.0e-9
  "minimal value for physical variables which are always > 0.0";

  constant SI.Area AMIN=MINPOS "minimal init area";

  constant SI.Area AMAX=1.0e5 "maximal init area";

  constant SI.Area ANOM=1.0 "nominal init area";

  constant SI.AmountOfSubstance MOLMIN=-1.0*MINPOS "minimal Mole Number";

  constant SI.AmountOfSubstance MOLMAX=1.0e8 "maximal Mole Number";

  constant SI.AmountOfSubstance MOLNOM=1.0 "nominal Mole Number";

  constant SI.Density DMIN=MINPOS "minimal init density";

  constant SI.Density DMAX=1.0e5 "maximal init density";

  constant SI.Density DNOM=1.0 "nominal init density";

  constant SI.ThermalConductivity LAMMIN=MINPOS "minimal thermal conductivity";

  constant SI.ThermalConductivity LAMNOM=1.0 "nominal thermal conductivity";

  constant SI.ThermalConductivity LAMMAX=1000.0 "maximal thermal conductivity";

  constant SI.DynamicViscosity ETAMIN=MINPOS "minimal init dynamic viscosity";

  constant SI.DynamicViscosity ETAMAX=1.0e8 "maximal init dynamic viscosity";

  constant SI.DynamicViscosity ETANOM=100.0 "nominal init dynamic viscosity";

  constant SI.Energy EMIN=-1.0e10 "minimal init energy";

  constant SI.Energy EMAX=1.0e10 "maximal init energy";

  constant SI.Energy ENOM=1.0e3 "nominal init energy";

  constant SI.Entropy SMIN=-1.0e6 "minimal init entropy";

  constant SI.Entropy SMAX=1.0e6 "maximal init entropy";

  constant SI.Entropy SNOM=1.0e3 "nominal init entropy";

  constant SI.MassFlowRate MDOTMIN=-1.0e5 "minimal init mass flow rate";

  constant SI.MassFlowRate MDOTMAX=1.0e5 "maximal init mass flow rate";
```

```
constant SI.MassFlowRate MDOTNOM=1.0 "nominal init mass flow rate";

constant SI.MassFraction MASSXMIN=-1.0*MINPOS "minimal init mass fraction";

constant SI.MassFraction MASSXMAX=1.0 "maximal init mass fraction";

constant SI.MassFraction MASSXNOM=0.1 "nominal init mass fraction";

constant SI.Mass MMIN=-1.0*MINPOS "minimal init mass";

constant SI.Mass MMAX=1.0e8 "maximal init mass";

constant SI.Mass MNOM=1.0 "nominal init mass";

constant SI.MolarMass MMMIN=0.001 "minimal initial molar mass";

constant SI.MolarMass MMMAX=250.0 "maximal initial molar mass";

constant SI.MolarMass MMNOM=0.2 "nominal initial molar mass";

constant SI.MoleFraction MOLEYMIN=-1.0*MINPOS "minimal init mole fraction";

constant SI.MoleFraction MOLEYMAX=1.0 "maximal init mole fraction";

constant SI.MoleFraction MOLEYNOM=0.1 "nominal init mole fraction";

constant SI.MomentumFlux GMIN=-1.0e8 "minimal init momentum flux";

constant SI.MomentumFlux GMAX=1.0e8 "maximal init momentum flux";

constant SI.MomentumFlux GNOM=1.0 "nominal init momentum flux";

constant SI.Power POWMIN=-1.0e8 "minimal init power or heat";

constant SI.Power POWMAX=1.0e8 "maximal init power or heat";

constant SI.Power POWNOM=1.0e3 "nominal init power or heat";

constant SI.Pressure PMIN=1.0e4 "minimal init pressure";

constant SI.Pressure PMAX=1.0e8 "maximal init pressure";

constant SI.Pressure PNOM=1.0e5 "nominal init pressure";

constant SI.Pressure COMPPMIN=-1.0*MINPOS "minimal init pressure";

constant SI.Pressure COMPPMAX=1.0e8 "maximal init pressure";

constant SI.Pressure COMPPNOM=1.0e5 "nominal init pressure";
```

```
  constant SI.RatioOfSpecificHeatCapacities KAPPAMIN=1.0
  "minimal init isentropic exponent";

  constant SI.RatioOfSpecificHeatCapacities KAPPAMAX=1.7
  "maximal init isentropic exponent";

  constant SI.RatioOfSpecificHeatCapacities KAPPANOM=1.2
  "nominal init isentropic exponent";

  constant SI.SpecificEnergy SEMIN=-1.0e8 "minimal init specific energy";

  constant SI.SpecificEnergy SEMAX=1.0e8 "maximal init specific energy";

  constant SI.SpecificEnergy SENOM=1.0e6 "nominal init specific energy";

  constant SI.SpecificEnthalpy SHMIN=-1.0e8 "minimal init specific enthalpy";

  constant SI.SpecificEnthalpy SHMAX=1.0e8 "maximal init specific enthalpy";

  constant SI.SpecificEnthalpy SHNOM=1.0e6 "nominal init specific enthalpy";

  constant SI.SpecificEntropy SSMIN=-1.0e6 "minimal init specific entropy";

  constant SI.SpecificEntropy SSMAX=1.0e6 "maximal init specific entropy";

  constant SI.SpecificEntropy SSNOM=1.0e3 "nominal init specific entropy";

  constant SI.SpecificHeatCapacity CPMIN=MINPOS
  "minimal init specific heat capacity";

  constant SI.SpecificHeatCapacity CPMAX=1.0e6
  "maximal init specific heat capacity";

  constant SI.SpecificHeatCapacity CPNOM=1.0e3
  "nominal init specific heat capacity";

  constant SI.Temperature TMIN=MINPOS "minimal init temperature";

  constant SI.Temperature TMAX=1.0e5 "maximal init temperature";

  constant SI.Temperature TNOM=320.0 "nominal init temperature";

  constant SI.ThermalConductivity LMIN=MINPOS
  "minimal init thermal conductivity";

  constant SI.ThermalConductivity LMAX=500.0
  "maximal init thermal conductivity";

  constant SI.ThermalConductivity LNOM=1.0 "nominal init thermal conductivity";

  constant SI.Velocity VELMIN=-1.0e5 "minimal init speed";
```

```
  constant SI.Velocity VELMAX=1.0e5 "maximal init speed";

  constant SI.Velocity VELNOM=1.0 "nominal init speed";

  constant SI.Volume VMIN=0.0 "minimal init volume";

  constant SI.Volume VMAX=1.0e5 "maximal init volume";

  constant SI.Volume VNOM=1.0e-3 "nominal init volume";
```

Modelica.Media.Common.SaturationProperties

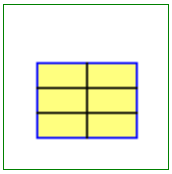


properties in the two phase region

Modelica definition

```
record SaturationProperties "properties in the two phase region"
  extends Modelica.Icons.Record;
  SI.Temp_K T "temperature";
  SI.Density d "density";
  SI.Pressure p "pressure";
  SI.SpecificEnergy u "specific inner energy";
  SI.SpecificEnthalpy h "specific enthalpy";
  SI.SpecificEntropy s "specific entropy";
  SI.SpecificHeatCapacity cp "heat capacity at constant pressure";
  SI.SpecificHeatCapacity cv "heat capacity at constant volume";
  SI.SpecificHeatCapacity R "gas constant";
  SI.RatioOfSpecificHeatCapacities kappa "isentropic expansion coefficient";
  PhaseBoundaryProperties liq
    "thermodynamic base properties on the boiling curve";
  PhaseBoundaryProperties vap "thermodynamic base properties on the dew curve";
  Real dpT(unit="Pa/K") "derivative of saturation pressure wrt temperature";
  SI.MassFraction x "vapour mass fraction";
end SaturationProperties;
```

Modelica.Media.Common.SaturationBoundaryProperties



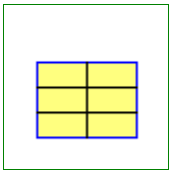
properties on both phase boundaries, including some derivatives

Modelica definition

```
record SaturationBoundaryProperties
  "properties on both phase boundaries, including some derivatives"

  extends Modelica.Icons.Record;
  SI.Temp_K T "Saturation temperature";
  SI.Density dl "Liquid density";
  SI.Density dv "Vapour density";
  SI.SpecificEnthalpy hl "Liquid specific enthalpy";
  SI.SpecificEnthalpy hv "Vapour specific enthalpy";
```

```
  Real dTp "derivative of temperature wrt saturation pressure";
  Real ddldp "derivative of density along boiling curve";
  Real ddvdp "derivative of density along dew curve";
  Real dhldp "derivative of specific enthalpy along boiling curve";
  Real dhvdp "derivative of specific enthalpy along dew curve";
  SI.MassFraction x "vapour mass fraction";
end SaturationBoundaryProperties;
```

Modelica.Media.Common.IF97BaseTwoPhase

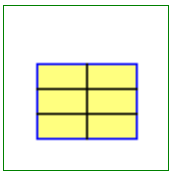


Intermediate property data record for IF 97

Modelica definition

```
record IF97BaseTwoPhase "Intermediate property data record for IF 97"
  extends Modelica.Icons.Record;
  Integer phase= 0 "phase: 2 for two-phase, 1 for one phase, 0 if unknown";
  Integer region(min=1, max=5) "IF 97 region";
  SI.Pressure p "pressure";
  SI.Temperature T "temperature";
  SI.SpecificEnthalpy h "specific enthalpy";
  SI.SpecificHeatCapacity R "gas constant";
  SI.SpecificHeatCapacity cp "specific heat capacity";
  SI.SpecificHeatCapacity cv "specific heat capacity";
  SI.Density rho "density";
  SI.SpecificEntropy s "specific entropy";
  DerPressureByTemperature pt "derivative of pressure wrt temperature";
  DerPressureByDensity pd "derivative of pressure wrt density";
  Real vt "derivative of specific volume w.r.t. temperature";
  Real vp "derivative of specific volume w.r.t. pressure";
  Real x "dryness fraction";
  Real dpT "dp/dT derivative of saturation curve";
end IF97BaseTwoPhase;
```

Modelica.Media.Common.IF97PhaseBoundaryProperties



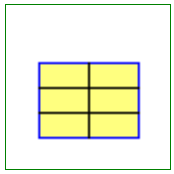
thermodynamic base properties on the phase boundary for IF97 steam tables

Modelica definition

```
record IF97PhaseBoundaryProperties
  "thermodynamic base properties on the phase boundary for IF97 steam tables"

  extends Modelica.Icons.Record;
  Boolean region3boundary "true if boundary between 2-phase and region 3";
  SI.SpecificHeatCapacity R "specific heat capacity";
  SI.Temperature T "temperature";
  SI.Density d "density";
  SI.SpecificEnthalpy h "specific enthalpy";
  SI.SpecificEntropy s "specific entropy";
  SI.SpecificHeatCapacity cp "heat capacity at constant pressure";
  SI.SpecificHeatCapacity cv "heat capacity at constant volume";
```

```
  DerPressureByTemperature dpT "dp/dT derivative of saturation curve";
  DerPressureByTemperature pt "derivative of pressure wrt temperature";
  DerPressureByDensity pd "derivative of pressure wrt density";
  Real vt(unit="m3/(kg.K)") "derivative of specific volume w.r.t. temperature";
  Real vp(unit="m3/(kg.Pa)") "derivative of specific volume w.r.t. pressure";
end IF97PhaseBoundaryProperties;
```

Modelica.Media.Common.GibbsDerivs

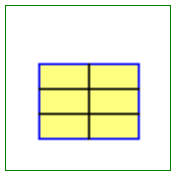


derivatives of dimensionless Gibbs-function w.r.t dimensionless pressure and temperature

Modelica definition

```
record GibbsDerivs
  "derivatives of dimensionless Gibbs-function w.r.t dimensionless pressure and
temperature"

  extends Modelica.Icons.Record;
  SI.Pressure p "pressure";
  SI.Temperature T "temperature";
  SI.SpecificHeatCapacity R "specific heat capacity";
  Real pi(unit="1") "dimensionless pressure";
  Real tau(unit="1") "dimensionless temperature";
  Real g(unit="1") "dimensionless Gibbs-function";
  Real gpi(unit="1") "derivative of g w.r.t. pi";
  Real gpipi(unit="1") "2nd derivative of g w.r.t. pi";
  Real gtau(unit="1") "derivative of g w.r.t. tau";
  Real gtautau(unit="1") "2nd derivative of g w.r.t tau";
  Real gtaupi(unit="1") "mixed derivative of g w.r.t. pi and tau";
end GibbsDerivs;
```

Modelica.Media.Common.HelmholtzDerivs



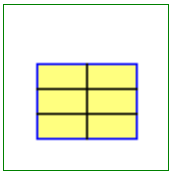
derivatives of dimensionless Helmholtz-function w.r.t dimensionless pressuredensity and temperature

Modelica definition

```
record HelmholtzDerivs
  "derivatives of dimensionless Helmholtz-function w.r.t dimensionless
pressuredensity and temperature"
  extends Modelica.Icons.Record;
  SI.Density d "density";
  SI.Temperature T "temperature";
  SI.SpecificHeatCapacity R "specific heat capacity";
  Real delta(unit="1") "dimensionless density";
  Real tau(unit="1") "dimensionless temperature";
  Real f(unit="1") "dimensionless Helmholtz-function";
  Real fdelta(unit="1") "derivative of f w.r.t. delta";
  Real fdeltadelta(unit="1") "2nd derivative of f w.r.t. delta";
  Real ftau(unit="1") "derivative of f w.r.t. tau";
```

```
  Real ftautau(unit="1") "2nd derivative of f w.r.t. tau";
  Real fdeltatau(unit="1") "mixed derivative of f w.r.t. delta and tau";
end HelmholtzDerivs;
```

Modelica.Media.Common.TwoPhaseTransportProps

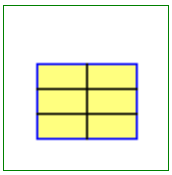


defines properties on both phase boundaries, needed in the two phase region

Modelica definition

```
record TwoPhaseTransportProps
  "defines properties on both phase boundaries, needed in the two phase region"
  extends Modelica.Icons.Record;
  SI.Density d_vap "density on the dew line";
  SI.Density d_liq "density on the bubble line";
  SI.DynamicViscosity eta_vap "dynamic viscosity on the dew line";
  SI.DynamicViscosity eta_liq "dynamic viscosity on the bubble line";
  SI.ThermalConductivity lam_vap "thermal conductivity on the dew line";
  SI.ThermalConductivity lam_liq "thermal conductivity on the bubble line";
  SI.SpecificHeatCapacity cp_vap "cp on the dew line";
  SI.SpecificHeatCapacity cp_liq "cp on the bubble line";
  SI.MassFraction x "steam quality";
end TwoPhaseTransportProps;
```

Modelica.Media.Common.PhaseBoundaryProperties

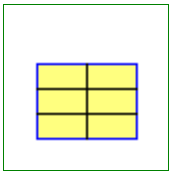


thermodynamic base properties on the phase boundary

Modelica definition

```
record PhaseBoundaryProperties
  "thermodynamic base properties on the phase boundary"
  extends Modelica.Icons.Record;
  SI.Density d "density";
  SI.SpecificEnthalpy h "specific enthalpy";
  SI.SpecificEnergy u "inner energy";
  SI.SpecificEntropy s "specific entropy";
  SI.SpecificHeatCapacity cp "heat capacity at constant pressure";
  SI.SpecificHeatCapacity cv "heat capacity at constant volume";
  DerPressureByTemperature pt "derivative of pressure wrt temperature";
  DerPressureByDensity pd "derivative of pressure wrt density";
end PhaseBoundaryProperties;
```

Modelica.Media.Common.NewtonDerivatives_ph

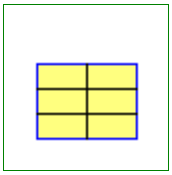


derivatives for fast inverse calculations of Helmholtz functions: p & h

Modelica definition

```
record NewtonDerivatives_ph
  "derivatives for fast inverse calculations of Helmholtz functions: p & h"

  extends Modelica.Icons.Record;
  SI.Pressure p "pressure";
  SI.SpecificEnthalpy h "specific enthalpy";
  DerPressureByDensity pd "derivative of pressure w.r.t. density";
  DerPressureByTemperature pt "derivative of pressure w.r.t. temperature";
  Real hd "derivative of specific enthalpy w.r.t. density";
  Real ht "derivative of specific enthalpy w.r.t. temperature";
end NewtonDerivatives_ph;
```

Modelica.Media.Common.NewtonDerivatives_ps

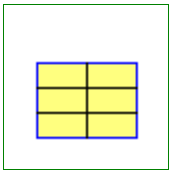


derivatives for fast inverse calculation of Helmholtz functions: p & s

Modelica definition

```
record NewtonDerivatives_ps
  "derivatives for fast inverse calculation of Helmholtz functions: p & s"

  extends Modelica.Icons.Record;
  SI.Pressure p "pressure";
  SI.SpecificEntropy s "specific entropy";
  DerPressureByDensity pd "derivative of pressure w.r.t. density";
  DerPressureByTemperature pt "derivative of pressure w.r.t. temperature";
  Real sd "derivative of specific entropy w.r.t. density";
  Real st "derivative of specific entropy w.r.t. temperature";
end NewtonDerivatives_ps;
```

Modelica.Media.Common.NewtonDerivatives_pT

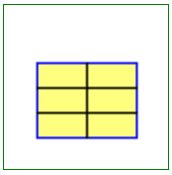


derivatives for fast inverse calculations of Helmholtz functions:p & T

Modelica definition

```
record NewtonDerivatives_pT
  "derivatives for fast inverse calculations of Helmholtz functions:p & T"

  extends Modelica.Icons.Record;
  SI.Pressure p "pressure";
  DerPressureByDensity pd "derivative of pressure w.r.t. density";
end NewtonDerivatives_pT;
```

Modelica.Media.Common.ExtraDerivatives

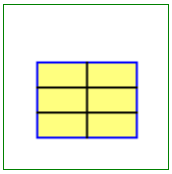


additional thermodynamic derivatives

Modelica definition

```
record ExtraDerivatives "additional thermodynamic derivatives"
  extends Modelica.Icons.Record;
  IsentropicExponent kappa "isentropic expansion coefficient";
  // k in Bejan
  IsenthalpicExponent theta "isenthalpic exponent";
  // same as kappa, except derivative at const h
  IsobaricVolumeExpansionCoefficient alpha
    "isobaric volume expansion coefficient";
  // beta in Bejan
  IsochoricPressureCoefficient beta "isochoric pressure coefficient";
  // kT in Bejan
  IsothermalCompressibility gamma "isothermal compressibility";
  // kappa in Bejan
  JouleThomsonCoefficient mu "Joule-Thomson coefficient";
  // mu_J in Bejan
end ExtraDerivatives;
```

Modelica.Media.Common.BridgmansTables



Calculates all entries in Bridgmans tables if first seven variables given

Information

Important: the phase equilibrium conditions are not yet considered. this means that bridgemans tables do not yet work in the two phase region. Some derivatives are 0 or infinity anyways. Idea: don't use the values in Bridgmans table directly, all derivatives are calculated as the quotient of two entries in the table. The last letter indicates which variable is held constant in taking the derivative. The second letters are the two variables involved in the derivative and the first letter is alwys a d to remind of differentiation.

```
Example 1: Get the derivative of specific entropy s wrt Temperature at
constant specific volume (btw identical to constant density)
constant volume  --> last letter v
Temperature      --> second letter T
Specific entropy --> second letter s
--> the needed value is dsv/dTv
Known variables:
Temperature T
pressure p
specific volume v
specific inner energy u
specific enthalpy h
specific entropy s
specific helmholtz energy f
specific gibbs enthalpy g
Not included but useful:
density d
In order to convert derivatives involving density use the following
rules:
at constant density == at constant specific volume
ddx/dyx = -d*d*dvx/dyx with y,x any of T,p,u,h,s,f,g
dyx/ddx = -1/(d*d)dyx/dvx with y,x any of T,p,u,h,s,f,g
Usage example assuming water as the medium:
model BridgmansTablesForWater
extends ThermoFluid.BaseClasses.MediumModels.Water.WaterSteamMedium_ph;
```

```
Real derOfsByTAtConstantv "derivative of sp. entropy by temperature at constant
sp. volume"
ThermoFluid.BaseClasses.MediumModels.Common.ExtraDerivatives dpro;
ThermoFluid.BaseClasses.MediumModels.Common.BridgmansTables bt;
equation
dpro = ThermoFluid.BaseClasses.MediumModels.SteamIF97.extraDerivs_pT(p[1],T[1]);
bt.p = p[1];
bt.T = T[1];
bt.v = 1/pro[1].d;
bt.s = pro[1].s;
bt.cp = pro[1].cp;
bt.alpha = dpro.alpha;
bt.gamma = dpro.gamma;
derOfsByTAtConstantv =  bt.dsv/bt.dTv;
                  ...
end BridgmansTablesForWater;
```

Modelica definition

```
record BridgmansTables
  "Calculates all entries in Bridgmans tables if first seven variables given"
  extends Modelica.Icons.Record;
  // the first 7 need to calculated in a function!
  SI.SpecificVolume v "specific volume";
  SI.Pressure p "pressure";
  SI.Temperature T "temperature";
  SI.SpecificEntropy s "specific entropy";
  SI.SpecificHeatCapacity cp "heat capacity at constant pressure";
  IsobaricVolumeExpansionCoefficient alpha
    "isobaric volume expansion coefficient";
  // beta in Bejan
  IsothermalCompressibility gamma "isothermal compressibility";
  // kappa in Bejan
  // Derivatives at constant pressure
  Real dTp=1 "coefficient in Bridgmans table, see info for usage";
  Real dpT=-dTp "coefficient in Bridgmans table, see info for usage";
  Real dvp=alpha*v "coefficient in Bridgmans table, see info for usage";
  Real dpv=-dvp "coefficient in Bridgmans table, see info for usage";
  Real dsp=cp/T "coefficient in Bridgmans table, see info for usage";
  Real dps=-dsp "coefficient in Bridgmans table, see info for usage";
  Real dup=cp - alpha*p*v "coefficient in Bridgmans table, see info for usage";
  Real dpu=-dup "coefficient in Bridgmans table, see info for usage";
  Real dhp=cp "coefficient in Bridgmans table, see info for usage";
  Real dph=-dhp "coefficient in Bridgmans table, see info for usage";
  Real dfp=-s - alpha*p*v "coefficient in Bridgmans table, see info for usage";
  Real dpf=-dfp "coefficient in Bridgmans table, see info for usage";
  Real dgp=-s "coefficient in Bridgmans table, see info for usage";
  Real dpg=-dgp "coefficient in Bridgmans table, see info for usage";
  // Derivatives at constant Temperature
  Real dvT=gamma*v "coefficient in Bridgmans table, see info for usage";
  Real dTv=-dvT "coefficient in Bridgmans table, see info for usage";
  Real dsT=alpha*v "coefficient in Bridgmans table, see info for usage";
  Real dTs=-dsT "coefficient in Bridgmans table, see info for usage";
  Real duT=alpha*T*v - gamma*p*v
```

```
    "coefficient in Bridgmans table, see info for usage";
  Real dTu=-duT "coefficient in Bridgmans table, see info for usage";
  Real dhT=-v + alpha*T*v "coefficient in Bridgmans table, see info for usage";
  Real dTh=-dhT "coefficient in Bridgmans table, see info for usage";
  Real dfT=-gamma*p*v "coefficient in Bridgmans table, see info for usage";
  Real dTf=-dfT "coefficient in Bridgmans table, see info for usage";
  Real dgT=-v "coefficient in Bridgmans table, see info for usage";
  Real dTg=-dgT "coefficient in Bridgmans table, see info for usage";
  // Derivatives at constant v
  Real dsv=alpha*alpha*v*v - gamma*v*cp/T
    "coefficient in Bridgmans table, see info for usage";
  Real dvs=-dsv "coefficient in Bridgmans table, see info for usage";
  Real duv=T*alpha*alpha*v*v - gamma*v*cp
    "coefficient in Bridgmans table, see info for usage";
  Real dvu=-duv "coefficient in Bridgmans table, see info for usage";
  Real dhv=T*alpha*alpha*v*v - alpha*v*v - gamma*v*cp
    "coefficient in Bridgmans table, see info for usage";
  Real dvh=-dhv "coefficient in Bridgmans table, see info for usage";
  Real dfv=gamma*v*s "coefficient in Bridgmans table, see info for usage";
  Real dvf=-dfv "coefficient in Bridgmans table, see info for usage";
  Real dgv=gamma*v*s - alpha*v*v
    "coefficient in Bridgmans table, see info for usage";
  Real dvg=-dgv "coefficient in Bridgmans table, see info for usage";
  // Derivatives at constant s
  Real dus=dsv*p "coefficient in Bridgmans table, see info for usage";
  Real dsu=-dus "coefficient in Bridgmans table, see info for usage";
  Real dhs=-v*cp/T "coefficient in Bridgmans table, see info for usage";
  Real dsh=-dhs "coefficient in Bridgmans table, see info for usage";
  Real dfs=alpha*v*s + dus "coefficient in Bridgmans table, see info for usage";
  Real dsf=-dfs "coefficient in Bridgmans table, see info for usage";
  Real dgs=alpha*v*s - v*cp/T
    "coefficient in Bridgmans table, see info for usage";
  Real dsg=-dgs "coefficient in Bridgmans table, see info for usage";
  // Derivatives at constant u
  Real dhu=p*alpha*v*v + gamma*v*cp*p - v*cp - p*T*alpha*alpha*v*v
    "coefficient in Bridgmans table, see info for usage";
  Real duh=-dhu "coefficient in Bridgmans table, see info for usage";
  Real dfu=s*T*alpha*v - gamma*v*cp*p - gamma*v*s*p + p*T*alpha*alpha*v*v
    "coefficient in Bridgmans table, see info for usage";
  Real duf=-dfu "coefficient in Bridgmans table, see info for usage";
  Real dgu=alpha*v*v*p + alpha*v*s*T - v*cp - gamma*v*s*p
    "coefficient in Bridgmans table, see info for usage";
  Real dug=-dgu "coefficient in Bridgmans table, see info for usage";
  //  Derivatives at constant h
  Real dfh=(s - v*alpha*p)*(v - v*alpha*T) - gamma*v*cp*p
    "coefficient in Bridgmans table, see info for usage";
  Real dhf=-dfh "coefficient in Bridgmans table, see info for usage";
  Real dgh=alpha*v*s*T - v*(s + cp)
    "coefficient in Bridgmans table, see info for usage";
  Real dhg=-dgh "coefficient in Bridgmans table, see info for usage";
  // Derivatives at constant g
  Real dfg=gamma*v*s*p - v*s - alpha*v*v*p
    "coefficient in Bridgmans table, see info for usage";
  Real dgf=-dfg "coefficient in Bridgmans table, see info for usage";
end BridgmansTables;
```

Modelica.Media.Common.gibbsToBridgmansTables

calculates base coefficients for bridgemans tables from gibbs enthalpy

Inputs

Type	Name	Default	Description
GibbsDerivs	g		dimensionless derivatives of Gibbs function

Outputs

Type	Name	Description
SpecificVolume	v	specific volume [m3/kg]
Pressure	p	pressure [Pa]
Temperature	T	temperature [K]
SpecificEntropy	s	specific entropy [J/(kg.K)]
SpecificHeatCapacity	cp	heat capacity at constant pressure [J/(kg.K)]
IsobaricVolumeExpansionCoefficient	alpha	isobaric volume expansion coefficient [1/K]
IsothermalCompressibility	gamma	isothermal compressibility [1/Pa]

Modelica.Media.Common.helmholtzToBridgmansTables

calculates base coefficients for Bridgmans tables from helmholtz energy

Inputs

Type	Name	Default	Description
HelmholtzDerivs	f		dimensionless derivatives of Helmholtz function

Outputs

Type	Name	Description
SpecificVolume	v	specific volume [m3/kg]
Pressure	p	pressure [Pa]
Temperature	T	temperature [K]
SpecificEntropy	s	specific entropy [J/(kg.K)]
SpecificHeatCapacity	cp	heat capacity at constant pressure [J/(kg.K)]
IsobaricVolumeExpansionCoefficient	alpha	isobaric volume expansion coefficient [1/K]
IsothermalCompressibility	gamma	isothermal compressibility [1/Pa]

Modelica.Media.Common.gibbsToBoundaryProps

calulate phase boundary property record from dimensionless Gibbs function

Inputs

Type	Name	Default	Description
GibbsDerivs	g		dimensionless derivatives of Gibbs function

Outputs

Type	Name	Description
PhaseBoundaryProperties	sat	phase boundary properties

Modelica.Media.Common.helmholtzToBoundaryProps

calulate phase boundary property record from dimensionless Helmholtz function

Inputs

Type	Name	Default	Description
HelmholtzDerivs	f		dimensionless derivatives of Helmholtz function

Outputs

Type	Name	Description
PhaseBoundaryProperties	sat	phase boundary property record

Modelica.Media.Common.cv2Phase

compute isochoric specific heat capacity inside the two-phase region

Inputs

Type	Name	Default	Description
PhaseBoundaryProperties	liq		properties on the boiling curve
PhaseBoundaryProperties	vap		properties on the condensation curve
MassFraction	x		vapour mass fraction [1]
Temperature	T		temperature [K]
Pressure	p		preoperties [Pa]

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	isochoric specific heat capacity [J/(kg.K)]

Modelica.Media.Common.cvdpT2Phase

compute isochoric specific heat capacity inside the two-phase region and derivative of pressure w.r.t. temperature

Inputs

Type	Name	Default	Description
PhaseBoundaryProperties	liq		properties on the boiling curve
PhaseBoundaryProperties	vap		properties on the condensation curve
MassFraction	x		vapour mass fraction [1]
Temperature	T		temperature [K]
Pressure	p		preoperties [Pa]

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	isochoric specific heat capacity [J/(kg.K)]
Real	dpT	derivative of pressure w.r.t. temperature

Modelica.Media.Common.gibbsToExtraDerivs

compute additional thermodynamic derivatives from dimensionless Gibbs function

Inputs

Type	Name	Default	Description
GibbsDerivs	g		dimensionless derivatives of Gibbs function

Outputs

Type	Name	Description
ExtraDerivatives	dpro	additional property derivatives

Modelica.Media.Common.helmholtzToExtraDerivs

compute additional thermodynamic derivatives from dimensionless Helmholtz function

Inputs

Type	Name	Default	Description
HelmholtzDerivs	f		dimensionless derivatives of Helmholtz function

Outputs

Type	Name	Description
ExtraDerivatives	dpro	additional property derivatives

Modelica.Media.Common.Helmholtz_ph

function to calculate analytic derivatives for computing d and t given p and h

Inputs

Type	Name	Default	Description
HelmholtzDerivs	f		dimensionless derivatives of Helmholtz function

Outputs

Type	Name	Description
NewtonDerivatives_ph	nderivs	derivatives for Newton iteration to calculate d and t from p and h

---

Modelica.Media.Common.Helmholtz_pT

function to calculate analytic derivatives for computing d and t given p and t

Inputs

Type	Name	Default	Description
HelmholtzDerivs	f		dimensionless derivatives of Helmholtz function

Outputs

Type	Name	Description
NewtonDerivatives_pT	nderivs	derivatives for Newton iteration to compute d and t from p and t

---

Modelica.Media.Common.Helmholtz_ps

function to calculate analytic derivatives for computing d and t given p and s

Inputs

Type	Name	Default	Description
HelmholtzDerivs	f		dimensionless derivatives of Helmholtz function

Outputs

Type	Name	Description
NewtonDerivatives_ps	nderivs	derivatives for Newton iteration to compute d and t from p and s

---

Modelica.Media.Common.OneNonLinearEquation

Determine solution of a non-linear algebraic equation in one unknown without derivatives in a reliable and efficient way

Information

This function should currently only be used in Modelica.Media, since it might be replaced in the future by another strategy, where the tool is responsible for the solution of the non-linear equation.

This library determines the solution of one non-linear algebraic equation "y=f(x)" in one unknown "x" in a reliable way. As input, the desired value y of the non-linear function has to be given, as well as an interval x_min, x_max that contains the solution, i.e., "f(x_min) - y" and "f(x_max) - y" must have a different sign. If possible, a smaller interval is computed by inverse quadratic interpolation (interpolating with a quadratic polynomial through the last 3 points and computing the zero). If this fails, bisection is used, which always reduces the interval by a factor of 2. The inverse quadratic interpolation method has superlinear convergence. This is roughly the same convergence rate as a globally convergent Newton method, but without the need to compute derivatives of the non-linear function. The solver function is a direct mapping of the Algol 60 procedure "zero" to Modelica, from:

Brent R.P.:
Algorithms for Minimization without derivatives. Prentice Hall, 1973, pp. 58-59.

Due to current limitations of the Modelica language (not possible to pass a function reference to a function), the construction to use this solver on a user-defined function is a bit complicated (this method is from Hans Olsson, Dynasim AB). A user has to provide a package in the following way:

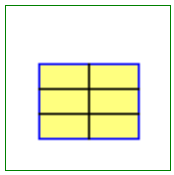
```
package MyNonLinearSolver
  extends OneNonLinearEquation;

  redeclare record extends Data
    // Define data to be passed to user function
    ...
  end Data;

  redeclare function extends f_nonlinear
  algorithm
     // Compute the non-linear equation: y = f(x, Data)
  end f_nonlinear;

  // Dummy definition that has to be present for current Dymola
  redeclare function extends solve
  end solve;
end MyNonLinearSolver;

x_zero = MyNonLinearSolver.solve(y_zero, x_min, x_max, data=data);
```

Package Content

Name	Description
f_nonlinear_Data	Data specific for function f_nonlinear
f_nonlinear	Nonlinear algebraic equation in one unknown: y = f_nonlinear(x,p,X)
solve	Solve f_nonlinear(x_zero)=y_zero; f_nonlinear(x_min) - y_zero and f_nonlinear(x_max)-y_zero must have different sign

---

Modelica.Media.Common.OneNonLinearEquation.f_nonlinear_Data



Data specific for function f_nonlinear

Modelica definition

```
replaceable record f_nonlinear_Data
  "Data specific for function f_nonlinear"
  extends Modelica.Icons.Record;
end f_nonlinear_Data;
```

Modelica.Media.Common.OneNonLinearEquation.f_nonlinear

Nonlinear algebraic equation in one unknown: y = f_nonlinear(x,p,X)

Inputs

Type	Name	Default	Description
Real	x		Independent variable of function
Real	p	0.0	disregaded variables (here always used for pressure)
Real	X[:]	fill(0, 0)	disregaded variables (her always used for

			composition)
f_nonlinear_Data	f_nonlinear_data		Additional data for the function

Outputs

Type	Name	Description
Real	y	= f_nonlinear(x)

Modelica.Media.Common.OneNonLinearEquation.solve

Solve f_nonlinear(x_zero)=y_zero; f_nonlinear(x_min) - y_zero and f_nonlinear(x_max)-y_zero must have different sign

Inputs

Type	Name	Default	Description
Real	y_zero		Determine x_zero, such that f_nonlinear(x_zero) = y_zero
Real	x_min		Minimum value of x
Real	x_max		Maximum value of x
Real	pressure	0.0	disregaded variables (here always used for pressure)
Real	X[:]	fill(0, 0)	disregaded variables (here always used for composition)
f_nonlinear_Data	f_nonlinear_data		Additional data for function f_nonlinear
Real	x_tol	100*Modelica.Constants.eps	Relative tolerance of the result

Outputs

Type	Name	Description
Real	x_zero	f_nonlinear(x_zero) = y_zero

Modelica.Media.Air

Medium models for air

Information

This package contains different medium models for air:

- **SimpleAir**
  Simple dry air medium in a limited temperature range.
- **DryAirNasa**
  Dry air as an ideal gas from Media.IdealGases.MixtureGases.Air.
- **MoistAir**
  Moist air as an ideal gas mixture of steam and dry air with fog below and above the triple point temperature.

Package Content

Name	Description
SimpleAir	Air: Simple dry air model (0..100 degC)
DryAirNasa	Air: Detailed dry air model as ideal gas (200..6000 K)
MoistAir	Air: Moist air model (240 ... 400 K)

Modelica.Media.Air.SimpleAir

Air: Simple dry air model (0..100 degC)

Information

Simple Ideal gas air model for low temperatures

This model demonstrats how to use the PartialSimpleIdealGas base class to build a simple ideal gas model with a limited temperature validity range.

Package Content

Name	Description
fluidConstants=FluidConstants(iupacName={"simple air"}, casRegistryNumber={"not a real substance"}, chemicalFormula={"N2, O2"}, structureFormula={"N2, O2"}, molarMass=Modelica.Media.IdealGases.Common.SingleGasesData.N2.MM)	constant data for the fluid
Inherited	
cp_const	Constant specific heat capacity at constant pressure
cv_const=cp_const - R_gas	Constant specific heat capacity at constant volume
R_gas	medium specific gas constant
MM_const	Molar mass
eta_const	Constant dynamic viscosity
lambda_const	Constant thermal conductivity
T_min	Minimum temperature valid for medium model
T_max	Maximum temperature valid for medium model
T0=reference_T	Zero enthalpy temperature
ThermodynamicState	Thermodynamic state of ideal gas
BaseProperties	Base properties of ideal gas
setState_pTX	Return thermodynamic state from p, T, and X or Xi
setState_phX	Return thermodynamic state from p, h, and X or Xi
setState_psX	Return thermodynamic state from p, s, and X or Xi
setState_dTX	Return thermodynamic state from

	d, T, and X or Xi
pressure	Return pressure of ideal gas
temperature	Return temperature of ideal gas
density	Return density of ideal gas
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
specificHeatCapacityCp	Return specific heat capacity at constant pressure
specificHeatCapacityCv	Return specific heat capacity at constant volume
isentropicExponent	Return isentropic exponent
velocityOfSound	Return velocity of sound
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
mediumName="unusablePartialMedium"	Name of the medium

substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
isentropicEnthalpy	Return isentropic enthalpy
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for

	user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
density_pTX	Return density from p, T, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with

	medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
  constant FluidConstants[nS] fluidConstants=
    FluidConstants(iupacName={"simple air"},
                   casRegistryNumber={"not a real substance"},
                   chemicalFormula={"N2, O2"},
                   structureFormula={"N2, O2"},
                   molarMass=Modelica.Media.IdealGases.Common.SingleGasesData.N2
.MM)
  "constant data for the fluid";
```

Modelica.Media.Air.DryAirNasa

Air: Detailed dry air model as ideal gas (200..6000 K)

Information

Package Content

Name	Description
dynamicViscosity	Simple polynomial for dry air (moisture influence small), valid from 73.15 K to 373.15 K
thermalConductivity	Simple polynomial for dry air (moisture influence small), valid from 73.15 K to 373.15 K
Inherited	
ThermodynamicState	thermodynamic state variables for ideal gases
FluidConstants	Extended fluid constants
excludeEnthalpyOfFormation=true	If true, enthalpy of formation Hf is not included in specific enthalpy h
referenceChoice=Choices.ReferenceEnthalpy.ZeroAt0K	Choice of reference enthalpy
h_offset=0.0	User defined offset for reference enthalpy, if referenceChoice = UserDefined
data	Data record of ideal gas substance
fluidConstants	constant data for the fluid
BaseProperties	Base properties of ideal gas medium
setState_pTX	Return thermodynamic state as function of p, T and composition X
setState_phX	Return thermodynamic state as function of p, h and composition X
setState_psX	Return thermodynamic state as function of p, s and composition X
setState_dTX	Return thermodynamic state as function of d, T and composition X
pressure	return pressure of ideal gas
temperature	return temperature of ideal gas
density	return density of ideal gas
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificHeatCapacityCp	Return specific heat capacity at constant pressure
specificHeatCapacityCv	Compute specific heat capacity at constant volume from temperature and gas data

isentropicExponent	Return isentropic exponent
velocityOfSound	Return velocity of sound
isentropicEnthalpyApproximation	approximate method of calculating h_is from upstream properties and downstream pressure
isentropicEnthalpy	Return isentropic enthalpy
isobaricExpansionCoefficient	Returns overall the isobaric expansion coefficient beta
isothermalCompressibility	Returns overall the isothermal compressibility factor
density_derp_T	density derivative by temperature at constant pressure
density_derT_p	density derivative by temperature at constant pressure
density_derX	density derivative by mass fraction
cp_T	Compute specific heat capacity at constant pressure from temperature and gas data
cp_Tlow	Compute specific heat capacity at constant pressure, low T region
cp_Tlow_der	Compute specific heat capacity at constant pressure, low T region
h_T	Compute specific enthalpy from temperature and gas data; reference is decided by the refChoice input, or by the referenceChoice package constant by default
h_T_der	derivative function for h_T
h_Tlow	Compute specific enthalpy, low T region; reference is decided by the refChoice input, or by the referenceChoice package constant by default
h_Tlow_der	Compute specific enthalpy, low T region; reference is decided by the refChoice input, or by the referenceChoice package constant by default
s0_T	Compute specific entropy from temperature and gas data
s0_Tlow	Compute specific entropy, low T region
dynamicViscosityLowPressure	Dynamic viscosity of low pressure gases
thermalConductivityEstimate	Thermal conductivity of polyatomic gases(Eucken and Modified Eucken correlation)
molarMass	return the molar mass of the medium
T_h	Compute temperature from specific enthalpy
T_ps	Compute temperature from pressure and specific entropy
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T

specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi

temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Modelica.Media.Air.DryAirNasa.dynamicViscosity

Simple polynomial for dry air (moisture influence small), valid from 73.15 K to 373.15 K

Information

Dynamic viscosity is computed from temperature using a second order polynomial with a range of validity between 73 and 373 K.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		Thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Air.DryAirNasa.thermalConductivity

Simple polynomial for dry air (moisture influence small), valid from 73.15 K to 373.15 K

Information

Thermal conductivity is computed from temperature using a second order polynomial with a range of validity between 73 and 373 K.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		Thermodynamic state record
Integer	method	1	Dummy for compatibility reasons

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Air.MoistAir

Air: Moist air model (240 ... 400 K)

Information

Thermodynamic Model

This package provides a full thermodynamic model of moist air including the fog region and temperatures below zero degC. The governing assumptions in this model are:

- the perfect gas law applies
- water volume other than that of steam is neglected

All extensive properties are expressed in terms of the total mass in order to comply with other media in this libary. However, for moist air it is rather common to express the absolute humidity in terms of mass of dry air only, which has advantages when working with charts. In addition, care must be taken, when working with mass fractions with respect to total mass, that all properties refer to the same water content when being used in mathematical operations (which is always the case if based on dry air only). Therefore two absolute

humidities are computed in the **BaseProperties** model: **X** denotes the absolute humidity in terms of the total mass while **x** denotes the absolute humitity per unit mass of dry air. In addition, the relative humidity **phi** is also computed.

At the triple point temperature of water of 0.01°C or 273.16 K and a relative humidity greater than 1 fog may be present as liquid and as ice resulting in a specific enthalpy somewhere between those of the two isotherms for solid and liquid fog, respectively. For numerical reasons a coexisting mixture of 50% solid and 50% liquid fog is assumed in the fog region at the triple point in this model.

Range of validity

From the assumptions mentioned above it follows that the **pressure** should be in the region around **atmospheric** conditions or below (a few bars may still be fine though). Additionally a very high water content at low temperatures would yield incorrect densities, because the volume of the liquid or solid phase would not be negligible anymore. The model does not provide information on limits for water drop size in the fog region or transport information for the actual condensation or evaporation process in combination with surfaces. All excess water which is not in its vapour state is assumed to be still present in the air regarding its energy but not in terms of its spatial extent.

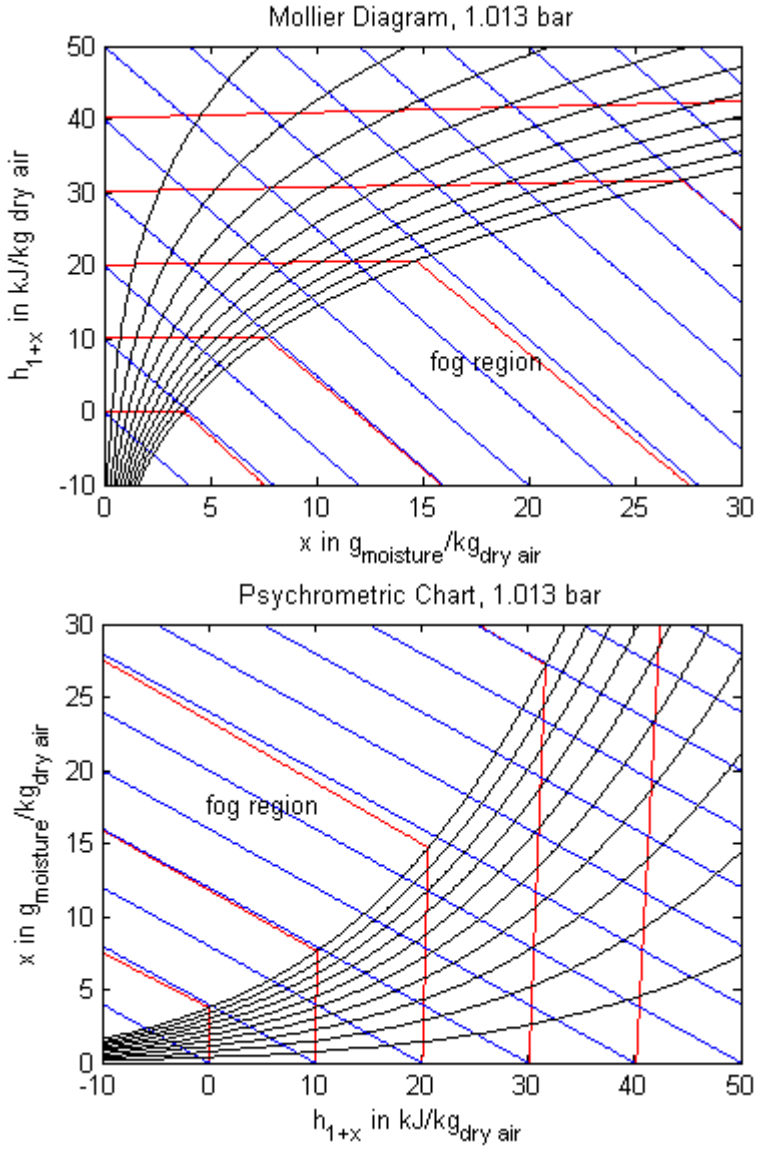
The thermodynamic model may be used for **temperatures** ranging from **240 - 400 K**. This holds for all functions unless otherwise stated in their description. However, although the model works at temperatures above the saturation temperature it is questionable to use the term "relative humidity" in this region. Please note, that although several functions compute pure water properties, they are designed to be used within the moist air medium model where properties are dominated by air and steam in their vapor states, and not for pure liquid water applications.

Transport Properties

Several additional functions that are not needed to describe the thermodynamic system, but are required to model transport processes, like heat and mass transfer, may be called. They usually neglect the moisture influence unless otherwise stated.

Application

The model's main area of application is all processes that involve moist air cooling under near atmospheric pressure with possible moisture condensation. This is the case in all domestic and industrial air conditioning applications. Another large domain of moist air applications covers all processes that deal with dehydration of bulk material using air as a transport medium. Engineering tasks involving moist air are often performed (or at least visualized) by using charts that contain all relevant thermodynamic data for a moist air system. These so called psychrometric charts can be generated from the medium properties in this package. The model PsychrometricData may be used for this purpose in order to obtain data for figures like those below (the plotting itself is not part of the model though).



Legend: blue - constant specific enthalpy, red - constant temperature, black - constant relative humidity

Package Content

Name	Description
Water=1	Index of water (in substanceNames, massFractions X, etc.)
Air=2	Index of air (in substanceNames, massFractions X, etc.)
k_mair=steam.MM/dryair.MM	ratio of molar weights
dryair=IdealGases.Common.SingleGasesData.Air	
steam=IdealGases.Common.SingleGasesData.H2O	
BaseProperties	Moist air base properties record
setState_pTX	Return thermodynamic state as function of pressure p, temperature T and composition X
setState_phX	Return thermodynamic state as function of pressure p, specific enthalpy h and composition X

setState_dTX	Return thermodynamic state as function of density d, temperature T and composition X
Xsaturation	Return absolute humitity per unit mass of moist air at saturation as a function of the thermodynamic state record
xsaturation	Return absolute humitity per unit mass of dry air at saturation as a function of the thermodynamic state record
xsaturation_pT	Return absolute humitity per unit mass of dry air at saturation as a function of pressure p and temperature T
massFraction_pTphi	Return steam mass fraction as a function of relative humidity phi and temperature T
relativeHumidity_pTX	Return relative humidity as a function of pressure p, temperature T and composition X
relativeHumidity	Return relative humidity as a function of the thermodynamic state record
gasConstant	Return ideal gas constant as a function from thermodynamic state, only valid for phi<1
gasConstant_X	Return ideal gas constant as a function from composition X
saturationPressureLiquid	Return saturation pressure of water as a function of temperature T in the range of 273.16 to 373.16 K
saturationPressureLiquid_der	Time derivative of saturationPressureLiquid
sublimationPressureIce	Return sublimation pressure of water as a function of temperature T between 223.16 and 273.16 K
sublimationPressureIce_der	Derivative function for 'sublimationPressureIce'
saturationPressure	Return saturation pressure of water as a function of temperature T between 223.16 and 373.16 K
saturationPressure_der	Derivative function for 'saturationPressure'
saturationTemperature	Return saturation temperature of water as a function of (partial) pressure p
enthalpyOfVaporization	Return enthalpy of vaporization of water as a function of temperature T, 0 - 130 degC
HeatCapacityOfWater	Return specific heat capacity of water (liquid only) as a function of temperature T
enthalpyOfLiquid	Return enthalpy of liquid water as a function of temperature T(use enthalpyOfWater instead)
enthalpyOfGas	Return specific enthalpy of gas (air and steam) as a function of temperature T and composition X
enthalpyOfCondensingGas	Return specific enthalpy of steam as a function of temperature T
enthalpyOfWater	Computes specific enthalpy of water (solid/liquid) near atmospheric pressure from temperature T
enthalpyOfWater_der	Derivative function of enthalpyOfWater
pressure	Returns pressure of ideal gas as a function of the thermodynamic state record
temperature	Return temperature of ideal gas as a function of the thermodynamic state record
T_phX	Return temperature as a function of pressure p, specific enthalpy h and composition X

density	Returns density of ideal gas as a function of the thermodynamic state record
specificEnthalpy	Return specific enthalpy of moist air as a function of the thermodynamic state record
h_pTX	Return specific enthalpy of moist air as a function of pressure p, temperature T and composition X
h_pTX_der	Derivative function of h_pTX
specificInternalEnergy	Return specific internal energy of moist air as a function of the thermodynamic state record
specificInternalEnergy_pTX	Return specific internal energy of moist air as a function of pressure p, temperature T and composition X
specificInternalEnergy_pTX_der	Derivative function for specificInternalEnergy_pTX
specificEntropy	Return specific entropy from thermodynamic state record, only valid for phi<1
specificGibbsEnergy	Return specific Gibbs energy as a function of the thermodynamic state record, only valid for phi<1
specificHelmholtzEnergy	Return specific Helmholtz energy as a function of the thermodynamic state record, only valid for phi<1
specificHeatCapacityCp	Return specific heat capacity at constant pressure as a function of the thermodynamic state record
specificHeatCapacityCv	Return specific heat capacity at constant volume as a function of the thermodynamic state record
dynamicViscosity	Return dynamic viscosity as a function of the thermodynamic state record, valid from 73.15 K to 373.15 K
thermalConductivity	Return thermal conductivity as a function of the thermodynamic state record, valid from 73.15 K to 373.15 K
Utilities	utility functions
PsychrometricData	Produces plot data for psychrometric charts
Inherited	
ThermodynamicState	thermodynamic state variables
FluidConstants	extended fluid constants
fluidConstants	constant data for the fluid
moleToMassFractions	Return mass fractions X from mole fractions
massToMoleFractions	Return mole fractions from mass fractions X
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere

reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
setState_psX	Return thermodynamic state as function of p, s and composition X or Xi
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
isentropicExponent	Return isentropic exponent
isentropicEnthalpy	Return isentropic enthalpy
velocityOfSound	Return velocity of sound
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi

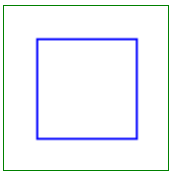
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes

Choices	Types, constants to define menu choices

Types and constants

```
  constant Integer Water=1
  "Index of water (in substanceNames, massFractions X, etc.)";

  constant Integer Air=2
  "Index of air (in substanceNames, massFractions X, etc.)";

  constant Real k_mair =  steam.MM/dryair.MM "ratio of molar weights";

  constant IdealGases.Common.DataRecord dryair =
IdealGases.Common.SingleGasesData.Air;

  constant IdealGases.Common.DataRecord steam =
IdealGases.Common.SingleGasesData.H2O;
```

Modelica.Media.Air.MoistAir.BaseProperties



Moist air base properties record

Information

This model computes thermodynamic properties of moist air from three independent (thermodynamic or/and numerical) state variables. Preferred numerical states are temperature T, pressure p and the reduced composition vector Xi, which contains the water mass fraction only. As an EOS the **ideal gas law** is used and associated restrictions apply. The model can also be used in the **fog region**, when moisture is present in its liquid state. However, it is assumed that the liquid water volume is negligible compared to that of the gas phase. Computation of thermal properties is based on property data of dry air and water (source: VDI-Wärmeatlas), respectively. Besides the standard thermodynamic variables **absolute and relative humidity**, x_water and phi, respectively, are given by the model. Upper case X denotes absolute humidity with respect to mass of moist air while absolute humidity with respect to mass of dry air only is denoted by a lower case x throughout the model. See package description for further information.

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

---

Modelica.Media.Air.MoistAir.setState_pTX

Return thermodynamic state as function of pressure p, temperature T and composition X

Information

The thermodynamic state record is computed from pressure p, temperature T and composition X.

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	Thermodynamic state

Modelica.Media.Air.MoistAir.setState_phX

Return thermodynamic state as function of pressure p, specific enthalpy h and composition X

Information

The thermodynamic state record is computed from pressure p, specific enthalpy h and composition X.

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	Thermodynamic state

Modelica.Media.Air.MoistAir.setState_dTX

Return thermodynamic state as function of density d, temperature T and composition X

Information

The thermodynamic state record is computed from density d, temperature T and composition X.

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	Thermodynamic state

Modelica.Media.Air.MoistAir.Xsaturation

Return absolute humitity per unit mass of moist air at saturation as a function of the thermodynamic state record

Information

Absolute humidity per unit mass of moist air at saturation is computed from pressure and temperature in the state record. Note, that unlike X_sat in the BaseProperties model this mass fraction refers to mass of moist air at saturation.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		Thermodynamic state record

Outputs

Type	Name	Description
MassFraction	X_sat	Steam mass fraction of sat. boundary [kg/kg]

Modelica.Media.Air.MoistAir.xsaturation

Return absolute humitity per unit mass of dry air at saturation as a function of the thermodynamic state record

Information

Absolute humidity per unit mass of dry air at saturation is computed from pressure and temperature in the thermodynamic state record.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		Thermodynamic state record

Outputs

Type	Name	Description
MassFraction	x_sat	Absolute humidity per unit mass of dry air [kg/kg]

Modelica.Media.Air.MoistAir.xsaturation_pT

Return absolute humitity per unit mass of dry air at saturation as a function of pressure p and temperature T

Information

Absolute humidity per unit mass of dry air at saturation is computed from pressure and temperature.

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
MassFraction	x_sat	Absolute humidity per unit mass of dry air [kg/kg]

Modelica.Media.Air.MoistAir.massFraction_pTphi

Return steam mass fraction as a function of relative humidity phi and temperature T

Information

Absolute humidity per unit mass of moist air is computed from temperature, pressure and relative humidity.

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
Real	phi		Relative humidity (0 ... 1.0)

Outputs

Type	Name	Description
MassFraction	X_steam	Absolute humidity, steam mass fraction [kg/kg]

Modelica.Media.Air.MoistAir.relativeHumidity_pTX

Return relative humidity as a function of pressure p, temperature T and composition X

Information

Relative humidity is computed from pressure, temperature and composition with 1.0 as the upper limit at saturation. Water mass fraction is the first entry in the composition vector.

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]		Composition [1]

Outputs

Type	Name	Description
Real	phi	Relative humidity

Modelica.Media.Air.MoistAir.relativeHumidity

Return relative humidity as a function of the thermodynamic state record

Information

Relative humidity is computed from the thermodynamic state record with 1.0 as the upper limit at saturation.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		Thermodynamic state

Outputs

Type	Name	Description
Real	phi	Relative humidity

Modelica.Media.Air.MoistAir.gasConstant

Return ideal gas constant as a function from thermodynamic state, only valid for phi<1

Information

The ideal gas constant for moist air is computed from thermodynamic state assuming that all water is in the gas phase.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state

Outputs

Type	Name	Description
SpecificHeatCapacity	R	mixture gas constant [J/(kg.K)]

Modelica.Media.Air.MoistAir.gasConstant_X

Return ideal gas constant as a function from composition X

Information

The ideal gas constant for moist air is computed from the gas phase composition. The first entry in composition vector X is the steam mass fraction of the gas phase.

Inputs

Type	Name	Default	Description
MassFraction	X[:]		Gas phase composition [1]

Outputs

Type	Name	Description
SpecificHeatCapacity	R	Ideal gas constant [J/(kg.K)]

Modelica.Media.Air.MoistAir.saturationPressureLiquid

Return saturation pressure of water as a function of temperature T in the range of 273.16 to 373.16 K

Information

Saturation pressure of water above the triple point temperature is computed from temperature. It's range of validity is between 273.16 and 373.16 K. Outside these limits a less accurate result is returned.

Inputs

Type	Name	Default	Description
Temperature	Tsat		saturation temperature [K]

Outputs

Type	Name	Description
AbsolutePressure	psat	saturation pressure [Pa]

Modelica.Media.Air.MoistAir.saturationPressureLiquid_der

Time derivative of saturationPressureLiquid

Information

Derivative function of saturationPressureLiquid

Inputs

Type	Name	Default	Description
Temperature	Tsat		Saturation temperature [K]
Real	dTsat		Saturation temperature derivative [K/s]

Outputs

Type	Name	Description
Real	psat_der	Saturation pressure [Pa/s]

Modelica.Media.Air.MoistAir.sublimationPressureIce

Return sublimation pressure of water as a function of temperature T between 223.16 and 273.16 K

Information

Sublimation pressure of water below the triple point temperature is computed from temperature. It's range of

validity is between 223.16 and 273.16 K. Outside of these limits a less accurate result is returned.

Inputs

Type	Name	Default	Description
Temperature	Tsat		sublimation temperature [K]

Outputs

Type	Name	Description
AbsolutePressure	psat	sublimation pressure [Pa]

Modelica.Media.Air.MoistAir.sublimationPressureIce_der

Derivative function for 'sublimationPressureIce'

Information

Derivative function of saturationPressureIce

Inputs

Type	Name	Default	Description
Temperature	Tsat		Sublimation temperature [K]
Real	dTsat		Time derivative of sublimation temperature [K/s]

Outputs

Type	Name	Description
Real	psat_der	Sublimation pressure [Pa/s]

Modelica.Media.Air.MoistAir.saturationPressure

Return saturation pressure of water as a function of temperature T between 223.16 and 373.16 K

Information

Saturation pressure of water in the liquid and the solid region is computed using an Antoine-type correlation. It's range of validity is between 223.16 and 373.16 K. Outside of these limits a (less accurate) result is returned. Functions for the solid and the liquid region, respectively, are combined using the first derivative continuous spliceFunction.

Inputs

Type	Name	Default	Description
Temperature	Tsat		saturation temperature [K]

Outputs

Type	Name	Description
AbsolutePressure	psat	saturation pressure [Pa]

Modelica.Media.Air.MoistAir.saturationPressure_der

Derivative function for 'saturationPressure'

Information

Derivative function of saturationPressure

Inputs

Type	Name	Default	Description
Temperature	Tsat		Saturation temperature [K]
Real	dTsat		Time derivative of saturation temperature [K/s]

Outputs

Type	Name	Description
Real	psat_der	Saturation pressure [Pa/s]

Modelica.Media.Air.MoistAir.saturationTemperature

Return saturation temperature of water as a function of (partial) pressure p

Information

Computes saturation temperature from (partial) pressure via numerical inversion of the function saturationPressure. Therefore additional inputs are required (or the defaults are used) for upper and lower temperature bounds.

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
Temperature	T_min	200	Lower boundary of solution [K]
Temperature	T_max	400	Upper boundary of solution [K]

Outputs

Type	Name	Description
Temperature	T	Saturation temperature [K]

Modelica.Media.Air.MoistAir.enthalpyOfVaporization

Return enthalpy of vaporization of water as a function of temperature T, 0 - 130 degC

Information

Enthalpy of vaporization of water is computed from temperature in the region of 0 to 130 °C.

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	r0	vaporization enthalpy [J/kg]

Modelica.Media.Air.MoistAir.HeatCapacityOfWater

Return specific heat capacity of water (liquid only) as a function of temperature T

Information

The specific heat capacity of water (liquid and solid) is calculated using a polynomial approach and data from VDI-Waermeatlas 8. Edition (Db1)

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificHeatCapacity	cp_fl	Specific heat capacity of liquid [J/(kg.K)]

Modelica.Media.Air.MoistAir.enthalpyOfLiquid

Return enthalpy of liquid water as a function of temperature T(use enthalpyOfWater instead)

Information

Specific enthalpy of liquid water is computed from temperature using a polynomial approach. Kept for compatibility reasons, better use enthalpyOfWater instead.

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	liquid enthalpy [J/kg]

Modelica.Media.Air.MoistAir.enthalpyOfGas

Return specific enthalpy of gas (air and steam) as a function of temperature T and

composition X

Information

Specific enthalpy of moist air is computed from temperature, provided all water is in the gaseous state. The first entry in the composition vector X must be the mass fraction of steam. For a function that also covers the fog region please refer to h_pTX.

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]
MassFraction	X[:]		vector of mass fractions [kg/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	liquid enthalpy [J/kg]

Modelica.Media.Air.MoistAir.enthalpyOfCondensingGas

Return specific enthalpy of steam as a function of temperature T

Information

Specific enthalpy of steam is computed from temperature.

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	liquid enthalpy [J/kg]

Modelica.Media.Air.MoistAir.enthalpyOfWater

Computes specific enthalpy of water (solid/liquid) near atmospheric pressure from temperature T

Information

Specific enthalpy of water (liquid and solid) is computed from temperature using constant properties as follows:

- heat capacity of liquid water:4200 J/kg
- heat capacity of solid water: 2050 J/kg
- enthalpy of fusion (liquid=>solid): 333000 J/kg

Pressure is assumed to be around 1 bar. This function is usually used to determine the specific enthalpy of the liquid or solid fraction of moist air.

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy of water [J/kg]

Modelica.Media.Air.MoistAir.enthalpyOfWater_der

Derivative function of enthalpyOfWater

Information

Derivative function for enthalpyOfWater.

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]
Real	dT		Time derivative of temperature [K/s]

Outputs

Type	Name	Description
Real	dh	Time derivative of specific enthalpy [J/(kg.s)]

Modelica.Media.Air.MoistAir.pressure

Returns pressure of ideal gas as a function of the thermodynamic state record

Information

Pressure is returned from the thermodynamic state record input as a simple assignment.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Air.MoistAir.temperature

Return temperature of ideal gas as a function of the thermodynamic state record

Information

Temperature is returned from the thermodynamic state record input as a simple assignment.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Air.MoistAir.T_phX

Return temperature as a function of pressure p, specific enthalpy h and composition X

Information

Temperature is computed from pressure, specific enthalpy and composition via numerical inversion of function h_pTX.

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]		Mass fractions of composition [kg/kg]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Air.MoistAir.density

Returns density of ideal gas as a function of the thermodynamic state record

Information

Density is computed from pressure, temperature and composition in the thermodynamic state record applying the ideal gas law.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description

Density	d	Density [kg/m3]

Modelica.Media.Air.MoistAir.specificEnthalpy

Return specific enthalpy of moist air as a function of the thermodynamic state record

Information

Specific enthalpy of moist air is computed from the thermodynamic state record. The fog region is included for both, ice and liquid fog.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Air.MoistAir.h_pTX

Return specific enthalpy of moist air as a function of pressure p, temperature T and composition X

Information

Specific enthalpy of moist air is computed from pressure, temperature and composition with X[1] as the total water mass fraction. The fog region is included for both, ice and liquid fog.

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]		Mass fractions of moist air [1]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy at p, T, X [J/kg]

Modelica.Media.Air.MoistAir.h_pTX_der

Derivative function of h_pTX

Information

Derivative function for h_pTX.

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]		Mass fractions of moist air [1]
Real	dp		Pressure derivative [Pa/s]
Real	dT		Temperature derivative [K/s]
Real	dX[:]		Composition derivative [1/s]

Outputs

Type	Name	Description
Real	h_der	Time derivative of specific enthalpy [J/(kg.s)]

Modelica.Media.Air.MoistAir.specificInternalEnergy

Return specific internal energy of moist air as a function of the thermodynamic state record

Information

Specific internal energy is determined from the thermodynamic state record, assuming that the liquid or solid water volume is negligible.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificInternalEnergy	u	Specific internal energy [J/kg]

Modelica.Media.Air.MoistAir.specificInternalEnergy_pTX

Return specific internal energy of moist air as a function of pressure p, temperature T and composition X

Information

Specific internal energy is determined from pressure p, temperature T and composition X, assuming that the liquid or solid water volume is negligible.

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]		Mass fractions of moist air [1]

Outputs

Type	Name	Description
SpecificInternalEnergy	u	Specific internal energy [J/kg]

Modelica.Media.Air.MoistAir.specificInternalEnergy_pTX_der

Derivative function for specificInternalEnergy_pTX

Information

Derivative function for specificInternalEnergy_pTX.

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]		Mass fractions of moist air [1]
Real	dp		Pressure derivative [Pa/s]
Real	dT		Temperature derivative [K/s]
Real	dX[:]		Mass fraction derivatives [1/s]

Outputs

Type	Name	Description
Real	u_der	Specific internal energy derivative [J/(kg.s)]

Modelica.Media.Air.MoistAir.specificEntropy

Return specific entropy from thermodynamic state record, only valid for phi<1

Information

Specific entropy is calculated from the thermodynamic state record, assuming ideal gas behavior and including entropy of mixing. Liquid or solid water is not taken into account, the entire water content X[1] is assumed to be in the vapor state (relative humidity below 1.0).

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.Air.MoistAir.specificGibbsEnergy

Return specific Gibbs energy as a function of the thermodynamic state record, only

valid for phi<1

Information

The Gibbs Energy is computed from the thermodynamic state record for moist air with a water content below saturation.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	g	Specific Gibbs energy [J/kg]

Modelica.Media.Air.MoistAir.specificHelmholtzEnergy

Return specific Helmholtz energy as a function of the thermodynamic state record, only valid for phi<1

Information

The Specific Helmholtz Energy is computed from the thermodynamic state record for moist air with a water content below saturation.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	f	Specific Helmholtz energy [J/kg]

Modelica.Media.Air.MoistAir.specificHeatCapacityCp

Return specific heat capacity at constant pressure as a function of the thermodynamic state record

Information

The specific heat capacity at constant pressure **cp** is computed from temperature and composition for a mixture of steam (X[1]) and dry air. All water is assumed to be in the vapor state.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Air.MoistAir.specificHeatCapacityCv

Return specific heat capacity at constant volume as a function of the thermodynamic state record

Information

The specific heat capacity at constant density **cv** is computed from temperature and composition for a mixture of steam (X[1]) and dry air. All water is assumed to be in the vapor state.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Air.MoistAir.dynamicViscosity

Return dynamic viscosity as a function of the thermodynamic state record, valid from 73.15 K to 373.15 K

Information

Dynamic viscosity is computed from temperature using a simple polynomial for dry air, assuming that moisture influence is small. Range of validity is from 73.15 K to 373.15 K.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Air.MoistAir.thermalConductivity

Return thermal conductivity as a function of the thermodynamic state record, valid from 73.15 K to 373.15 K

Information

Thermal conductivity is computed from temperature using a simple polynomial for dry air, assuming that moisture influence is small. Range of validity is from 73.15 K to 373.15 K.

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Air.MoistAir.Utilities

utility functions

Package Content

Name	Description
spliceFunction	Spline interpolation of two functions
spliceFunction_der	Derivative of spliceFunction

Modelica.Media.Air.MoistAir.Utilities.spliceFunction

Spline interpolation of two functions

Inputs

Type	Name	Default	Description
Real	pos		Returned value for x-deltax >= 0
Real	neg		Returned value for x+deltax <= 0
Real	x		Function argument
Real	deltax	1	Region around x with spline interpolation

Outputs

Type	Name	Description
Real	out	

Modelica.Media.Air.MoistAir.Utilities.spliceFunction_der

Derivative of spliceFunction

Inputs

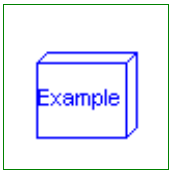
Type	Name	Default	Description
Real	pos		

Real	neg		
Real	x		
Real	deltax	1	
Real	dpos		
Real	dneg		
Real	dx		
Real	ddeltax	0	

Outputs

Type	Name	Description
Real	out	

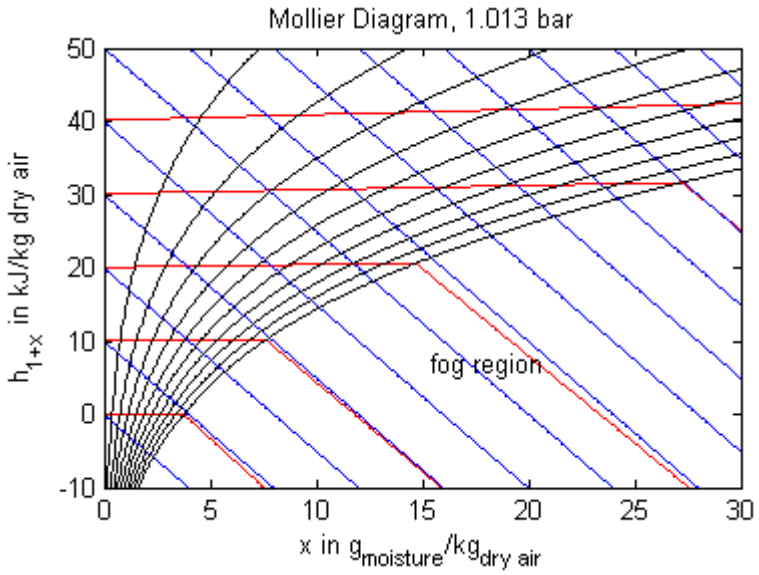
Modelica.Media.Air.MoistAir.PsychrometricData

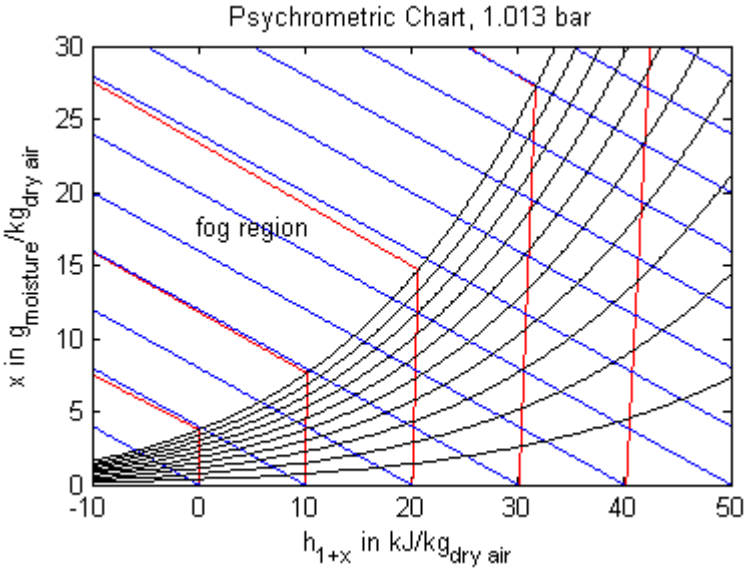


Produces plot data for psychrometric charts

Information

This model produces psychrometric data from the moist air model in this library to be plotted in charts. The two most common chart varieties are the Mollier Diagram and the Psycrometric Chart. The first is widely used in some European countries while the second is more common in the Anglo-American world. Specific enthalpy is plotted over absolute humidity in the Mollier Diagram, it is the other way round in the Psychrometric Chart.
It must be noted that the relationship of both axis variables is not right-angled, the absolute humidity follows a slope which equals the enthalpy of vaporization at 0°C. For better reading and in oder to reduce the fog region the humidity axis is rotated to obtain a right-angled plot. Both charts usually contain additional information as isochores or auxiliary scales for e.g. heat ratios. Those information are omitted in this model and the charts below. Other important features of psychrometric chart data are that all mass specific variables (like absolute humidity, specific enthalpy etc.) are expressed in terms of kg dry air and that their baseline of 0 enthalpy is found at 0°C and zero humidity.





Legend: blue - constant specific enthalpy, red - constant temperature, black - constant relative humidity

The model provides data for lines of constant specific enthalpy, temperature and relative humidity in a Mollier Diagram or Psychrometric Chart as they were used for the figures above. For limitations and ranges of validity please refer to the MoistAir package description. Absolute humidity **x** is increased with time in this model. The specific enthalpies adjusted for plotting are then obtained from:

- **y_h**: constant specific enthalpy
- **y_T**: constant temperature
- **y_phi**: constant relative humidity

Parameters

Type	Name	Default	Description
Pressure	p_const	1e5	Pressure [Pa]
Integer	n_T	11	Number of isotherms
Temperature	T_min	253.15	Lowest isotherm [K]
Temperature	T_step	10	Temperature step between two isotherms [K]
Integer	n_h	16	Number of lines with constant specific enthalpy
SpecificEnthalpy	h_min	-20e3	Lowest line of constant enthalpy [J/kg]
SpecificEnthalpy	h_step	1e4	Enthalpy step between two lines of constant enthalpy [J/kg]
Integer	n_phi	10	Number of lines with constant relative humidity
Real	phi_min	0.1	Lowest line of constant humidity
Real	phi_step	0.1	Step between two lines of constant humidity
MassFraction	x_min	0.00	Minimum diagram absolute humidity [1]
MassFraction	x_max	0.03	Maximum diagram absolute humidity [1]
Time	t	1	Simulation time [s]

Modelica.Media.CompressibleLiquids

compressible liquid models

Information

Fluid models with linear compressibility, using PartialLinearFluid as base class.

The linear compressibility fluid models contained in this package are based on the assumptions that:

- The specific heat capacity at constant pressure (cp) is constant
- The isobaric expansion coefficient (beta) is constant
- The isothermal compressibility (kappa) is constant
- Pressure and temperature are used as states

This results in models that are only valid for small temperature ranges, but sufficient to model compressibility and e.g. the "water hammer" effect. ANother advantage id that only 3 values need to be measured to have an initial model. Hydraulic fluids can often be approximated by this type of model.

That means that the density is a linear function in temperature and in pressure. In order to define the complete model, a number of constant reference values are needed which are computed at the reference values of the states pressure p and temperature T. The model can be interprested as a linearization of a full non-linear fluid model (but it is not linear in all thermodynamic coordinates). Reference values are needed for

1. the density (reference_d),
2. the specific enthalpy (reference_h),
3. the specific entropy (reference_s).

Apart from that, a user needs to define the molar mass, MM_const. Note that it is possible to define a fluid by computing the reference values from a full non-linear fluid model by computing the package constants using the standard functions defined in a fluid package (see example in Common, LinearWater_pT).

Package Content

Name	Description
Common	base classes for compressible liquids
LinearColdWater	cold water model with linear compressibility
LinearWater_pT_Ambient	liquid, linear compressibility water model at 1.01325 bar and 25 degree Celsius

---

Modelica.Media.CompressibleLiquids.Common

base classes for compressible liquids

Package Content

Name	Description
LinearWater_pT	base class for liquid, linear compressibility water models

---

Modelica.Media.CompressibleLiquids.Common.LinearWater_pT

base class for liquid, linear compressibility water models

Package Content

Name	Description
state=Modelica.Media.Water.StandardWater.setState_pT (reference_p, reference_T)	

Inherited	
cp_const	Specific heat capacity at constant pressure
beta_const	Thermal expansion coefficient at constant pressure
kappa_const	Isothermal compressibility
MM_const	Molar mass
reference_d	Density in reference conditions
reference_h	Specific enthalpy in reference conditions
reference_s	Specific enthalpy in reference conditions
constantJacobian	if true, entries in thermodynamic Jacobian are constant, taken at reference conditions
ThermodynamicState	a selction of variables that uniquely defines the thermodynamic state
BaseProperties	Base properties of medium
setState_pTX	set the thermodynamic state record from p and T (X not needed)
setState_phX	set the thermodynamic state record from p and h (X not needed)
setState_dTX	set the thermodynamic state record from d and T (X not needed)
setState_psX	set the thermodynamic state record from p and s (X not needed)
pressure	Return the pressure from the thermodynamic state
temperature	Return the temperature from the thermodynamic state
density	Return the density from the thermodynamic state
specificEnthalpy	Return the specific enthalpy from the thermodynamic state
specificEntropy	Return the specific entropy from the thermodynamic state
specificInternalEnergy	Return the specific internal energy from the thermodynamic state
specificGibbsEnergy	Return specific Gibbs energy from the thermodynamic state
specificHelmholtzEnergy	Return specific Helmholtz energy from the thermodynamic state
velocityOfSound	Return velocity of sound from the thermodynamic state
isentropicExponent	Return isentropic exponent from the thermodynamic state
isentropicEnthalpy	Return isentropic enthalpy
specificHeatCapacityCp	Return specific heat capacity at constant volume
specificHeatCapacityCv	Return specific heat capacity at constant volume from the thermodynamic state
isothermalCompressibility	Return the iso-thermal compressibility kappa
isobaricExpansionCoefficient	Return the iso-baric expansion coefficient
density_derp_h	Return density derivative wrt pressure at const specific enthalpy

density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
molarMass	Return molar mass
T_ph	Return temperature from pressure and specific enthalpy
T_ps	Return temperature from pressure and specific entropy
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)

h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
BasePropertiesRecord	Variables contained in every instance of BaseProperties
dynamicViscosity	Return dynamic viscosity
thermalConductivity	Return thermal conductivity
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derX	Return density derivative wrt mass fraction
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific

	attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
constant Modelica.Media.Water.StandardWater.ThermodynamicState state=
    Modelica.Media.Water.StandardWater.setState_pT(reference_p, reference_T);
```

Modelica.Media.CompressibleLiquids.LinearColdWater

cold water model with linear compressibility

Package Content

Name	Description
dynamicViscosity	Dynamic viscosity of water
thermalConductivity	Thermal conductivity of water
Inherited	
cp_const	Specific heat capacity at constant pressure
beta_const	Thermal expansion coefficient at constant pressure
kappa_const	Isothermal compressibility
MM_const	Molar mass
reference_d	Density in reference conditions
reference_h	Specific enthalpy in reference conditions
reference_s	Specific enthalpy in reference conditions
constantJacobian	if true, entries in thermodynamic Jacobian are constant, taken at reference conditions
ThermodynamicState	a selction of variables that uniquely defines the thermodynamic state
BaseProperties	Base properties of medium
setState_pTX	set the thermodynamic state record from p and T (X not needed)
setState_phX	set the thermodynamic state record from p and h (X not needed)
setState_dTX	set the thermodynamic state record from d and T (X not needed)
setState_psX	set the thermodynamic state record from p and s (X not needed)
pressure	Return the pressure from the thermodynamic state
temperature	Return the temperature from the thermodynamic state
density	Return the density from the thermodynamic state
specificEnthalpy	Return the specific enthalpy from the thermodynamic state
specificEntropy	Return the specific entropy from the thermodynamic state
specificInternalEnergy	Return the specific internal energy from the thermodynamic state
specificGibbsEnergy	Return specific Gibbs energy from the thermodynamic state
specificHelmholtzEnergy	Return specific Helmholtz energy from the thermodynamic state
velocityOfSound	Return velocity of sound from the thermodynamic state
isentropicExponent	Return isentropic exponent from the thermodynamic state
isentropicEnthalpy	Return isentropic enthalpy
specificHeatCapacityCp	Return specific heat capacity at constant volume
specificHeatCapacityCv	Return specific heat capacity at constant volume from the thermodynamic state

isothermalCompressibility	Return the iso-thermal compressibility kappa
isobaricExpansionCoefficient	Return the iso-baric expansion coefficient
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
molarMass	Return molar mass
T_ph	Return temperature from pressure and specific enthalpy
T_ps	Return temperature from pressure and specific entropy
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)

X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derX	Return density derivative wrt mass fraction
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes

VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Modelica.Media.CompressibleLiquids.LinearColdWater.dynamicViscosity

Dynamic viscosity of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.CompressibleLiquids.LinearColdWater.thermalConductivity

Thermal conductivity of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.CompressibleLiquids.LinearWater_pT_Ambient

liquid, linear compressibility water model at 1.01325 bar and 25 degree Celsius

Information

Water model with linear compressibility at ambient conditions

---

Modelica.Media.IdealGases

Data and models of ideal gases (single, fixed and dynamic mixtures) from NASA source

Information

This package contains data for the 1241 ideal gases from

> McBride B.J., Zehe M.J., and Gordon S. (2002): **NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species**. NASA report TP-2002-211556

Medium models for some of these gases are available in package IdealGases.SingleGases and some examples for mixtures are available in package IdealGases.MixtureGases

Using and Adapting Medium Models

The data records allow computing the ideal gas specific enthalpy, specific entropy and heat capacity of the substances listed below. From them, even the Gibbs energy and equilibrium constants for reactions can be computed. Critical data that is needed for computing the viscosity and thermal conductivity is not included. In order to add mixtures or single substance medium packages that are subtypes of Interfaces.PartialMedium (i.e., can be utilized at all places where PartialMedium is defined), a few additional steps have to be performed:

1. All single gas media need to define a constant instance of record IdealGases.Common.SingleGasNasa.FluidConstants. For 37 ideal gases such records are provided in package IdealGases.Common.FluidData. For the other gases, such a record instance has to be provided by the user, e.g. by getting the data from a commercial or public data base. A public source of the needed data is for example the NIST Chemistry WebBook
2. When the data is available, and a user has an instance of a FluidConstants record filled with data, a medium package has to be written. Note that only the dipole moment, the accentric factor and critical data are necessary for the viscosity and thermal conductivity functions.
    - For single components, a new package following the pattern in IdealGases.SingleGases has to be created, pointing both to a data record for cp and to a user-defined fluidContants record.
    - For mixtures of several components, a new package following the pattern in IdealGases.MixtureGases has to be created, building an array of data records for cp and an array of (partly) user-defined fluidContants records.

Note that many properties can computed for the full set of 1241 gases listed below, but due to the missing viscosity and thermal conductivity functions, no fully Modelica.Media-compliant media can be defined.

Data records for heat capacity, specific enthalpy and specific entropy exist for the following substances and ions:

Ag	BaOH+	C2H4O_ethylen_o	DF	In2I4	Nb	ScO2
Ag+	Ba_OH_2	CH3CHO_ethanal	DOCl	In2I6	Nb+	Sc2O
Ag-	BaS	CH3COOH	DO2	In2O	Nb-	Sc2O2
Air	Ba2	OHCH2COOH	DO2-	K	NbCl5	Si
Al	Be	C2H5	D2	K+	NbO	Si+
Al+	Be+	C2H5Br	D2+	K-	NbOCl3	Si-
Al-	Be++	C2H6	D2-	KAlF4	NbO2	SiBr
AlBr	BeBr	CH3N2CH3	D2O	KBO2	Ne	SiBr2
AlBr2	BeBr2	C2H5OH	D2O2	KBr	Ne+	SiBr3
AlBr3	BeCl	CH3OCH3	D2S	KCN	Ni	SiBr4

AlC	BeCl2	CH3O2CH3	e-	KCl	Ni+	SiC
AlC2	BeF	CCN	F	KF	Ni-	SiC2
AlCl	BeF2	CNC	F+	KH	NiCl	SiCl
AlCl+	BeH	OCCN	F-	KI	NiCl2	SiCl2
AlCl2	BeH+	C2N2	FCN	Kli	NiO	SiCl3
AlCl3	BeH2	C2O	FCO	KNO2	NiS	SiCl4
AlF	BeI	C3	FO	KNO3	O	SiF
AlF+	BeI2	C3H3_1_propynl	FO2_FOO	KNa	O+	SiFCl
AlFCl	BeN	C3H3_2_propynl	FO2_OFO	KO	O-	SiF2
AlFCl2	BeO	C3H4_allene	F2	KOH	OD	SiF3
AlF2	BeOH	C3H4_propyne	F2O	K2	OD-	SiF4
AlF2-	BeOH+	C3H4_cyclo	F2O2	K2+	OH	SiH
AlF2Cl	Be_OH_2	C3H5_allyl	FS2F	K2Br2	OH+	SiH+
AlF3	BeS	C3H6_propylene	Fe	K2CO3	OH-	SiHBr3
AlF4-	Be2	C3H6_cyclo	Fe+	K2C2N2	O2	SiHCl
AlH	Be2Cl4	C3H6O_propylox	Fe_CO_5	K2Cl2	O2+	SiHCl3
AlHCl	Be2F4	C3H6O_acetone	FeCl	K2F2	O2-	SiHF
AlHCl2	Be2O	C3H6O_propanal	FeCl2	K2I2	O3	SiHF3
AlHF	Be2OF2	C3H7_n_propyl	FeCl3	K2O	P	SiHI3
AlHFCl	Be2O2	C3H7_i_propyl	FeO	K2O+	P+	SiH2
AlHF2	Be3O3	C3H8	Fe_OH_2	K2O2	P-	SiH2Br2
AlH2	Be4O4	C3H8O_1propanol	Fe2Cl4	K2O2H2	PCl	SiH2Cl2
AlH2Cl	Br	C3H8O_2propanol	Fe2Cl6	K2SO4	PCl2	SiH2F2
AlH2F	Br+	CNCOCN	Ga	Kr	PCl2-	SiH2I2
AlH3	Br-	C3O2	Ga+	Kr+	PCl3	SiH3
AlI	BrCl	C4	GaBr	li	PCl5	SiH3Br
AlI2	BrF	C4H2_butadiyne	GaBr2	li+	PF	SiH3Cl
AlI3	BrF3	C4H4_1_3-cyclo	GaBr3	li-	PF+	SiH3F
AlN	BrF5	C4H6_butadiene	GaCl	liAlF4	PF-	SiH3I
AlO	BrO	C4H6_1butyne	GaCl2	liBO2	PFCl	SiH4
AlO+	OBrO	C4H6_2butyne	GaCl3	liBr	PFCl-	SiI
AlO-	BrOO	C4H6_cyclo	GaF	liCl	PFCl2	SiI2
AlOCl	BrO3	C4H8_1_butene	GaF2	liF	PFCl4	SiN
AlOCl2	Br2	C4H8_cis2_buten	GaF3	liH	PF2	SiO
AlOF	BrBrO	C4H8_isobutene	GaH	liI	PF2-	SiO2
AlOF2	BrOBr	C4H8_cyclo	GaI	liN	PF2Cl	SiS
AlOF2-	C	C4H9_n_butyl	GaI2	liNO2	PF2Cl3	SiS2
AlOH	C+	C4H9_i_butyl	GaI3	liNO3	PF3	Si2
AlOHCl	C-	C4H9_s_butyl	GaO	liO	PF3Cl2	Si2C
AlOHCl2	CBr	C4H9_t_butyl	GaOH	liOF	PF4Cl	Si2F6
AlOHF	CBr2	C4H10_n_butane	Ga2Br2	liOH	PF5	Si2N
AlOHF2	CBr3	C4H10_isobutane	Ga2Br4	liON	PH	Si3
AlO2	CBr4	C4N2	Ga2Br6	li2	PH2	Sn
AlO2-	CCl	C5	Ga2Cl2	li2+	PH2-	Sn+
Al_OH_2	CCl2	C5H6_1_3cyclo	Ga2Cl4	li2Br2	PH3	Sn-
Al_OH_2Cl	CCl2Br2	C5H8_cyclo	Ga2Cl6	li2F2	PN	SnBr
Al_OH_2F	CCl3	C5H10_1_pentene	Ga2F2	li2I2	PO	SnBr2
Al_OH_3	CCl3Br	C5H10_cyclo	Ga2F4	li2O	PO-	SnBr3
AlS	CCl4	C5H11_pentyl	Ga2F6	li2O+	POCl3	SnBr4
AlS2	CF	C5H11_t_pentyl	Ga2I2	li2O2	POFCl2	SnCl
Al2	CF+	C5H12_n_pentane	Ga2I4	li2O2H2	POF2Cl	SnCl2
Al2Br6	CFBr3	C5H12_i_pentane	Ga2I6	li2SO4	POF3	SnCl3
Al2C2	CFCl	CH3C_CH3_2CH3	Ga2O	li3+	PO2	SnCl4
Al2Cl6	CFClBr2	C6D5_phenyl	Ge	li3Br3	PO2-	SnF
Al2F6	CFCl2	C6D6	Ge+	li3Cl3	PS	SnF2
Al2I6	CFCl2Br	C6H2	Ge-	li3F3	P2	SnF3
Al2O	CFCl3	C6H5_phenyl	GeBr	li3I3	P2O3	SnF4
Al2O+	CF2	C6H5O_phenoxy	GeBr2	Mg	P2O4	SnI
Al2O2	CF2+	C6H6	GeBr3	Mg+	P2O5	SnI2

Al2O2+	CF2Br2	C6H5OH_phenol	GeBr4	MgBr	P3	SnI3
Al2O3	CF2Cl	C6H10_cyclo	GeCl	MgBr2	P3O6	SnI4
Al2S	CF2ClBr	C6H12_1_hexene	GeCl2	MgCl	P4	SnO
Al2S2	CF2Cl2	C6H12_cyclo	GeCl3	MgCl+	P4O6	SnO2
Ar	CF3	C6H13_n_hexyl	GeCl4	MgCl2	P4O7	SnS
Ar+	CF3+	C6H14_n_hexane	GeF	MgF	P4O8	SnS2
B	CF3Br	C7H7_benzyl	GeF2	MgF+	P4O9	Sn2
B+	CF3Cl	C7H8	GeF3	MgF2	P4O10	Sr
B-	CF4	C7H8O_cresol_mx	GeF4	MgF2+	Pb	Sr+
BBr	CH+	C7H14_1_heptene	GeH4	MgH	Pb+	SrBr
BBr2	CHBr3	C7H15_n_heptyl	GeI	MgI	Pb-	SrBr2
BBr3	CHCl	C7H16_n_heptane	GeO	MgI2	PbBr	SrCl
BC	CHClBr2	C7H16_2_methylh	GeO2	MgN	PbBr2	SrCl+
BC2	CHCl2	C8H8_styrene	GeS	MgO	PbBr3	SrCl2
BCl	CHCl2Br	C8H10_ethylbenz	GeS2	MgOH	PbBr4	SrF
BCl+	CHCl3	C8H16_1_octene	Ge2	MgOH+	PbCl	SrF+
BClOH	CHF	C8H17_n_octyl	H	Mg_OH_2	PbCl2	SrF2
BCl_OH_2	CHFBr2	C8H18_n_octane	H+	MgS	PbCl3	SrH
BCl2	CHFCl	C8H18_isooctane	H-	Mg2	PbCl4	SrI
BCl2+	CHFClBr	C9H19_n_nonyl	HAlO	Mg2F4	PbF	SrI2
BCl2OH	CHFCl2	C10H8_naphthale	HAlO2	Mn	PbF2	SrO
BF	CHF2	C10H21_n_decyl	HBO	Mn+	PbF3	SrOH
BFCl	CHF2Br	C12H9_o_bipheny	HBO+	Mo	PbF4	SrOH+
BFCl2	CHF2Cl	C12H10_biphenyl	HBO2	Mo+	PbI	Sr_OH_2
BFOH	CHF3	Ca	HBS	Mo-	PbI2	SrS
BF_OH_2	CHI3	Ca+	HBS+	MoO	PbI3	Sr2
BF2	CH2	CaBr	HCN	MoO2	PbI4	Ta
BF2+	CH2Br2	CaBr2	HCO	MoO3	PbO	Ta+
BF2-	CH2Cl	CaCl	HCO+	MoO3-	PbO2	Ta-
BF2Cl	CH2ClBr	CaCl+	HCCN	Mo2O6	PbS	TaCl5
BF2OH	CH2Cl2	CaCl2	HCCO	Mo3O9	PbS2	TaO
BF3	CH2F	CaF	HCl	Mo4O12	Rb	TaO2
BF4-	CH2FBr	CaF+	HD	Mo5O15	Rb+	Ti
BH	CH2FCl	CaF2	HD+	N	Rb-	Ti+
BHCl	CH2F2	CaH	HDO	N+	RbBO2	Ti-
BHCl2	CH2I2	CaI	HDO2	N-	RbBr	TiCl
BHF	CH3	CaI2	HF	NCO	RbCl	TiCl2
BHFCl	CH3Br	CaO	HI	ND	RbF	TiCl3
BHF2	CH3Cl	CaO+	HNC	ND2	RbH	TiCl4
BH2	CH3F	CaOH	HNCO	ND3	RbI	TiO
BH2Cl	CH3I	CaOH+	HNO	NF	RbK	TiO+
BH2F	CH2OH	Ca_OH_2	HNO2	NF2	Rbli	TiOCl
BH3	CH2OH+	CaS	HNO3	NF3	RbNO2	TiOCl2
BH3NH3	CH3O	Ca2	HOCl	NH	RbNO3	TiO2
BH4	CH4	Cd	HOF	NH+	RbNa	U
BI	CH3OH	Cd+	HO2	NHF	RbO	UF
BI2	CH3OOH	Cl	HO2-	NHF2	RbOH	UF+
BI3	CI	Cl+	HPO	NH2	Rb2Br2	UF-
BN	CI2	Cl-	HSO3F	NH2F	Rb2Cl2	UF2
BO	CI3	ClCN	H2	NH3	Rb2F2	UF2+
BO-	CI4	ClF	H2+	NH2OH	Rb2I2	UF2-
BOCl	CN	ClF3	H2-	NH4+	Rb2O	UF3
BOCl2	CN+	ClF5	HBOH	NO	Rb2O2	UF3+
BOF	CN-	ClO	HCOOH	NOCl	Rb2O2H2	UF3-
BOF2	CNN	ClO2	H2F2	NOF	Rb2SO4	UF4
BOH	CO	Cl2	H2O	NOF3	Rn	UF4+
BO2	CO+	Cl2O	H2O+	NO2	Rn+	UF4-
BO2-	COCl	Co	H2O2	NO2-	S	UF5
B_OH_2	COCl2	Co+	H2S	NO2Cl	S+	UF5+

BS	COFCl	Co-	H2SO4	NO2F	S-	UF5-
BS2	COF2	Cr	H2BOH	NO3	SCl	UF6
B2	COHCl	Cr+	HB_OH_2	NO3-	SCl2	UF6-
B2C	COHF	Cr-	H3BO3	NO3F	SCl2+	UO
B2Cl4	COS	CrN	H3B3O3	N2	SD	UO+
B2F4	CO2	CrO	H3B3O6	N2+	SF	UOF
B2H	CO2+	CrO2	H3F3	N2-	SF+	UOF2
B2H2	COOH	CrO3	H3O+	NCN	SF-	UOF3
B2H3	CP	CrO3-	H4F4	N2D2_cis	SF2	UOF4
B2H3_db	CS	Cs	H5F5	N2F2	SF2+	UO2
B2H4	CS2	Cs+	H6F6	N2F4	SF2-	UO2+
B2H4_db	C2	Cs-	H7F7	N2H2	SF3	UO2-
B2H5	C2+	CsBO2	He	NH2NO2	SF3+	UO2F
B2H5_db	C2-	CsBr	He+	N2H4	SF3-	UO2F2
B2H6	C2Cl	CsCl	Hg	N2O	SF4	UO3
B2O	C2Cl2	CsF	Hg+	N2O+	SF4+	UO3-
B2O2	C2Cl3	CsH	HgBr2	N2O3	SF4-	V
B2O3	C2Cl4	CsI	I	N2O4	SF5	V+
B2_OH_4	C2Cl6	Csli	I+	N2O5	SF5+	V-
B2S	C2F	CsNO2	I-	N3	SF5-	VCl4
B2S2	C2FCl	CsNO3	IF5	N3H	SF6	VN
B2S3	C2FCl3	CsNa	IF7	Na	SF6-	VO
B3H7_C2v	C2F2	CsO	I2	Na+	SH	VO2
B3H7_Cs	C2F2Cl2	CsOH	In	Na-	SH-	V4O10
B3H9	C2F3	CsRb	In+	NaAlF4	SN	W
B3N3H6	C2F3Cl	Cs2	InBr	NaBO2	SO	W+
B3O3Cl3	C2F4	Cs2Br2	InBr2	NaBr	SO-	W-
B3O3FCl2	C2F6	Cs2CO3	InBr3	NaCN	SOF2	WCl6
B3O3F2Cl	C2H	Cs2Cl2	InCl	NaCl	SO2	WO
B3O3F3	C2HCl	Cs2F2	InCl2	NaF	SO2-	WOCl4
B4H4	C2HCl3	Cs2I2	InCl3	NaH	SO2Cl2	WO2
B4H10	C2HF	Cs2O	InF	NaI	SO2FCl	WO2Cl2
B4H12	C2HFCl2	Cs2O+	InF2	Nali	SO2F2	WO3
B5H9	C2HF2Cl	Cs2O2	InF3	NaNO2	SO3	WO3-
Ba	C2HF3	Cs2O2H2	InH	NaNO3	S2	Xe
Ba+	C2H2_vinylidene	Cs2SO4	InI	NaO	S2-	Xe+
BaBr	C2H2Cl2	Cu	InI2	NaOH	S2Cl2	Zn
BaBr2	C2H2FCl	Cu+	InI3	NaOH+	S2F2	Zn+
BaCl	C2H2F2	Cu-	InO	Na2	S2O	Zr
BaCl+	CH2CO_ketene	CuCl	InOH	Na2Br2	S3	Zr+
BaCl2	O_CH_2O	CuF	In2Br2	Na2Cl2	S4	Zr-
BaF	HO_CO_2OH	CuF2	In2Br4	Na2F2	S5	ZrN
BaF+	C2H3_vinyl	CuO	In2Br6	Na2I2	S6	ZrO
BaF2	CH2Br-COOH	Cu2	In2Cl2	Na2O	S7	ZrO+
BaH	C2H3Cl	Cu3Cl3	In2Cl4	Na2O+	S8	ZrO2
BaI	CH2Cl-COOH	D	In2Cl6	Na2O2	Sc	
BaI2	C2H3F	D+	In2F2	Na2O2H2	Sc+	
BaO	CH3CN	D-	In2F4	Na2SO4	Sc-	
BaO+	CH3CO_acetyl	DBr	In2F6	Na3Cl3	ScO	
BaOH	C2H4	DCl	In2I2	Na3F3	ScO+	

Package Content

Name	Description
Common	Common packages and data for the ideal gas models
SingleGases	Media models of ideal gases from NASA tables

MixtureGases	Medium models consisting of mixtures of ideal gases

Modelica.Media.IdealGases.Common

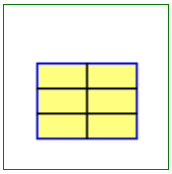
Common packages and data for the ideal gas models

Information

Package Content

Name	Description
DataRecord	Coefficient data record for properties of ideal gases based on NASA source
SingleGasNasa	Medium model of an ideal gas based on NASA source
MixtureGasNasa	Medium model of a mixture of ideal gases based on NASA source
FluidData	Critical data, dipole moments and related data
SingleGasesData	Ideal gas data based on the NASA Glenn coefficients

Modelica.Media.IdealGases.Common.DataRecord



Coefficient data record for properties of ideal gases based on NASA source

Information

This data record contains the coefficients for the ideal gas equations according to:

> McBride B.J., Zehe M.J., and Gordon S. (2002): **NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species**. NASA report TP-2002-211556

The equations have the following structure:

$$cp(T) = R\sum_{i=1}^{7} a_i T^{i-3}$$

$$h(T) = RT\left(-\frac{a_1}{T^2} + a_2\frac{\log(T)}{T} + \sum_{i=3}^{7} a_i\frac{T^{i-3}}{i-2} + \frac{b_1}{T}\right)$$

$$s_0(T) = R\left(-\frac{a_1}{2T^2} - \frac{a_2}{T} + a_3\log(T) + \sum_{i=4}^{7} a_i\frac{T^{i-3}}{i-3} + b_2\right)$$

$$s(T,p) = s_0(T) - R\ln\left(\frac{p}{p_0}\right)$$

The polynomials for h(T) and s0(T) are derived via integration from the one for cp(T) and contain the integration constants b1, b2 that define the reference specific enthalpy and entropy. For entropy differences the reference pressure p0 is arbitrary, but not for absolute entropies. It is chosen as 1 standard atmosphere (101325 Pa).

For most gases, the region of validity is from 200 K to 6000 K. The equations are splitted into two regions that are separated by Tlimit (usually 1000 K). In both regions the gas is described by the data above. The

two branches are continuous and in most gases also differentiable at Tlimit.

Modelica definition

```
record DataRecord
  "Coefficient data record for properties of ideal gases based on NASA source"
  extends Modelica.Icons.Record;
  String name "Name of ideal gas";
  SI.MolarMass MM "Molar mass";
  SI.SpecificEnthalpy Hf "Enthalpy of formation at 298.15K";
  SI.SpecificEnthalpy H0 "H0(298.15K) - H0(0K)";
  SI.Temperature Tlimit "Temperature limit between low and high data sets";
  Real alow[7] "Low temperature coefficients a";
  Real blow[2] "Low temperature constants b";
  Real ahigh[7] "High temperature coefficients a";
  Real bhigh[2] "High temperature constants b";
  SI.SpecificHeatCapacity R "Gas constant";
end DataRecord;
```

Modelica.Media.IdealGases.Common.SingleGasNasa

Medium model of an ideal gas based on NASA source

Information

This model calculates medium properties for an ideal gas of a single substance, or for an ideal gas consisting of several substances where the mass fractions are fixed. Independent variables are temperature **T** and pressure **p**. Only density is a function of T and p. All other quantities are solely a function of T. The properties are valid in the range:

$$200\ K \leq T \leq 6000\ K$$

The following quantities are always computed:

Variable	Unit	Description
h	J/kg	specific enthalpy h = h(T)
u	J/kg	specific internal energy u = u(T)
d	kg/m^3	density d = d(p,T)

For the other variables, see the functions in Modelica.Media.IdealGases.Common.SingleGasNasa. Note, dynamic viscosity and thermal conductivity are only provided for gases that use a data record from Modelica.Media.IdealGases.FluidData. Currently these are the following gases:

```
  Ar
  C2H2_vinylidene
  C2H4
  C2H5OH
  C2H6
  C3H6_propylene
  C3H7OH
  C3H8
  C4H8_1_butene
  C4H9OH
  C4H10_n_butane
  C5H10_1_pentene
  C5H12_n_pentane
```

```
C6H6
C6H12_1_hexene
C6H14_n_heptane
C7H14_1_heptene
C8H10_ethylbenz
CH3OH
CH4
CL2
CO
CO2
F2
H2
H2O
He
N2
N2O
NH3
NO
O2
SO2
SO3
```

Sources for model and literature:
Original Data: Computer program for calculation of complex chemical equilibrium compositions and applications. Part 1: Analysis Document ID: 19950013764 N (95N20180) File Series: NASA Technical Reports Report Number: NASA-RP-1311 E-8017 NAS 1.61:1311 Authors: Gordon, Sanford (NASA Lewis Research Center) Mcbride, Bonnie J. (NASA Lewis Research Center) Published: Oct 01, 1994.

Known limits of validity:
The data is valid for temperatures between 200K and 6000K. A few of the data sets for monatomic gases have a discontinuous 1st derivative at 1000K, but this never caused problems so far.

This model has been copied from the ThermoFluid library and adapted to the Modelica.Media package.

Package Content

Name	Description
ThermodynamicState	thermodynamic state variables for ideal gases
FluidConstants	Extended fluid constants
excludeEnthalpyOfFormation=true	If true, enthalpy of formation Hf is not included in specific enthalpy h
referenceChoice=Choices.ReferenceEnthalpy.ZeroAt0K	Choice of reference enthalpy
h_offset=0.0	User defined offset for reference enthalpy, if referenceChoice = UserDefined
data	Data record of ideal gas substance
fluidConstants	constant data for the fluid
BaseProperties	Base properties of ideal gas medium
setState_pTX	Return thermodynamic state as function of p, T and composition X
setState_phX	Return thermodynamic state as function of p, h and composition X
setState_psX	Return thermodynamic state as function of p, s and composition X

setState_dTX	Return thermodynamic state as function of d, T and composition X
pressure	return pressure of ideal gas
temperature	return temperature of ideal gas
density	return density of ideal gas
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificHeatCapacityCp	Return specific heat capacity at constant pressure
specificHeatCapacityCv	Compute specific heat capacity at constant volume from temperature and gas data
isentropicExponent	Return isentropic exponent
velocityOfSound	Return velocity of sound
isentropicEnthalpyApproximation	approximate method of calculating h_is from upstream properties and downstream pressure
isentropicEnthalpy	Return isentropic enthalpy
isobaricExpansionCoefficient	Returns overall the isobaric expansion coefficient beta
isothermalCompressibility	Returns overall the isothermal compressibility factor
density_derp_T	density derivative by temperature at constant pressure
density_derT_p	density derivative by temperature at constant pressure
density_derX	density derivative by mass fraction
cp_T	Compute specific heat capacity at constant pressure from temperature and gas data
cp_Tlow	Compute specific heat capacity at constant pressure, low T region
cp_Tlow_der	Compute specific heat capacity at constant pressure, low T region
h_T	Compute specific enthalpy from temperature and gas data; reference is decided by the refChoice input, or by the referenceChoice package constant by default
h_T_der	derivative function for h_T
h_Tlow	Compute specific enthalpy, low T region; reference is decided by the refChoice input, or by the referenceChoice package constant by default
h_Tlow_der	Compute specific enthalpy, low T region; reference is decided by the refChoice input, or by the referenceChoice package constant by default
s0_T	Compute specific entropy from temperature and gas data
s0_Tlow	Compute specific entropy, low T region
dynamicViscosityLowPressure	Dynamic viscosity of low pressure gases
dynamicViscosity	dynamic viscosity

thermalConductivityEstimate	Thermal conductivity of polyatomic gases(Eucken and Modified Eucken correlation)
thermalConductivity	thermal conductivity of gas
molarMass	return the molar mass of the medium
T_h	Compute temperature from specific enthalpy
T_ps	Compute temperature from pressure and specific entropy
Inherited	
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
density_ph	Return density from p and h
temperature_ph	Return temperature from p and h
pressure_dT	Return pressure from d and T
specificEnthalpy_dT	Return specific enthalpy from d and T
specificEnthalpy_ps	Return specific enthalpy from p and s
temperature_ps	Return temperature from p and s
density_ps	Return density from p and s
specificEnthalpy_pT	Return specific enthalpy from p and T
density_pT	Return density from p and T
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)

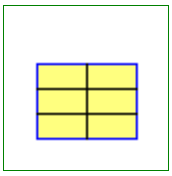
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property

ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
  constant Boolean excludeEnthalpyOfFormation=true
  "If true, enthalpy of formation Hf is not included in specific enthalpy h";

  constant ReferenceEnthalpy.Temp referenceChoice=Choices.
        ReferenceEnthalpy.ZeroAt0K "Choice of reference enthalpy";

  constant SpecificEnthalpy h_offset=0.0
  "User defined offset for reference enthalpy, if referenceChoice =
UserDefined";

  constant IdealGases.Common.DataRecord data
  "Data record of ideal gas substance";

  constant FluidConstants[nS] fluidConstants "constant data for the fluid";
```

Modelica.Media.IdealGases.Common.SingleGasNasa.ThermodynamicState

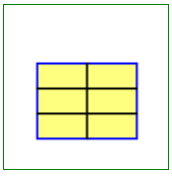


thermodynamic state variables for ideal gases

Modelica definition

```
redeclare replaceable record extends ThermodynamicState
  "thermodynamic state variables for ideal gases"
  AbsolutePressure p "Absolute pressure of medium";
  Temperature T "Temperature of medium";
end ThermodynamicState;
```

Modelica.Media.IdealGases.Common.SingleGasNasa.FluidConstants



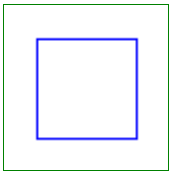
Extended fluid constants

Modelica definition

```
redeclare replaceable record extends FluidConstants
```

```
  "Extended fluid constants"
  Temperature criticalTemperature "critical temperature";
  AbsolutePressure criticalPressure "critical pressure";
  MolarVolume criticalMolarVolume "critical molar Volume";
  Real acentricFactor "Pitzer acentric factor";
  Temperature triplePointTemperature "triple point temperature";
  AbsolutePressure triplePointPressure "triple point pressure";
  Temperature meltingPoint "melting point at 101325 Pa";
  Temperature normalBoilingPoint "normal boiling point (at 101325 Pa)";
  DipoleMoment dipoleMoment
    "dipole moment of molecule in Debye (1 debye = 3.33564e10-30 C.m)";
  Boolean hasIdealGasHeatCapacity=false
    "true if ideal gas heat capacity is available";
  Boolean hasCriticalData=false "true if critical data are known";
  Boolean hasDipoleMoment=false "true if a dipole moment known";
  Boolean hasFundamentalEquation=false "true if a fundamental equation";
  Boolean hasLiquidHeatCapacity=false
    "true if liquid heat capacity is available";
  Boolean hasSolidHeatCapacity=false "true if solid heat capacity is available";
  Boolean hasAccurateViscosityData=false
    "true if accurate data for a viscosity function is available";
  Boolean hasAccurateConductivityData=false
    "true if accurate data for thermal conductivity is available";
  Boolean hasVapourPressureCurve=false
    "true if vapour pressure data, e.g. Antoine coefficents are known";
  Boolean hasAcentricFactor=false "true if Pitzer accentric factor is known";
  SpecificEnthalpy HCRIT0=0.0
    "Critical specific enthalpy of the fundamental equation";
  SpecificEntropy SCRIT0=0.0
    "Critical specific entropy of the fundamental equation";
  SpecificEnthalpy deltah=0.0
    "Difference between specific enthalpy model (h_m) and f.eq. (h_f) (h_m -
h_f)";
  SpecificEntropy deltas=0.0
    "Difference between specific enthalpy model (s_m) and f.eq. (s_f) (s_m -
s_f)";
end FluidConstants;
```

Modelica.Media.IdealGases.Common.SingleGasNasa.BaseProperties



Base properties of ideal gas medium

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

Modelica.Media.IdealGases.Common.SingleGasNasa.setState_pTX

Return thermodynamic state as function of p, T and composition X

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	

Modelica.Media.IdealGases.Common.SingleGasNasa.setState_phX

Return thermodynamic state as function of p, h and composition X

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	

Modelica.Media.IdealGases.Common.SingleGasNasa.setState_psX

Return thermodynamic state as function of p, s and composition X

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	

Modelica.Media.IdealGases.Common.SingleGasNasa.setState_dTX

Return thermodynamic state as function of d, T and composition X

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	

Modelica.Media.IdealGases.Common.SingleGasNasa.pressure

return pressure of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.IdealGases.Common.SingleGasNasa.temperature

return temperature of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.IdealGases.Common.SingleGasNasa.density

return density of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.IdealGases.Common.SingleGasNasa.specificEnthalpy

Return specific enthalpy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.IdealGases.Common.SingleGasNasa.specificInternalEnergy

Return specific internal energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	u	Specific internal energy [J/kg]

Modelica.Media.IdealGases.Common.SingleGasNasa.specificEntropy

Return specific entropy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.specificGibbsEnergy

Return specific Gibbs energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	g	Specific Gibbs energy [J/kg]

Modelica.Media.IdealGases.Common.SingleGasNasa.specificHelmholtzEnergy

Return specific Helmholtz energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	f	Specific Helmholtz energy [J/kg]

Modelica.Media.IdealGases.Common.SingleGasNasa.specificHeatCapacityCp

Return specific heat capacity at constant pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.specificHeatCapacityCv

Compute specific heat capacity at constant volume from temperature and gas data

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.isentropicExponent

Return isentropic exponent

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.IdealGases.Common.SingleGasNasa.velocityOfSound

Return velocity of sound

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.IdealGases.Common.SingleGasNasa.isentropicEnthalpyApproximation

approximate method of calculating h_is from upstream properties and downstream pressure

Inputs

Type	Name	Default	Description
Pressure	p2		downstream pressure [Pa]
ThermodynamicState	state		properties at upstream location
Boolean	exclEnthForm	excludeEnthalpyOfFormation	If true, enthalpy of formation Hf is not included in specific enthalpy h
Temp	refChoice	referenceChoice	Choice of reference enthalpy
SpecificEnthalpy	h_off	h_offset	User defined offset for reference enthalpy, if referenceChoice = UserDefined [J/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h_is	isentropic enthalpy [J/kg]

Modelica.Media.IdealGases.Common.SingleGasNasa.isentropicEnthalpy

Return isentropic enthalpy

Inputs

Type	Name	Default	Description
Boolean	exclEnthForm	excludeEnthalpyOfFormation	If true, enthalpy of formation Hf is not included in specific enthalpy h
Temp	refChoice	referenceChoice	Choice of reference enthalpy
SpecificEnthalpy	h_off	h_offset	User defined offset for reference enthalpy, if referenceChoice = UserDefined [J/kg]
AbsolutePressure	p_downstream		downstream pressure [Pa]
ThermodynamicState	refState		reference state for entropy

Outputs

Type	Name	Description
SpecificEnthalpy	h_is	Isentropic enthalpy [J/kg]

Modelica.Media.IdealGases.Common.SingleGasNasa.isobaricExpansionCoefficient

Returns overall the isobaric expansion coefficient beta

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsobaricExpansionCoefficient	beta	Isobaric expansion coefficient [1/K]

Modelica.Media.IdealGases.Common.SingleGasNasa.isothermalCompressibility

Returns overall the isothermal compressibility factor

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	Isothermal compressibility [1/Pa]

Modelica.Media.IdealGases.Common.SingleGasNasa.density_derp_T

density derivative by temperature at constant pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByPressure	ddpT	Density derivative wrt pressure [s2/m2]

Modelica.Media.IdealGases.Common.SingleGasNasa.density_derT_p

density derivative by temperature at constant pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByTemperature	ddTp	Density derivative wrt temperature [kg/(m3.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.density_derX

density derivative by mass fraction

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	dddX[nX]	Derivative of density wrt mass fraction [kg/m3]

Modelica.Media.IdealGases.Common.SingleGasNasa.cp_T

Compute specific heat capacity at constant pressure from temperature and gas data

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at temperature T [J/(kg.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.cp_Tlow

Compute specific heat capacity at constant pressure, low T region

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at temperature T [J/(kg.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.cp_Tlow_der

Compute specific heat capacity at constant pressure, low T region

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data
Temperature	T		Temperature [K]
Real	dT		Temperature derivative

Outputs

Type	Name	Description
Real	cp_der	Derivative of specific heat capacity

Modelica.Media.IdealGases.Common.SingleGasNasa.h_T

Compute specific enthalpy from temperature and gas data; reference is decided by the refChoice input, or by the referenceChoice package constant by default

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data

Temperature	T		Temperature [K]
Boolean	exclEnthForm	excludeEnthalpyOfFormation	If true, enthalpy of formation Hf is not included in specific enthalpy h
Temp	refChoice	referenceChoice	Choice of reference enthalpy
SpecificEnthalpy	h_off	h_offset	User defined offset for reference enthalpy, if referenceChoice = UserDefined [J/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy at temperature T [J/kg]

Modelica.Media.IdealGases.Common.SingleGasNasa.h_T_der

derivative function for h_T

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data
Temperature	T		Temperature [K]
Boolean	exclEnthForm	excludeEnthalpyOfFormation	If true, enthalpy of formation Hf is not included in specific enthalpy h
Temp	refChoice	referenceChoice	Choice of reference enthalpy
SpecificEnthalpy	h_off	h_offset	User defined offset for reference enthalpy, if referenceChoice = UserDefined [J/kg]
Real	dT		Temperature derivative

Outputs

Type	Name	Description
Real	h_der	Specific enthalpy at temperature T

Modelica.Media.IdealGases.Common.SingleGasNasa.h_Tlow

Compute specific enthalpy, low T region; reference is decided by the refChoice input, or by the referenceChoice package constant by default

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data
Temperature	T		Temperature [K]
Boolean	exclEnthForm	excludeEnthalpyOfFormation	If true, enthalpy of formation Hf is not included in specific enthalpy h
Temp	refChoice	referenceChoice	Choice of reference enthalpy
SpecificEnthalpy	h_off	h_offset	User defined offset for reference enthalpy, if referenceChoice = UserDefined [J/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy at temperature T [J/kg]

Modelica.Media.IdealGases.Common.SingleGasNasa.h_Tlow_der

Compute specific enthalpy, low T region; reference is decided by the refChoice input, or by the referenceChoice package constant by default

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data
Temperature	T		Temperature [K]
Boolean	exclEnthForm	excludeEnthalpyOfFormation	If true, enthalpy of formation Hf is not included in specific enthalpy h
Temp	refChoice	referenceChoice	Choice of reference enthalpy
SpecificEnthalpy	h_off	h_offset	User defined offset for reference enthalpy, if referenceChoice = UserDefined [J/kg]
Real	dT		Temperature derivative [K/s]

Outputs

Type	Name	Description
Real	h_der	Derivative of specific enthalpy at temperature T [J/(kg.s)]

Modelica.Media.IdealGases.Common.SingleGasNasa.s0_T

Compute specific entropy from temperature and gas data

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy at temperature T [J/(kg.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.s0_Tlow

Compute specific entropy, low T region

Inputs

Type	Name	Default	Description
DataRecord	data		Ideal gas data

Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy at temperature T [J/(kg.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.dynamicViscosityLowPressure

Dynamic viscosity of low pressure gases

Information

The used formula are based on the method of Chung et al (1984, 1988) referred to in ref [1] chapter 9. The formula 9-4.10 is the one being used. The Formula is given in non-SI units, the follwong onversion constants were used to transform the formula to SI units:

- **Const1_SI:** The factor 10^(-9.5) =10^(-2.5)*1e-7 where the factor 10^(-2.5) originates from the conversion of g/mol->kg/mol + cm^3/mol->m^3/mol and the factor 1e-7 is due to conversionfrom microPoise->Pa.s.
- **Const2_SI:** The factor 1/3.335641e-27 = 1e-3/3.335641e-30 where the factor 3.335641e-30 comes from debye->C.m and 1e-3 is due to conversion from cm^3/mol->m^3/mol

References:

[1] Bruce E. Poling, John E. Prausnitz, John P. O'Connell, "The Properties of Gases and Liquids" 5th Ed. Mc Graw Hill.

Author

T. Skoglund, Lund, Sweden, 2004-08-31

Inputs

Type	Name	Default	Description
Temp_K	T		Gas temperature [K]
Temp_K	Tc		Critical temperature of gas [K]
MolarMass	M		Molar mass of gas [kg/mol]
MolarVolume	Vc		Critical molar volume of gas [m3/mol]
Real	w		Acentric factor of gas
ElectricDipoleMomentOfMolecule	mu		Dipole moment of gas molecule [C.m]
Real	k	0.0	Special correction for highly polar substances

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity of gas [Pa.s]

Modelica.Media.IdealGases.Common.SingleGasNasa.dynamicViscosity

dynamic viscosity

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.IdealGases.Common.SingleGasNasa.thermalConductivityEstimate

Thermal conductivity of polyatomic gases(Eucken and Modified Eucken correlation)

Information

This function provides two similar methods for estimating the thermal conductivity of polyatomic gases. The Eucken method (input method == 1) gives good results for low temperatures, but it tends to give an underestimated value of the thermal conductivity (lambda) at higher temperatures.
The Modified Eucken method (input method == 2) gives good results for high-temperatures, but it tends to give an overestimated value of the thermal conductivity (lambda) at low temperatures.

Inputs

Type	Name	Default	Description
SpecificHeatCapacity	Cp		Constant pressure heat capacity [J/(kg.K)]
DynamicViscosity	eta		Dynamic viscosity [Pa.s]
Integer	method	1	1: Eucken Method, 2: Modified Eucken Method

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.k)] [W/(m.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.thermalConductivity

thermal conductivity of gas

Inputs

Type	Name	Default	Description
Integer	method	1	1: Eucken Method, 2: Modified Eucken Method
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.IdealGases.Common.SingleGasNasa.molarMass

return the molar mass of the medium

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
MolarMass	MM	Mixture molar mass [kg/mol]

Modelica.Media.IdealGases.Common.SingleGasNasa.T_h

Compute temperature from specific enthalpy

Inputs

Type	Name	Default	Description
SpecificEnthalpy	h		Specific enthalpy [J/kg]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.IdealGases.Common.SingleGasNasa.T_ps

Compute temperature from pressure and specific entropy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.IdealGases.Common.MixtureGasNasa

Medium model of a mixture of ideal gases based on NASA source

Information

This model calculates the medium properties for single component ideal gases.

Sources for model and literature:

Original Data: Computer program for calculation of complex chemical equilibrium compositions and applications. Part 1: Analysis Document ID: 19950013764 N (95N20180) File Series: NASA Technical Reports Report Number: NASA-RP-1311 E-8017 NAS 1.61:1311 Authors: Gordon, Sanford (NASA Lewis Research Center) Mcbride, Bonnie J. (NASA Lewis Research Center) Published: Oct 01, 1994.

Known limits of validity:
The data is valid for temperatures between 200 K and 6000 K. A few of the data sets for monatomic gases have a discontinuous 1st derivative at 1000 K, but this never caused problems so far.

This model has been copied from the ThermoFluid library. It has been developed by Hubertus Tummescheit.

Package Content

Name	Description
data	Data records of ideal gas substances
excludeEnthalpyOfFormation=true	If true, enthalpy of formation Hf is not included in specific enthalpy h
referenceChoice=Choices.ReferenceEnthalpy.ZeroAt0K	Choice of reference enthalpy
h_offset=0.0	User defined offset for reference enthalpy, if referenceChoice = UserDefined
MMX=data[:].MM	molar masses of components
BaseProperties	
setState_pTX	Return thermodynamic state as function of p, T and composition X
setState_phX	Return thermodynamic state as function of p, h and composition X
setState_psX	Return thermodynamic state as function of p, s and composition X
setState_dTX	Return thermodynamic state as function of d, T and composition X
pressure	Return pressure of ideal gas
temperature	Return temperature of ideal gas
density	Return density of ideal gas
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificEntropy	Return specific entropy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
h_TX	Return specific enthalpy
h_TX_der	Return specific enthalpy derivative
gasConstant	Return gasConstant
specificHeatCapacityCp	Return specific heat capacity at constant pressure
specificHeatCapacityCv	Return specific heat capacity at constant volume from temperature and gas data
MixEntropy	Return mixing entropy of ideal gases / R
s_TX	Return temperature dependent part of the entropy, expects full entropy vector

isentropicExponent	Return isentropic exponent
velocityOfSound	Return velocity of sound
isentropicEnthalpyApproximation	Approximate method of calculating h_is from upstream properties and downstream pressure
isentropicEnthalpy	Return isentropic enthalpy
gasMixtureViscosity	Return viscosities of gas mixtures at low pressures (Wilke method)
dynamicViscosity	Return mixture dynamic viscosity
mixtureViscosityChung	Return the viscosity of gas mixtures without access to component viscosities (Chung, et. al. rules)
lowPressureThermalConductivity	Return thermal conductivites of low-pressure gas mixtures (Mason and Saxena Modification)
thermalConductivity	Return thermal conductivity for low pressure gas mixtures
isobaricExpansionCoefficient	Return isobaric expansion coefficient beta
isothermalCompressibility	Return isothermal compressibility factor
density_derp_T	Return density derivative by temperature at constant pressure
density_derT_p	Return density derivative by temperature at constant pressure
density_derX	Return density derivative by mass fraction
molarMass	Return molar mass of mixture
T_hX	Return temperature from specific enthalpy and mass fraction
T_psX	Return temperature from pressure, specific entropy and mass fraction
Inherited	
ThermodynamicState	thermodynamic state variables
FluidConstants	extended fluid constants
fluidConstants	constant data for the fluid
moleToMassFractions	Return mass fractions X from mole fractions
massToMoleFractions	Return mole fractions from mass fractions X
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)

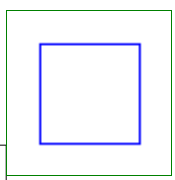
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific

	attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
  constant Modelica.Media.IdealGases.Common.DataRecord[:] data
  "Data records of ideal gas substances";

  constant Boolean excludeEnthalpyOfFormation=true
  "If true, enthalpy of formation Hf is not included in specific enthalpy h";

  constant Choices.ReferenceEnthalpy.Temp referenceChoice=Choices.
      ReferenceEnthalpy.ZeroAt0K "Choice of reference enthalpy";

  constant SpecificEnthalpy h_offset=0.0
  "User defined offset for reference enthalpy, if referenceChoice =
UserDefined";

  constant MolarMass[nX] MMX=data[:].MM "molar masses of components";
```

Modelica.Media.IdealGases.Common.MixtureGasNasa.BaseProperties



Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)

Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

Modelica.Media.IdealGases.Common.MixtureGasNasa.setState_pTX

Return thermodynamic state as function of p, T and composition X

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	

Modelica.Media.IdealGases.Common.MixtureGasNasa.setState_phX

Return thermodynamic state as function of p, h and composition X

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	

Modelica.Media.IdealGases.Common.MixtureGasNasa.setState_psX

Return thermodynamic state as function of p, s and composition X

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description

ThermodynamicState	state	

Modelica.Media.IdealGases.Common.MixtureGasNasa.setState_dTX

Return thermodynamic state as function of d, T and composition X

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	

Modelica.Media.IdealGases.Common.MixtureGasNasa.pressure

Return pressure of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.IdealGases.Common.MixtureGasNasa.temperature

Return temperature of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.IdealGases.Common.MixtureGasNasa.density

Return density of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.IdealGases.Common.MixtureGasNasa.specificEnthalpy

Return specific enthalpy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.IdealGases.Common.MixtureGasNasa.specificInternalEnergy

Return specific internal energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	u	Specific internal energy [J/kg]

Modelica.Media.IdealGases.Common.MixtureGasNasa.specificEntropy

Return specific entropy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.IdealGases.Common.MixtureGasNasa.specificGibbsEnergy

Return specific Gibbs energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	g	Specific Gibbs energy [J/kg]

Modelica.Media.IdealGases.Common.MixtureGasNasa.specificHelmholtzEnergy

Return specific Helmholtz energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	f	Specific Helmholtz energy [J/kg]

Modelica.Media.IdealGases.Common.MixtureGasNasa.h_TX

Return specific enthalpy

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Independent Mass fractions of gas mixture [kg/kg]
Boolean	exclEnthForm	excludeEnthalpyOfFormation	If true, enthalpy of formation Hf is not included in specific enthalpy h
Temp	refChoice	referenceChoice	Choice of reference enthalpy
SpecificEnthalpy	h_off	h_offset	User defined offset for reference enthalpy, if referenceChoice = UserDefined [J/kg]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy at temperature T [J/kg]

Modelica.Media.IdealGases.Common.MixtureGasNasa.h_TX_der

Return specific enthalpy derivative

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]
MassFraction	X[nX]		Independent Mass fractions of gas mixture [kg/kg]
Boolean	exclEnthForm	excludeEnthalpyOfFormation	If true, enthalpy of formation Hf is not included in specific enthalpy h
Temp	refChoice	referenceChoice	Choice of reference enthalpy
SpecificEnthalpy	h_off	h_offset	User defined offset for reference enthalpy, if referenceChoice = UserDefined [J/kg]
Real	dT		Temperature derivative
Real	dX[nX]		independent mass fraction derivative

Outputs

Type	Name	Description
Real	h_der	Specific enthalpy at temperature T

Modelica.Media.IdealGases.Common.MixtureGasNasa.gasConstant

Return gasConstant

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state

Outputs

Type	Name	Description
SpecificHeatCapacity	R	mixture gas constant [J/(kg.K)]

Modelica.Media.IdealGases.Common.MixtureGasNasa.specificHeatCapacityCp

Return specific heat capacity at constant pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.IdealGases.Common.MixtureGasNasa.specificHeatCapacityCv

Return specific heat capacity at constant volume from temperature and gas data

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.IdealGases.Common.MixtureGasNasa.MixEntropy

Return mixing entropy of ideal gases / R

Inputs

Type	Name	Default	Description
MoleFraction	x[:]		mole fraction of mixture [1]

Outputs

Type	Name	Description
Real	smix	mixing entropy contribution, divided by gas constant

---

Modelica.Media.IdealGases.Common.MixtureGasNasa.s_TX

Return temperature dependent part of the entropy, expects full entropy vector

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]
MassFraction	X[nX]		mass fraction [kg/kg]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.IdealGases.Common.MixtureGasNasa.isentropicExponent

Return isentropic exponent

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.IdealGases.Common.MixtureGasNasa.velocityOfSound

Return velocity of sound

Inputs

Type	Name	Default	Description
ThermodynamicState	state		properties at upstream location

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.IdealGases.Common.MixtureGasNasa.isentropicEnthalpyApproximation

Approximate method of calculating h_is from upstream properties and downstream pressure

Inputs

Type	Name	Default	Description
AbsolutePressure	p2		downstream pressure [Pa]
ThermodynamicState	state		thermodynamic state at upstream location

Outputs

Type	Name	Description
SpecificEnthalpy	h_is	isentropic enthalpy [J/kg]

Modelica.Media.IdealGases.Common.MixtureGasNasa.isentropicEnthalpy

Return isentropic enthalpy

Inputs

Type	Name	Default	Description
Boolean	exact	false	flag wether exact or approximate version should be used
AbsolutePressure	p_downstream		downstream pressure [Pa]
ThermodynamicState	refState		reference state for entropy

Outputs

Type	Name	Description
SpecificEnthalpy	h_is	Isentropic enthalpy [J/kg]

Modelica.Media.IdealGases.Common.MixtureGasNasa.gasMixtureViscosity

Return viscosities of gas mixtures at low pressures (Wilke method)

Information

Simplification of the kinetic theory (Chapman and Enskog theory) approach neglecting the second-order effects.

This equation has been extensively tested (Amdur and Mason, 1958; Bromley and Wilke, 1951; Cheung, 1958; Dahler, 1959; Gandhi and Saxena, 1964; Ranz and Brodowsky, 1962; Saxena and Gambhir, 1963a; Strunk, et al., 1964; Vanderslice, et al. 1962; Wright and Gray, 1962). In most cases, only nonpolar mixtures were compared, and very good results obtained. For some systems containing hidrogen as one component, less satisfactory agreement was noted. Wilke's method predicted mixture viscosities that were larger than experimental for the H2-N2 system, but for H2-NH3, it underestimated the viscosities.
Gururaja, et al. (1967) found that this method also overpredicted in the H2-O2 case but was quite accurate for the H2-CO2 system.
Wilke's approximation has proved reliable even for polar-polar gas mixtures of aliphatic alcohols (Reid and Belenyessy, 1960). The principal reservation appears to lie in those cases where Mi>>Mj and etai>>etaj.

Inputs

Type	Name	Default	Description
MoleFraction	yi[:]		Mole fractions [mol/mol]
MolarMass	M[:]		Mole masses [kg/mol]
DynamicViscosity	eta[:]		Pure component viscosities [Pa.s]

Outputs

Type	Name	Description
DynamicViscosity	etam	Viscosity of the mixture [Pa.s]

Modelica.Media.IdealGases.Common.MixtureGasNasa.dynamicViscosity

Return mixture dynamic viscosity

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.IdealGases.Common.MixtureGasNasa.mixtureViscosityChung

Return the viscosity of gas mixtures without access to component viscosities (Chung, et. al. rules)

Information

Equation to estimate the viscosity of gas mixtures at low pressures.
It is a simplification of an extension of the rigorous kinetic theory of Chapman and Enskog to determine the viscosity of multicomponent mixtures, at low pressures and with a factor to correct for molecule shape and polarity.

The input argument Kappa is a special correction for highly polar substances such as alcohols and acids. Values of kappa for a few such materials:

Compound	Kappa	Compound	Kappa
Methanol	0.215	n-Pentanol	0.122
Ethanol	0.175	n-Hexanol	0.114
n-Propanol	0.143	n-Heptanol	0.109
i-Propanol	0.143	Acetic Acid	0.0916
n-Butanol	0.132	Water	0.076
i-Butanol	0.132		

Chung, et al. (1984) suggest that for other alcohols not shown in the table:

kappa = 0.0682 + 4.704*[(number of -OH groups)]/[molecular weight]

S.I. units relation for the debyes:

1 debye = 3.162e-25 (J.m^3)^(1/2)

References

[1] THE PROPERTIES OF GASES AND LIQUIDS, Fifth Edition,
Bruce E. Poling, John M. Prausnitz, John P. O'Connell.
[2] Chung, T.-H., M. Ajlan, L. L. Lee, and K. E. Starling: Ind. Eng. Chem. Res., 27: 671 (1988).
[3] Chung, T.-H., L. L. Lee, and K. E. Starling; Ing. Eng. Chem. Fundam., 23: 3 ()1984).

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]
Temperature	Tc[:]		Critical temperatures [K]
MolarVolume	Vcrit[:]		Critical volumes (m3/mol) [m3/mol]
Real	w[:]		Acentric factors
Real	mu[:]		Dipole moments (debyes)
MolarMass	MolecularWeights[:]		Molecular weights (kg/mol) [kg/mol]
MoleFraction	y[:]		Molar Fractions [mol/mol]
Real	kappa[:]	zeros(nX)	Association Factors

Outputs

Type	Name	Description
DynamicViscosity	etaMixture	Mixture viscosity (Pa.s) [Pa.s]

Modelica.Media.IdealGases.Common.MixtureGasNasa.lowPressureThermalConductivity

Return thermal conductivites of low-pressure gas mixtures (Mason and Saxena Modification)

Information

This function applies the Masson and Saxena modification of the Wassiljewa Equation for the thermal conductivity for gas mixtures of n elements at low pressure.

For nonpolar gas mixtures errors will generally be less than 3 to 4%. For mixtures of nonpolar-polar and polar-polar gases, errors greater than 5 to 8% may be expected. For mixtures in which the sizes and polarities of the constituent molecules are not greatly different, the thermal conductivity can be estimated satisfactorily by a mole fraction average of the pure component conductivities.

Inputs

Type	Name	Default	Description
MoleFraction	y[:]		Mole fraction of the components in the gass mixture [mol/mol]
Temperature	T		Temperature [K]
Temperature	Tc[:]		Critical temperatures [K]
AbsolutePressure	Pc[:]		Critical pressures [Pa]
MolarMass	M[:]		Molecular weights [kg/mol]
ThermalConductivity	lambda[:]		Thermal conductivities of the pure gases [W/(m.K)]

Outputs

Type	Name	Description
ThermalConductivity	lambdam	Thermal conductivity of the gas mixture [W/(m.K)]

Modelica.Media.IdealGases.Common.MixtureGasNasa.thermalConductivity

Return thermal conductivity for low pressure gas mixtures

Inputs

Type	Name	Default	Description
Integer	method	1	method to compute single component thermal conductivity
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.IdealGases.Common.MixtureGasNasa.isobaricExpansionCoefficient

Return isobaric expansion coefficient beta

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsobaricExpansionCoefficient	beta	Isobaric expansion coefficient [1/K]

Modelica.Media.IdealGases.Common.MixtureGasNasa.isothermalCompressibility

Return isothermal compressibility factor

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	Isothermal compressibility [1/Pa]

Modelica.Media.IdealGases.Common.MixtureGasNasa.density_derp_T

Return density derivative by temperature at constant pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByPressure	ddpT	Density derivative wrt pressure [s2/m2]

Modelica.Media.IdealGases.Common.MixtureGasNasa.density_derT_p

Return density derivative by temperature at constant pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByTemperature	ddTp	Density derivative wrt temperature [kg/(m3.K)]

Modelica.Media.IdealGases.Common.MixtureGasNasa.density_derX

Return density derivative by mass fraction

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	dddX[nX]	Derivative of density wrt mass fraction [kg/m3]

Modelica.Media.IdealGases.Common.MixtureGasNasa.molarMass

Return molar mass of mixture

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
MolarMass	MM	Mixture molar mass [kg/mol]

Modelica.Media.IdealGases.Common.MixtureGasNasa.T_hX

Return temperature from specific enthalpy and mass fraction

Inputs

Type	Name	Default	Description
SpecificEnthalpy	h		specific enthalpy [J/kg]
MassFraction	X[:]		mass fractions of composition [kg/kg]

Outputs

Type	Name	Description
Temperature	T	temperature [K]

Modelica.Media.IdealGases.Common.MixtureGasNasa.T_psX

Return temperature from pressure, specific entropy and mass fraction

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
MassFraction	X[:]		mass fractions of composition [kg/kg]

Outputs

Type	Name	Description
Temperature	T	temperature [K]

Modelica.Media.IdealGases.Common.FluidData

Critical data, dipole moments and related data

Information

This package contains FluidConstants data records for the following 37 gases (see also the description in Modelica.Media.IdealGases):

```
Argon              Methane          Methanol       Carbon Monoxide  Carbon
Dioxide
Acetylene          Ethylene         Ethanol        Ethane           Propylene
Propane            1-Propanol       1-Butene       N-Butane         1-Pentene
N-Pentane          Benzene          1-Hexene       N-Hexane         1-Heptane
N-Heptane          Ethylbenzene     N-Octane       Chlorine         Fluorine
Hydrogen           Steam            Helium         Ammonia          Nitric Oxide
Nitrogen Dioxide   Nitrogen         Nitrous        Oxide            Neon Oxygen
Sulfur Dioxide     Sulfur Trioxide
```

Package Content

Name	Description
N2	
O2	
CL2	
F2	
CO2	
CO	
H2	
H2O	
N2O	
NO	
NO2	
NH3	
SO2	

SO3	
Ar	
He	
Ne	
CH4	
C2H6	
C3H8	
C4H10_n_butane	
C5H12_n_pentane	
C6H14_n_hexane	
C7H16_n_heptane	
C2H4	
C3H6_propylene	
C4H8_1_butene	
C5H10_1_pentene	
C6H12_1_hexene	
C7H14_1_heptene	
C2H2_vinylidene	
C6H6	
C8H18_n_octane	
C8H10_ethylbenz	
CH3OH	
C2H5OH	
C3H7OH	
C4H9OH	

Types and constants

```
  constant SingleGasNasa.FluidConstants N2(
                       chemicalFormula =         "N2",
                       iupacName =               "unknown",
                       structureFormula =        "unknown",
                       casRegistryNumber =       "7727-37-9",
                       meltingPoint =             63.15,
                       normalBoilingPoint =       77.35,
                       criticalTemperature =     126.20,
                       criticalPressure =         33.98e5,
                       criticalMolarVolume =      90.10e-6,
                       acentricFactor =            0.037,
                       dipoleMoment =              0.0,
                       molarMass =               SingleGasesData.N2.MM,
                       hasDipoleMoment =        true,
                       hasIdealGasHeatCapacity=true,
                       hasCriticalData =        true,
                       hasAcentricFactor =      true);

  constant SingleGasNasa.FluidConstants O2(
                       chemicalFormula =         "O2",
                       iupacName =               "unknown",
                       structureFormula =        "unknown",
                       casRegistryNumber =       "7782-44-7",
                       meltingPoint =             54.36,
```

```
                    normalBoilingPoint =      90.17,
                    criticalTemperature =    154.58,
                    criticalPressure =        50.43e5,
                    criticalMolarVolume =     73.37e-6,
                    acentricFactor =         0.022,
                    dipoleMoment =           0.0,
                    molarMass =              SingleGasesData.O2.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

constant SingleGasNasa.FluidConstants CL2(
                    chemicalFormula =        "CL2",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "7782-50-5",
                    meltingPoint =           172.19,
                    normalBoilingPoint =     239.12,
                    criticalTemperature =    417.00,
                    criticalPressure =        77.00e5,
                    criticalMolarVolume =    124.00e-6,
                    acentricFactor =         0.069,
                    dipoleMoment =           0.0,
                    molarMass =              SingleGasesData.CL2.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

constant SingleGasNasa.FluidConstants F2(
                    chemicalFormula =        "F2",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "7782-41-4",
                    meltingPoint =            53.48,
                    normalBoilingPoint =      84.95,
                    criticalTemperature =    144.30,
                    criticalPressure =        52.15e5,
                    criticalMolarVolume =     66.20e-6,
                    acentricFactor =           0.051,
                    dipoleMoment =             0.0,
                    molarMass =              SingleGasesData.F2.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

constant SingleGasNasa.FluidConstants CO2(
                    chemicalFormula =        "CO2",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "124-38-9",
                    meltingPoint =           216.58,
                    normalBoilingPoint =     -1.0,
                    criticalTemperature =    304.12,
                    criticalPressure =        73.74e5,
                    criticalMolarVolume =     94.07e-6,
```

```
                      acentricFactor =           0.225,
                      dipoleMoment =             0.0,
                      molarMass =               SingleGasesData.CO2.MM,
                      hasDipoleMoment =        true,
                      hasIdealGasHeatCapacity=true,
                      hasCriticalData =        true,
                      hasAcentricFactor =      true);

  constant SingleGasNasa.FluidConstants CO(
                      chemicalFormula =         "CO",
                      iupacName =               "unknown",
                      structureFormula =        "unknown",
                      casRegistryNumber =       "630-08-0",
                      meltingPoint =             68.15,
                      normalBoilingPoint =       81.66,
                      criticalTemperature =     132.85,
                      criticalPressure =         34.94e5,
                      criticalMolarVolume =      93.10e-6,
                      acentricFactor =           0.045,
                      dipoleMoment =             0.1,
                      molarMass =               SingleGasesData.CO.MM,
                      hasDipoleMoment =        true,
                      hasIdealGasHeatCapacity=true,
                      hasCriticalData =        true,
                      hasAcentricFactor =      true);

  constant SingleGasNasa.FluidConstants H2(
                      chemicalFormula =         "H2",
                      iupacName =               "unknown",
                      structureFormula =        "unknown",
                      casRegistryNumber =       "800000-51-5",
                      meltingPoint =             13.56,
                      normalBoilingPoint =       20.38,
                      criticalTemperature =      33.25,
                      criticalPressure =         12.97e5,
                      criticalMolarVolume =      65.00e-6,
                      acentricFactor =          -0.216,
                      dipoleMoment =             0.0,
                      molarMass =               SingleGasesData.H2.MM,
                      hasDipoleMoment =        true,
                      hasIdealGasHeatCapacity=true,
                      hasCriticalData =        true,
                      hasAcentricFactor =      true);

  constant SingleGasNasa.FluidConstants H2O(
                      chemicalFormula =         "H2O",
                      iupacName =               "unknown",
                      structureFormula =        "unknown",
                      casRegistryNumber =       "7732-18-5",
                      meltingPoint =            273.15,
                      normalBoilingPoint =      373.15,
                      criticalTemperature =     647.14,
                      criticalPressure =        220.64e5,
                      criticalMolarVolume =      55.95e-6,
                      acentricFactor =           0.344,
                      dipoleMoment =             1.8,
                      molarMass =               SingleGasesData.H2O.MM,
                      hasDipoleMoment =        true,
```

```
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants N2O(
                    chemicalFormula =        "N2O",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "10024-97-2",
                    meltingPoint =           182.33,
                    normalBoilingPoint =     184.67,
                    criticalTemperature =    309.60,
                    criticalPressure =        72.55e5,
                    criticalMolarVolume =     97.00e-6,
                    acentricFactor =         0.142,
                    dipoleMoment =           0.2,
                    molarMass =              SingleGasesData.N2O.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants NO(
                    chemicalFormula =        "NO",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "10102-43-9",
                    meltingPoint =           109.51,
                    normalBoilingPoint =     121.38,
                    criticalTemperature =    180.00,
                    criticalPressure =        64.80e5,
                    criticalMolarVolume =     58.00e-6,
                    acentricFactor =           0.582,
                    dipoleMoment =             0.2,
                    molarMass =              SingleGasesData.NO.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants NO2(
                    chemicalFormula =        "NO2",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "10102-44-0",
                    meltingPoint =           261.95,
                    normalBoilingPoint =     294.0,
                    criticalTemperature =    431.35,
                    criticalPressure =       101.33e5,
                    criticalMolarVolume =    82.5e-6,
                    acentricFactor =         0.849,
                    dipoleMoment =           0.32,
                    molarMass =              SingleGasesData.NO2.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);
```

```
  constant SingleGasNasa.FluidConstants NH3(
                       chemicalFormula =        "NH3",
                       iupacName =              "unknown",
                       structureFormula =       "unknown",
                       casRegistryNumber =      "7664-41-7",
                       meltingPoint =           195.41,
                       normalBoilingPoint =     239.82,
                       criticalTemperature =    405.40,
                       criticalPressure =       113.53e5,
                       criticalMolarVolume =     72.47e-6,
                       acentricFactor =           0.257,
                       dipoleMoment =             1.5,
                       molarMass =              SingleGasesData.NH3.MM,
                       hasDipoleMoment =       true,
                       hasIdealGasHeatCapacity=true,
                       hasCriticalData =       true,
                       hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants SO2(
                       chemicalFormula =        "SO2",
                       iupacName =              "unknown",
                       structureFormula =       "unknown",
                       casRegistryNumber =      "7446-09-5",
                       meltingPoint =           197.67,
                       normalBoilingPoint =     263.13,
                       criticalTemperature =    430.80,
                       criticalPressure =        78.84e5,
                       criticalMolarVolume =    122.00e-6,
                       acentricFactor =         0.245,
                       dipoleMoment =           1.6,
                       molarMass =              SingleGasesData.SO2.MM,
                       hasDipoleMoment =       true,
                       hasIdealGasHeatCapacity=true,
                       hasCriticalData =       true,
                       hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants SO3(
                       chemicalFormula =        "SO3",
                       iupacName =              "unknown",
                       structureFormula =       "unknown",
                       casRegistryNumber =      "7446-11-9",
                       meltingPoint =           289.95,
                       normalBoilingPoint =     317.90,
                       criticalTemperature =    490.90,
                       criticalPressure =        82.10e5,
                       criticalMolarVolume =    126.50e-6,
                       acentricFactor =         0.422,
                       dipoleMoment =           0.0,
                       molarMass =              SingleGasesData.SO3.MM,
                       hasDipoleMoment =       true,
                       hasIdealGasHeatCapacity=true,
                       hasCriticalData =       true,
                       hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants Ar(
                       chemicalFormula =        "Ar",
                       iupacName =              "unknown",
                       structureFormula =       "unknown",
```

```
                    casRegistryNumber =      "7440-37-1",
                    meltingPoint =            83.80,
                    normalBoilingPoint =      87.27,
                    criticalTemperature =    150.86,
                    criticalPressure =        48.98e5,
                    criticalMolarVolume =     74.57e-6,
                    acentricFactor =          -0.002,
                    dipoleMoment =             0.0,
                    molarMass =              SingleGasesData.Ar.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

 constant SingleGasNasa.FluidConstants He(
                    chemicalFormula =        "He",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "7440-59-7",
                    meltingPoint =             2.15,
                    normalBoilingPoint =       4.30,
                    criticalTemperature =      5.19,
                    criticalPressure =         2.27e5,
                    criticalMolarVolume =     57.30e-6,
                    acentricFactor =          -0.390,
                    dipoleMoment =             0.0,
                    molarMass =              SingleGasesData.He.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

constant SingleGasNasa.FluidConstants Ne(
                    chemicalFormula =        "Ne",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "7440-01-9",
                    meltingPoint =            24.56,
                    normalBoilingPoint =      27.07,
                    criticalTemperature =     44.40,
                    criticalPressure =        27.60e5,
                    criticalMolarVolume =     41.70e-6,
                    acentricFactor =          -0.016,
                    dipoleMoment =             0.0,
                    molarMass =              SingleGasesData.Ne.MM,
                    hasDipoleMoment =       true,
                    hasIdealGasHeatCapacity=true,
                    hasCriticalData =       true,
                    hasAcentricFactor =     true);

 constant SingleGasNasa.FluidConstants CH4(
                    chemicalFormula =        "CH4",
                    iupacName =              "unknown",
                    structureFormula =       "unknown",
                    casRegistryNumber =      "74-82-8",
                    meltingPoint =            90.69,
                    normalBoilingPoint =     111.66,
                    criticalTemperature =    190.56,
```

```
                      criticalPressure =        45.99e5,
                      criticalMolarVolume =     98.60e-6,
                      acentricFactor =           0.011,
                      dipoleMoment =             0.0,
                      molarMass =               SingleGasesData.CH4.MM,
                      hasDipoleMoment =        true,
                      hasIdealGasHeatCapacity=true,
                      hasCriticalData =        true,
                      hasAcentricFactor =      true);

  constant SingleGasNasa.FluidConstants C2H6(
                      chemicalFormula =         "C2H6",
                      iupacName =               "unknown",
                      structureFormula =        "unknown",
                      casRegistryNumber =       "74-84-0",
                      meltingPoint =             90.35,
                      normalBoilingPoint =      184.55,
                      criticalTemperature =     305.32,
                      criticalPressure =         48.72e5,
                      criticalMolarVolume =     145.50e-6,
                      acentricFactor =           0.099,
                      dipoleMoment =             0.0,
                      molarMass =               SingleGasesData.C2H6.MM,
                      hasDipoleMoment =        true,
                      hasIdealGasHeatCapacity=true,
                      hasCriticalData =        true,
                      hasAcentricFactor =      true);

  constant SingleGasNasa.FluidConstants C3H8(
                      chemicalFormula =         "C3H8",
                      iupacName =               "unknown",
                      structureFormula =        "unknown",
                      casRegistryNumber =       "74-98-6",
                      meltingPoint =             91.45,
                      normalBoilingPoint =      231.02,
                      criticalTemperature =     369.83,
                      criticalPressure =         42.48e5,
                      criticalMolarVolume =     200.00e-6,
                      acentricFactor =           0.152,
                      dipoleMoment =             0.0,
                      molarMass =               SingleGasesData.C3H8.MM,
                      hasDipoleMoment =        true,
                      hasIdealGasHeatCapacity=true,
                      hasCriticalData =        true,
                      hasAcentricFactor =      true);

  constant SingleGasNasa.FluidConstants C4H10_n_butane(
                      chemicalFormula =         "C4H10",
                      iupacName =               "unknown",
                      structureFormula =        "unknown",
                      casRegistryNumber =       "106-97-8",
                      meltingPoint =            134.79,
                      normalBoilingPoint =      272.66,
                      criticalTemperature =     425.12,
                      criticalPressure =         37.96e5,
                      criticalMolarVolume =     255.00e-6,
                      acentricFactor =           0.20,
                      dipoleMoment =             0.0,
```

```
                        molarMass =
SingleGasesData.C4H10_n_butane.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C5H12_n_pentane(
                        chemicalFormula =        "C5H12",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "109-66-0",
                        meltingPoint =           143.43,
                        normalBoilingPoint =     309.22,
                        criticalTemperature =    469.70,
                        criticalPressure =        33.70e5,
                        criticalMolarVolume =    311.00e-6,
                        acentricFactor =           0.252,
                        dipoleMoment =             0.0,
                        molarMass =
SingleGasesData.C5H12_n_pentane.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C6H14_n_hexane(
                        chemicalFormula =        "C6H14",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "110-54-3",
                        meltingPoint =           177.84,
                        normalBoilingPoint =     341.88,
                        criticalTemperature =    507.60,
                        criticalPressure =        30.25e5,
                        criticalMolarVolume =    368.00e-6,
                        acentricFactor =           0.300,
                        dipoleMoment =             0.0,
                        molarMass =
SingleGasesData.C6H14_n_hexane.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C7H16_n_heptane(
                        chemicalFormula =        "C7H16",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "142-82-5",
                        meltingPoint =           182.59,
                        normalBoilingPoint =     371.57,
                        criticalTemperature =    540.20,
                        criticalPressure =        27.40e5,
                        criticalMolarVolume =    428.00e-6,
                        acentricFactor =           0.350,
                        dipoleMoment =             0.0,
                        molarMass =
```

```
SingleGasesData.C7H16_n_heptane.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C2H4(
                        chemicalFormula =        "C2H4",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "74-85-1",
                        meltingPoint =           103.99,
                        normalBoilingPoint =     169.42,
                        criticalTemperature =    282.34,
                        criticalPressure =        50.41e5,
                        criticalMolarVolume =    131.10e-6,
                        acentricFactor =           0.087,
                        dipoleMoment =             0.0,
                        molarMass =              SingleGasesData.C2H4.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C3H6_propylene(
                        chemicalFormula =        "C3H6",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "115-07-1",
                        meltingPoint =            87.89,
                        normalBoilingPoint =     225.46,
                        criticalTemperature =    364.90,
                        criticalPressure =        46.00e5,
                        criticalMolarVolume =    184.60e-6,
                        acentricFactor =           0.142,
                        dipoleMoment =             0.4,
                        molarMass =
SingleGasesData.C3H6_propylene.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C4H8_1_butene(
                        chemicalFormula =        "C4H8",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "106-98-9",
                        meltingPoint =            87.79,
                        normalBoilingPoint =     266.92,
                        criticalTemperature =    419.50,
                        criticalPressure =        40.20e5,
                        criticalMolarVolume =    240.80e-6,
                        acentricFactor =           0.194,
                        dipoleMoment =             0.3,
                        molarMass =
SingleGasesData.C4H8_1_butene.MM,
                        hasDipoleMoment =       true,
```

```
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C5H10_1_pentene(
                        chemicalFormula =        "C5H10",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "109-67-1",
                        meltingPoint =           106.95,
                        normalBoilingPoint =     303.11,
                        criticalTemperature =    464.80,
                        criticalPressure =        35.60e5,
                        criticalMolarVolume =    298.40e-6,
                        acentricFactor =           0.237,
                        dipoleMoment =             0.4,
                        molarMass =
SingleGasesData.C5H10_1_pentene.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C6H12_1_hexene(
                        chemicalFormula =        "C6H12",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "592-41-6",
                        meltingPoint =           133.34,
                        normalBoilingPoint =     336.63,
                        criticalTemperature =    504.00,
                        criticalPressure =        31.43e5,
                        criticalMolarVolume =    355.10e-6,
                        acentricFactor =           0.281,
                        dipoleMoment =             0.4,
                        molarMass =
SingleGasesData.C6H12_1_hexene.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C7H14_1_heptene(
                        chemicalFormula =        "C7H14",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "592-76-7",
                        meltingPoint =           153.45,
                        normalBoilingPoint =     366.79,
                        criticalTemperature =    537.30,
                        criticalPressure =        29.20e5,
                        criticalMolarVolume =    409.00e-6,
                        acentricFactor =           0.343,
                        dipoleMoment =             0.3,
                        molarMass =
SingleGasesData.C7H14_1_heptene.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
```

```
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C2H2_vinylidene(
                        chemicalFormula =        "C2H2",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "74-86-2",
                        meltingPoint =           192.35,
                        normalBoilingPoint =     188.40,
                        criticalTemperature =    308.30,
                        criticalPressure =        61.14e5,
                        criticalMolarVolume =    112.20e-6,
                        acentricFactor =           0.189,
                        dipoleMoment =             0.0,
                        molarMass =
SingleGasesData.C2H2_vinylidene.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C6H6(
                        chemicalFormula =        "C6H6",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "71-43-2",
                        meltingPoint =           278.68,
                        normalBoilingPoint =     353.24,
                        criticalTemperature =    562.05,
                        criticalPressure =        48.95e5,
                        criticalMolarVolume =    256.00e-6,
                        acentricFactor =           0.210,
                        dipoleMoment =             0.0,
                        molarMass =              SingleGasesData.C6H6.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C8H18_n_octane(
                        chemicalFormula =        "C8H18",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "111-65-9",
                        meltingPoint =           216.39,
                        normalBoilingPoint =     398.82,
                        criticalTemperature =    568.70,
                        criticalPressure =        24.90e5,
                        criticalMolarVolume =    492.00e-6,
                        acentricFactor =           0.399,
                        dipoleMoment =             0.0,
                        molarMass =
SingleGasesData.C8H18_n_octane.MM,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);
```

```
  constant SingleGasNasa.FluidConstants C8H10_ethylbenz(
                       chemicalFormula =         "C8H10",
                       iupacName =               "unknown",
                       structureFormula =        "unknown",
                       casRegistryNumber =       "100-41-4",
                       meltingPoint =            178.18,
                       normalBoilingPoint =      409.36,
                       criticalTemperature =     617.15,
                       criticalPressure =         36.09e5,
                       criticalMolarVolume =     374.00e-6,
                       acentricFactor =            0.304,
                       dipoleMoment =              0.4,
                       molarMass =
SingleGasesData.C8H10_ethylbenz.MM,
                       hasDipoleMoment =       true,
                       hasIdealGasHeatCapacity=true,
                       hasCriticalData =       true,
                       hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants CH3OH(
                       chemicalFormula =         "CH3OH",
                       iupacName =               "unknown",
                       structureFormula =        "unknown",
                       casRegistryNumber =       "67-56-1",
                       meltingPoint =            175.49,
                       normalBoilingPoint =      337.69,
                       criticalTemperature =     512.64,
                       criticalPressure =         80.97e5,
                       criticalMolarVolume =     118.00e-6,
                       acentricFactor =            0.565,
                       dipoleMoment =              1.7,
                       molarMass =               SingleGasesData.CH3OH.MM,
                       hasDipoleMoment =       true,
                       hasIdealGasHeatCapacity=true,
                       hasCriticalData =       true,
                       hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C2H5OH(
                       chemicalFormula =         "C2H5OH",
                       iupacName =               "unknown",
                       structureFormula =        "unknown",
                       casRegistryNumber =       "64-17-5",
                       meltingPoint =            159.05,
                       normalBoilingPoint =      351.80,
                       criticalTemperature =     513.92,
                       criticalPressure =         61.48e5,
                       criticalMolarVolume =     167.00e-6,
                       acentricFactor =            0.649,
                       dipoleMoment =              1.7,
                       molarMass =               SingleGasesData.C2H5OH.MM,
                       hasDipoleMoment =       true,
                       hasIdealGasHeatCapacity=true,
                       hasCriticalData =       true,
                       hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C3H7OH(
                       chemicalFormula =         "C3H7OH",
                       iupacName =               "unknown",
```

```
                        structureFormula =       "unknown",
                        casRegistryNumber =      "71-23-8",
                        meltingPoint =           147.00,
                        normalBoilingPoint =     370.93,
                        criticalTemperature =    536.78,
                        criticalPressure =        51.75e5,
                        criticalMolarVolume =    219.00e-6,
                        acentricFactor =           0.629,
                        dipoleMoment =             1.7,
                        molarMass =              60.1e-3,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);

  constant SingleGasNasa.FluidConstants C4H9OH(
                        chemicalFormula =        "C4H9OH",
                        iupacName =              "unknown",
                        structureFormula =       "unknown",
                        casRegistryNumber =      "71-36-3",
                        meltingPoint =           183.35,
                        normalBoilingPoint =     390.88,
                        criticalTemperature =    563.05,
                        criticalPressure =        44.23e5,
                        criticalMolarVolume =    275.00e-6,
                        acentricFactor =           0.589,
                        dipoleMoment =             1.8,
                        molarMass =              74.12e-3,
                        hasDipoleMoment =       true,
                        hasIdealGasHeatCapacity=true,
                        hasCriticalData =       true,
                        hasAcentricFactor =     true);
```

Modelica.Media.IdealGases.Common.SingleGasesData

Ideal gas data based on the NASA Glenn coefficients

Information

This package contains ideal gas models for the 1241 ideal gases from

> McBride B.J., Zehe M.J., and Gordon S. (2002): **NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species**. NASA report TP-2002-211556

Ag	BaOH+	C2H4O_ethylen_o	DF	In2I4	Nb	ScO2
Ag+	Ba_OH_2	CH3CHO_ethanal	DOCl	In2I6	Nb+	Sc2O
Ag-	BaS	CH3COOH	DO2	In2O	Nb-	Sc2O2
Air	Ba2	OHCH2COOH	DO2-	K	NbCl5	Si
Al	Be	C2H5	D2	K+	NbO	Si+
Al+	Be+	C2H5Br	D2+	K-	NbOCl3	Si-
Al-	Be++	C2H6	D2-	KAlF4	NbO2	SiBr
AlBr	BeBr	CH3N2CH3	D2O	KBO2	Ne	SiBr2
AlBr2	BeBr2	C2H5OH	D2O2	KBr	Ne+	SiBr3
AlBr3	BeCl	CH3OCH3	D2S	KCN	Ni	SiBr4
AlC	BeCl2	CH3O2CH3	e-	KCl	Ni+	SiC
AlC2	BeF	CCN	F	KF	Ni-	SiC2

AlCl	BeF2	CNC	F+	KH	NiCl	SiCl
AlCl+	BeH	OCCN	F-	KI	NiCl2	SiCl2
AlCl2	BeH+	C2N2	FCN	Kli	NiO	SiCl3
AlCl3	BeH2	C2O	FCO	KNO2	NiS	SiCl4
AlF	BeI	C3	FO	KNO3	O	SiF
AlF+	BeI2	C3H3_1_propynl	FO2_FOO	KNa	O+	SiFCl
AlFCl	BeN	C3H3_2_propynl	FO2_OFO	KO	O-	SiF2
AlFCl2	BeO	C3H4_allene	F2	KOH	OD	SiF3
AlF2	BeOH	C3H4_propyne	F2O	K2	OD-	SiF4
AlF2-	BeOH+	C3H4_cyclo	F2O2	K2+	OH	SiH
AlF2Cl	Be_OH_2	C3H5_allyl	FS2F	K2Br2	OH+	SiH+
AlF3	BeS	C3H6_propylene	Fe	K2CO3	OH-	SiHBr3
AlF4-	Be2	C3H6_cyclo	Fe+	K2C2N2	O2	SiHCl
AlH	Be2Cl4	C3H6O_propylox	Fe_CO_5	K2Cl2	O2+	SiHCl3
AlHCl	Be2F4	C3H6O_acetone	FeCl	K2F2	O2-	SiHF
AlHCl2	Be2O	C3H6O_propanal	FeCl2	K2I2	O3	SiHF3
AlHF	Be2OF2	C3H7_n_propyl	FeCl3	K2O	P	SiHI3
AlHFCl	Be2O2	C3H7_i_propyl	FeO	K2O+	P+	SiH2
AlHF2	Be3O3	C3H8	Fe_OH_2	K2O2	P-	SiH2Br2
AlH2	Be4O4	C3H8O_1propanol	Fe2Cl4	K2O2H2	PCl	SiH2Cl2
AlH2Cl	Br	C3H8O_2propanol	Fe2Cl6	K2SO4	PCl2	SiH2F2
AlH2F	Br+	CNCOCN	Ga	Kr	PCl2-	SiH2I2
AlH3	Br-	C3O2	Ga+	Kr+	PCl3	SiH3
AlI	BrCl	C4	GaBr	li	PCl5	SiH3Br
AlI2	BrF	C4H2_butadiyne	GaBr2	li+	PF	SiH3Cl
AlI3	BrF3	C4H4_1_3-cyclo	GaBr3	li-	PF+	SiH3F
AlN	BrF5	C4H6_butadiene	GaCl	liAlF4	PF-	SiH3I
AlO	BrO	C4H6_1butyne	GaCl2	liBO2	PFCl	SiH4
AlO+	OBrO	C4H6_2butyne	GaCl3	liBr	PFCl-	SiI
AlO-	BrOO	C4H6_cyclo	GaF	liCl	PFCl2	SiI2
AlOCl	BrO3	C4H8_1_butene	GaF2	liF	PFCl4	SiN
AlOCl2	Br2	C4H8_cis2_buten	GaF3	liH	PF2	SiO
AlOF	BrBrO	C4H8_isobutene	GaH	liI	PF2-	SiO2
AlOF2	BrOBr	C4H8_cyclo	GaI	liN	PF2Cl	SiS
AlOF2-	C	C4H9_n_butyl	GaI2	liNO2	PF2Cl3	SiS2
AlOH	C+	C4H9_i_butyl	GaI3	liNO3	PF3	Si2
AlOHCl	C-	C4H9_s_butyl	GaO	liO	PF3Cl2	Si2C
AlOHCl2	CBr	C4H9_t_butyl	GaOH	liOF	PF4Cl	Si2F6
AlOHF	CBr2	C4H10_n_butane	Ga2Br2	liOH	PF5	Si2N
AlOHF2	CBr3	C4H10_isobutane	Ga2Br4	liON	PH	Si3
AlO2	CBr4	C4N2	Ga2Br6	li2	PH2	Sn
AlO2-	CCl	C5	Ga2Cl2	li2+	PH2-	Sn+
Al_OH_2	CCl2	C5H6_1_3cyclo	Ga2Cl4	li2Br2	PH3	Sn-
Al_OH_2Cl	CCl2Br2	C5H8_cyclo	Ga2Cl6	li2F2	PN	SnBr
Al_OH_2F	CCl3	C5H10_1_pentene	Ga2F2	li2I2	PO	SnBr2
Al_OH_3	CCl3Br	C5H10_cyclo	Ga2F4	li2O	PO-	SnBr3
AlS	CCl4	C5H11_pentyl	Ga2F6	li2O+	POCl3	SnBr4
AlS2	CF	C5H11_t_pentyl	Ga2I2	li2O2	POFCl2	SnCl
Al2	CF+	C5H12_n_pentane	Ga2I4	li2O2H2	POF2Cl	SnCl2
Al2Br6	CFBr3	C5H12_i_pentane	Ga2I6	li2SO4	POF3	SnCl3
Al2C2	CFCl	CH3C_CH3_2CH3	Ga2O	li3+	PO2	SnCl4
Al2Cl6	CFClBr2	C6D5_phenyl	Ge	li3Br3	PO2-	SnF
Al2F6	CFCl2	C6D6	Ge+	li3Cl3	PS	SnF2
Al2I6	CFCl2Br	C6H2	Ge-	li3F3	P2	SnF3
Al2O	CFCl3	C6H5_phenyl	GeBr	li3I3	P2O3	SnF4
Al2O+	CF2	C6H5O_phenoxy	GeBr2	Mg	P2O4	SnI
Al2O2	CF2+	C6H6	GeBr3	Mg+	P2O5	SnI2
Al2O2+	CF2Br2	C6H5OH_phenol	GeBr4	MgBr	P3	SnI3
Al2O3	CF2Cl	C6H10_cyclo	GeCl	MgBr2	P3O6	SnI4

Al2S	CF2ClBr	C6H12_1_hexene	GeCl2	MgCl	P4	SnO
Al2S2	CF2Cl2	C6H12_cyclo	GeCl3	MgCl+	P4O6	SnO2
Ar	CF3	C6H13_n_hexyl	GeCl4	MgCl2	P4O7	SnS
Ar+	CF3+	C6H14_n_hexane	GeF	MgF	P4O8	SnS2
B	CF3Br	C7H7_benzyl	GeF2	MgF+	P4O9	Sn2
B+	CF3Cl	C7H8	GeF3	MgF2	P4O10	Sr
B-	CF4	C7H8O_cresol_mx	GeF4	MgF2+	Pb	Sr+
BBr	CH+	C7H14_1_heptene	GeH4	MgH	Pb+	SrBr
BBr2	CHBr3	C7H15_n_heptyl	GeI	MgI	Pb-	SrBr2
BBr3	CHCl	C7H16_n_heptane	GeO	MgI2	PbBr	SrCl
BC	CHClBr2	C7H16_2_methylh	GeO2	MgN	PbBr2	SrCl+
BC2	CHCl2	C8H8_styrene	GeS	MgO	PbBr3	SrCl2
BCl	CHCl2Br	C8H10_ethylbenz	GeS2	MgOH	PbBr4	SrF
BCl+	CHCl3	C8H16_1_octene	Ge2	MgOH+	PbCl	SrF+
BClOH	CHF	C8H17_n_octyl	H	Mg_OH_2	PbCl2	SrF2
BCl_OH_2	CHFBr2	C8H18_n_octane	H+	MgS	PbCl3	SrH
BCl2	CHFCl	C8H18_isooctane	H-	Mg2	PbCl4	SrI
BCl2+	CHFClBr	C9H19_n_nonyl	HAlO	Mg2F4	PbF	SrI2
BCl2OH	CHFCl2	C10H8_naphthale	HAlO2	Mn	PbF2	SrO
BF	CHF2	C10H21_n_decyl	HBO	Mn+	PbF3	SrOH
BFCl	CHF2Br	C12H9_o_bipheny	HBO+	Mo	PbF4	SrOH+
BFCl2	CHF2Cl	C12H10_biphenyl	HBO2	Mo+	PbI	Sr_OH_2
BFOH	CHF3	Ca	HBS	Mo-	PbI2	SrS
BF_OH_2	CHI3	Ca+	HBS+	MoO	PbI3	Sr2
BF2	CH2	CaBr	HCN	MoO2	PbI4	Ta
BF2+	CH2Br2	CaBr2	HCO	MoO3	PbO	Ta+
BF2-	CH2Cl	CaCl	HCO+	MoO3-	PbO2	Ta-
BF2Cl	CH2ClBr	CaCl+	HCCN	Mo2O6	PbS	TaCl5
BF2OH	CH2Cl2	CaCl2	HCCO	Mo3O9	PbS2	TaO
BF3	CH2F	CaF	HCl	Mo4O12	Rb	TaO2
BF4-	CH2FBr	CaF+	HD	Mo5O15	Rb+	Ti
BH	CH2FCl	CaF2	HD+	N	Rb-	Ti+
BHCl	CH2F2	CaH	HDO	N+	RbBO2	Ti-
BHCl2	CH2I2	CaI	HDO2	N-	RbBr	TiCl
BHF	CH3	CaI2	HF	NCO	RbCl	TiCl2
BHFCl	CH3Br	CaO	HI	ND	RbF	TiCl3
BHF2	CH3Cl	CaO+	HNC	ND2	RbH	TiCl4
BH2	CH3F	CaOH	HNCO	ND3	RbI	TiO
BH2Cl	CH3I	CaOH+	HNO	NF	RbK	TiO+
BH2F	CH2OH	Ca_OH_2	HNO2	NF2	Rbli	TiOCl
BH3	CH2OH+	CaS	HNO3	NF3	RbNO2	TiOCl2
BH3NH3	CH3O	Ca2	HOCl	NH	RbNO3	TiO2
BH4	CH4	Cd	HOF	NH+	RbNa	U
BI	CH3OH	Cd+	HO2	NHF	RbO	UF
BI2	CH3OOH	Cl	HO2-	NHF2	RbOH	UF+
BI3	CI	Cl+	HPO	NH2	Rb2Br2	UF-
BN	CI2	Cl-	HSO3F	NH2F	Rb2Cl2	UF2
BO	CI3	ClCN	H2	NH3	Rb2F2	UF2+
BO-	CI4	ClF	H2+	NH2OH	Rb2I2	UF2-
BOCl	CN	ClF3	H2-	NH4+	Rb2O	UF3
BOCl2	CN+	ClF5	HBOH	NO	Rb2O2	UF3+
BOF	CN-	ClO	HCOOH	NOCl	Rb2O2H2	UF3-
BOF2	CNN	ClO2	H2F2	NOF	Rb2SO4	UF4
BOH	CO	Cl2	H2O	NOF3	Rn	UF4+
BO2	CO+	Cl2O	H2O+	NO2	Rn+	UF4-
BO2-	COCl	Co	H2O2	NO2-	S	UF5
B_OH_2	COCl2	Co+	H2S	NO2Cl	S+	UF5+
BS	COFCl	Co-	H2SO4	NO2F	S-	UF5-
BS2	COF2	Cr	H2BOH	NO3	SCl	UF6

B2	COHCl	Cr+	HB_OH_2	NO3-	SCl2	UF6-
B2C	COHF	Cr-	H3BO3	NO3F	SCl2+	UO
B2Cl4	COS	CrN	H3B3O3	N2	SD	UO+
B2F4	CO2	CrO	H3B3O6	N2+	SF	UOF
B2H	CO2+	CrO2	H3F3	N2-	SF+	UOF2
B2H2	COOH	CrO3	H3O+	NCN	SF-	UOF3
B2H3	CP	CrO3-	H4F4	N2D2_cis	SF2	UOF4
B2H3_db	CS	Cs	H5F5	N2F2	SF2+	UO2
B2H4	CS2	Cs+	H6F6	N2F4	SF2-	UO2+
B2H4_db	C2	Cs-	H7F7	N2H2	SF3	UO2-
B2H5	C2+	CsBO2	He	NH2NO2	SF3+	UO2F
B2H5_db	C2-	CsBr	He+	N2H4	SF3-	UO2F2
B2H6	C2Cl	CsCl	Hg	N2O	SF4	UO3
B2O	C2Cl2	CsF	Hg+	N2O+	SF4+	UO3-
B2O2	C2Cl3	CsH	HgBr2	N2O3	SF4-	V
B2O3	C2Cl4	CsI	I	N2O4	SF5	V+
B2_OH_4	C2Cl6	Csli	I+	N2O5	SF5+	V-
B2S	C2F	CsNO2	I-	N3	SF5-	VCl4
B2S2	C2FCl	CsNO3	IF5	N3H	SF6	VN
B2S3	C2FCl3	CsNa	IF7	Na	SF6-	VO
B3H7_C2v	C2F2	CsO	I2	Na+	SH	VO2
B3H7_Cs	C2F2Cl2	CsOH	In	Na-	SH-	V4O10
B3H9	C2F3	CsRb	In+	NaAlF4	SN	W
B3N3H6	C2F3Cl	Cs2	InBr	NaBO2	SO	W+
B3O3Cl3	C2F4	Cs2Br2	InBr2	NaBr	SO-	W-
B3O3FCl2	C2F6	Cs2CO3	InBr3	NaCN	SOF2	WCl6
B3O3F2Cl	C2H	Cs2Cl2	InCl	NaCl	SO2	WO
B3O3F3	C2HCl	Cs2F2	InCl2	NaF	SO2-	WOCl4
B4H4	C2HCl3	Cs2I2	InCl3	NaH	SO2Cl2	WO2
B4H10	C2HF	Cs2O	InF	NaI	SO2FCl	WO2Cl2
B4H12	C2HFCl2	Cs2O+	InF2	Nali	SO2F2	WO3
B5H9	C2HF2Cl	Cs2O2	InF3	NaNO2	SO3	WO3-
Ba	C2HF3	Cs2O2H2	InH	NaNO3	S2	Xe
Ba+	C2H2_vinylidene	Cs2SO4	InI	NaO	S2-	Xe+
BaBr	C2H2Cl2	Cu	InI2	NaOH	S2Cl2	Zn
BaBr2	C2H2FCl	Cu+	InI3	NaOH+	S2F2	Zn+
BaCl	C2H2F2	Cu-	InO	Na2	S2O	Zr
BaCl+	CH2CO_ketene	CuCl	InOH	Na2Br2	S3	Zr+
BaCl2	O_CH_2O	CuF	In2Br2	Na2Cl2	S4	Zr-
BaF	HO_CO_2OH	CuF2	In2Br4	Na2F2	S5	ZrN
BaF+	C2H3_vinyl	CuO	In2Br6	Na2I2	S6	ZrO
BaF2	CH2Br-COOH	Cu2	In2Cl2	Na2O	S7	ZrO+
BaH	C2H3Cl	Cu3Cl3	In2Cl4	Na2O+	S8	ZrO2
BaI	CH2Cl-COOH	D	In2Cl6	Na2O2	Sc	
BaI2	C2H3F	D+	In2F2	Na2O2H2	Sc+	
BaO	CH3CN	D-	In2F4	Na2SO4	Sc-	
BaO+	CH3CO_acetyl	DBr	In2F6	Na3Cl3	ScO	
BaOH	C2H4	DCl	In2I2	Na3F3	ScO+	

Modelica.Media.IdealGases.SingleGases

Media models of ideal gases from NASA tables

Information

This package contains medium models for the following 37 gases (see also the description in

Modelica.Media.IdealGases):

```
Argon              Methane          Methanol       Carbon Monoxide  Carbon
Dioxide
Acetylene          Ethylene         Ethanol        Ethane           Propylene
Propane            1-Propanol       1-Butene       N-Butane         1-Pentene
N-Pentane          Benzene          1-Hexene       N-Hexane         1-Heptane
N-Heptane          Ethylbenzene     N-Octane       Chlorine         Fluorine
Hydrogen           Steam            Helium         Ammonia          Nitric Oxide
Nitrogen Dioxide   Nitrogen         Nitrous        Oxide            Neon Oxygen
Sulfur Dioxide     Sulfur Trioxide
```

Package Content

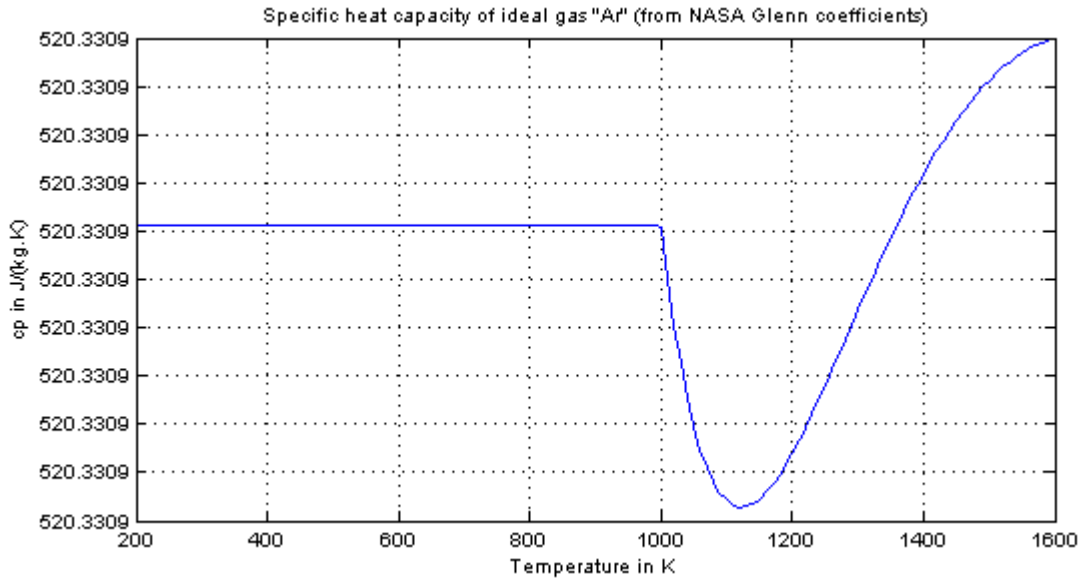
Name	Description
Ar	Ideal gas "Ar" from NASA Glenn coefficients
CH4	Ideal gas "CH4" from NASA Glenn coefficients
CH3OH	Ideal gas "CH3OH" from NASA Glenn coefficients
CO	Ideal gas "CO" from NASA Glenn coefficients
CO2	Ideal gas "CO2" from NASA Glenn coefficients
C2H2_vinylidene	Ideal gas "C2H2_vinylidene" from NASA Glenn coefficients
C2H4	Ideal gas "C2H4" from NASA Glenn coefficients
C2H5OH	Ideal gas "C2H5OH" from NASA Glenn coefficients
C2H6	Ideal gas "C2H6" from NASA Glenn coefficients
C3H6_propylene	Ideal gas "C3H6_propylene" from NASA Glenn coefficients
C3H8	Ideal gas "C3H8" from NASA Glenn coefficients
C3H8O_1propanol	Ideal gas "C3H8O_1propanol" from NASA Glenn coefficients
C4H8_1_butene	Ideal gas "C4H8_1_butene" from NASA Glenn coefficients
C4H10_n_butane	Ideal gas "C4H10_n_butane" from NASA Glenn coefficients
C5H10_1_pentene	Ideal gas "C5H10_1_pentene" from NASA Glenn coefficients
C5H12_n_pentane	Ideal gas "C5H12_n_pentane" from NASA Glenn coefficients
C6H6	Ideal gas "C6H6" from NASA Glenn coefficients
C6H12_1_hexene	Ideal gas "C6H12_1_hexene" from NASA Glenn coefficients
C6H14_n_hexane	Ideal gas "C6H14_n_hexane" from NASA Glenn coefficients
C7H14_1_heptene	Ideal gas "C7H14_1_heptene" from NASA Glenn coefficients
C7H16_n_heptane	Ideal gas "C7H16_n_heptane" from NASA Glenn coefficients
C8H10_ethylbenz	Ideal gas "C8H10_ethylbenz" from NASA Glenn coefficients
C8H18_n_octane	Ideal gas "C8H18_n_octane" from NASA Glenn coefficients
CL2	Ideal gas "Cl2" from NASA Glenn coefficients
F2	Ideal gas "F2" from NASA Glenn coefficients
H2	Ideal gas "H2" from NASA Glenn coefficients
H2O	Ideal gas "H2O" from NASA Glenn coefficients

He	Ideal gas "He" from NASA Glenn coefficients
NH3	Ideal gas "NH3" from NASA Glenn coefficients
NO	Ideal gas "NO" from NASA Glenn coefficients
NO2	Ideal gas "NO2" from NASA Glenn coefficients
N2	Ideal gas "N2" from NASA Glenn coefficients
N2O	Ideal gas "N2O" from NASA Glenn coefficients
Ne	Ideal gas "Ne" from NASA Glenn coefficients
O2	Ideal gas "O2" from NASA Glenn coefficients
SO2	Ideal gas "SO2" from NASA Glenn coefficients
SO3	Ideal gas "SO3" from NASA Glenn coefficients

Modelica.Media.IdealGases.SingleGases.Ar

Ideal gas "Ar" from NASA Glenn coefficients

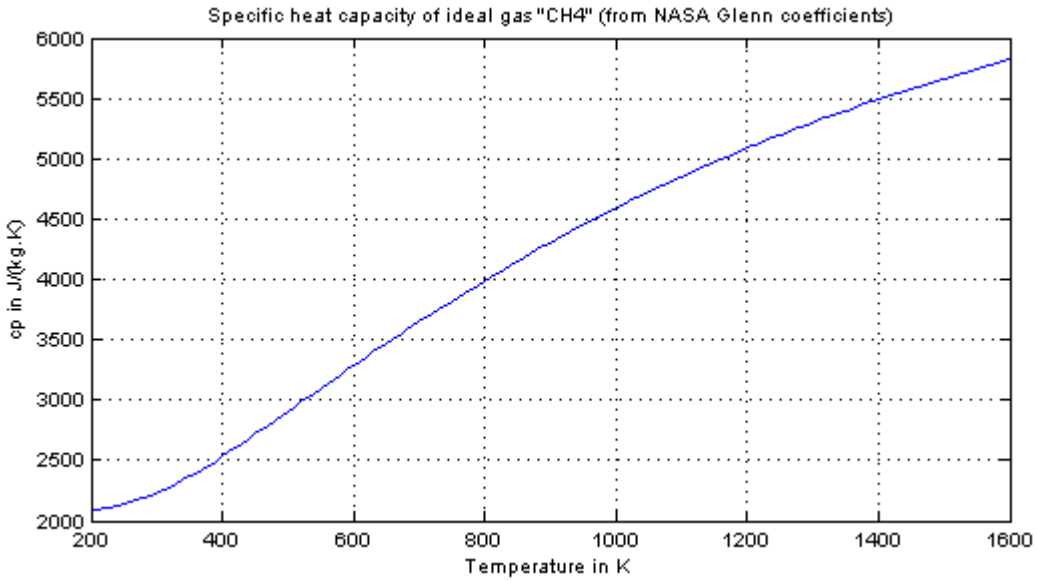
Information



Modelica.Media.IdealGases.SingleGases.CH4

Ideal gas "CH4" from NASA Glenn coefficients

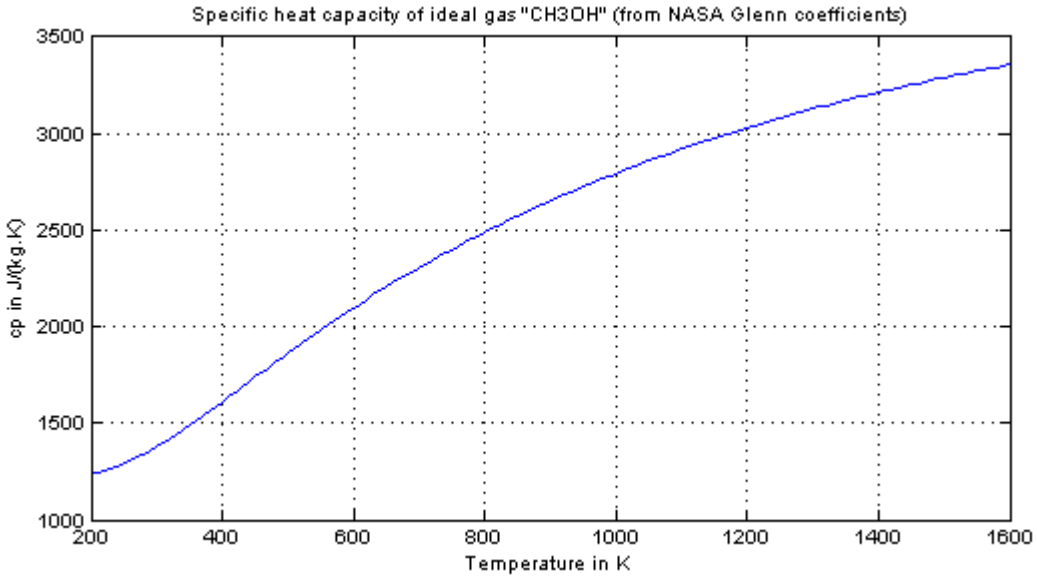
Information



Modelica.Media.IdealGases.SingleGases.CH3OH

Ideal gas "CH3OH" from NASA Glenn coefficients

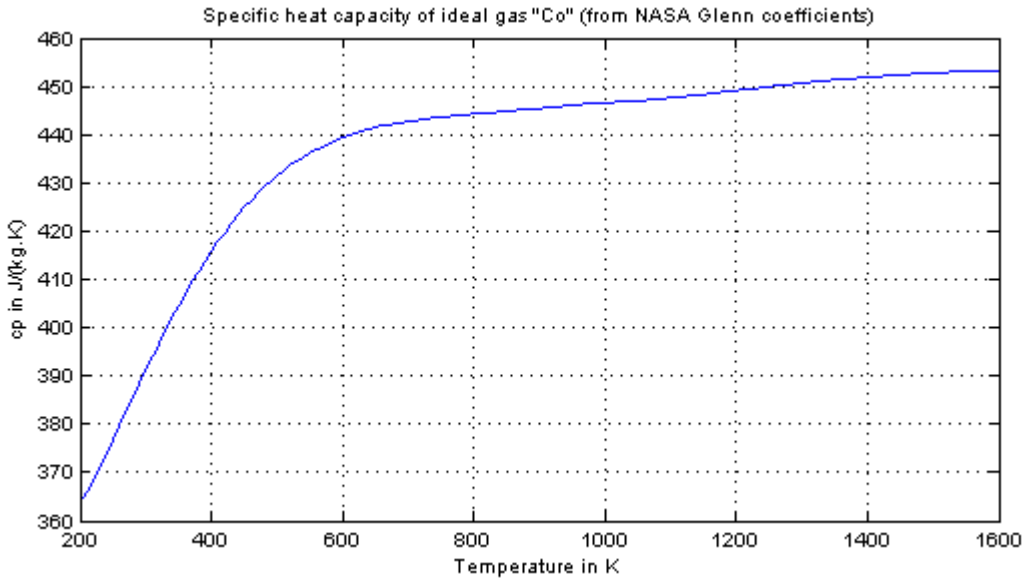
Information



Modelica.Media.IdealGases.SingleGases.CO

Ideal gas "CO" from NASA Glenn coefficients

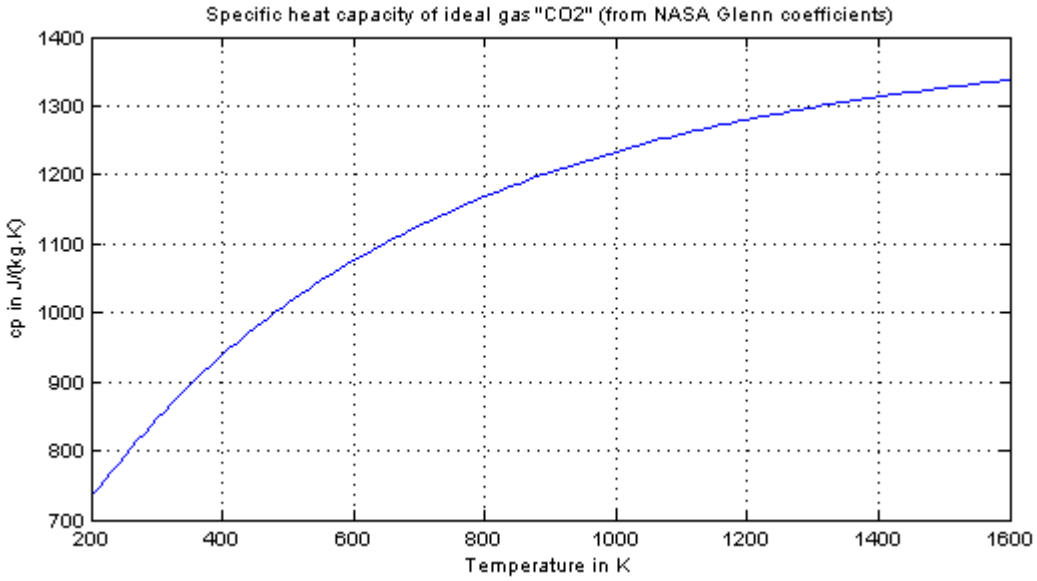
Information



Modelica.Media.IdealGases.SingleGases.CO2

Ideal gas "CO2" from NASA Glenn coefficients

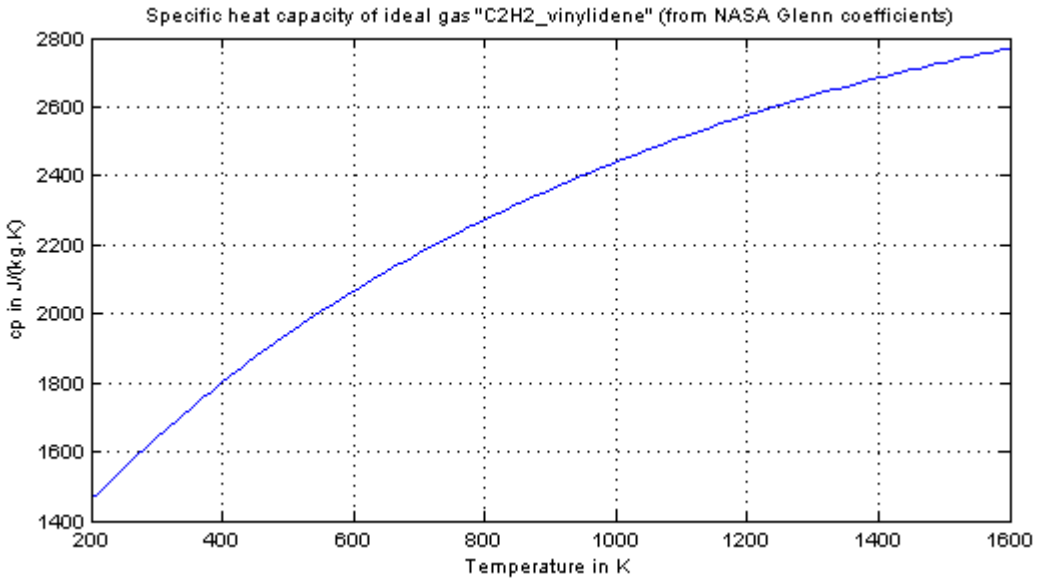
Information



Modelica.Media.IdealGases.SingleGases.C2H2_vinylidene

Ideal gas "C2H2_vinylidene" from NASA Glenn coefficients

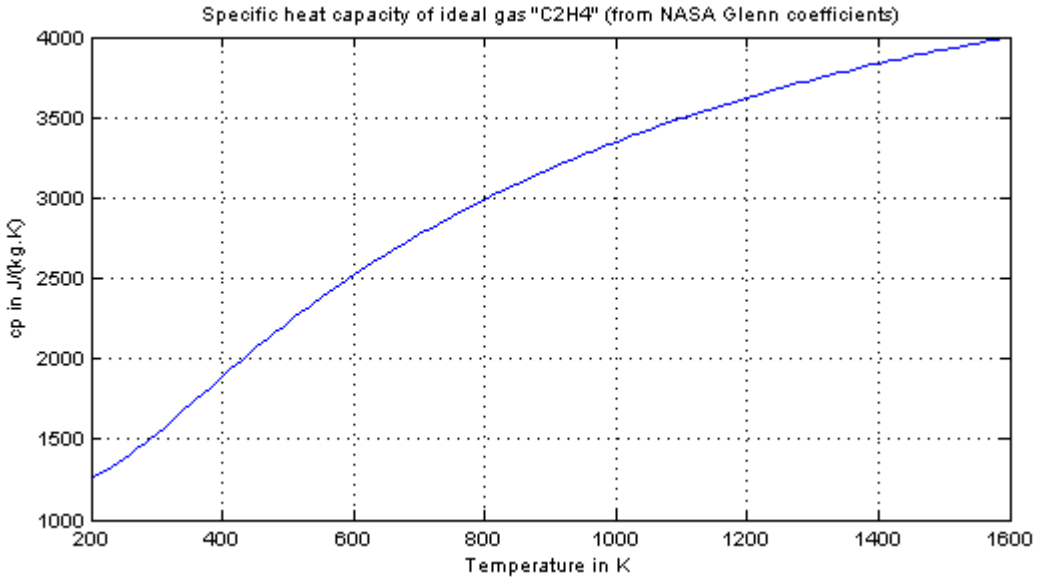
Information



Modelica.Media.IdealGases.SingleGases.C2H4

Ideal gas "C2H4" from NASA Glenn coefficients

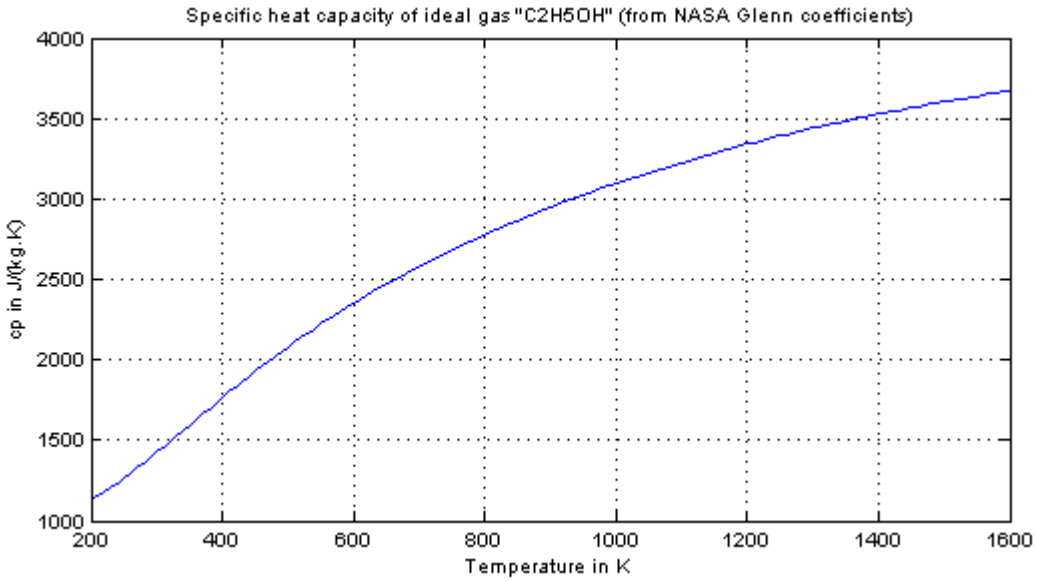
Information



Modelica.Media.IdealGases.SingleGases.C2H5OH

Ideal gas "C2H5OH" from NASA Glenn coefficients

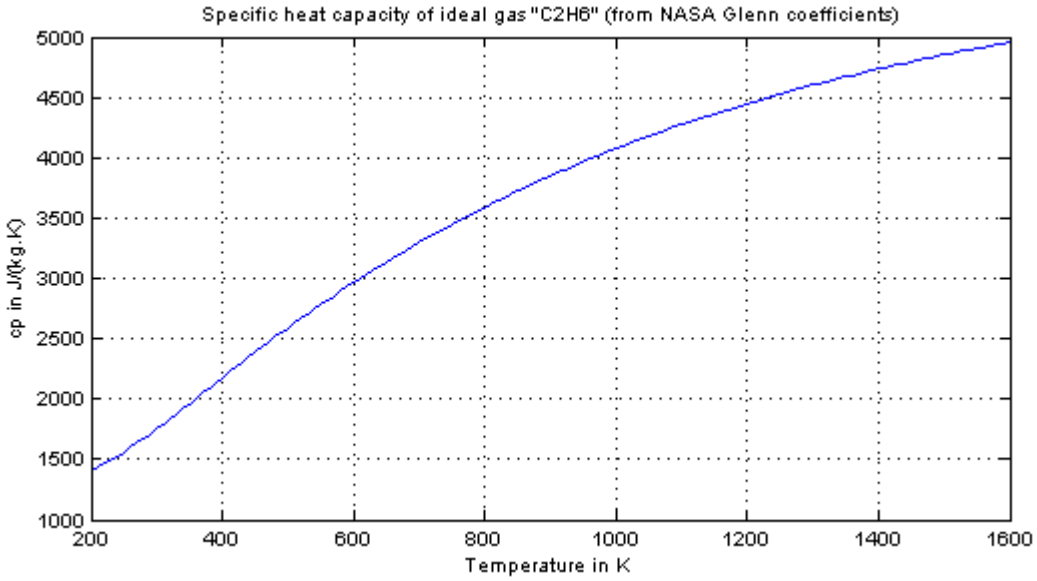
Information



Modelica.Media.IdealGases.SingleGases.C2H6

Ideal gas "C2H6" from NASA Glenn coefficients

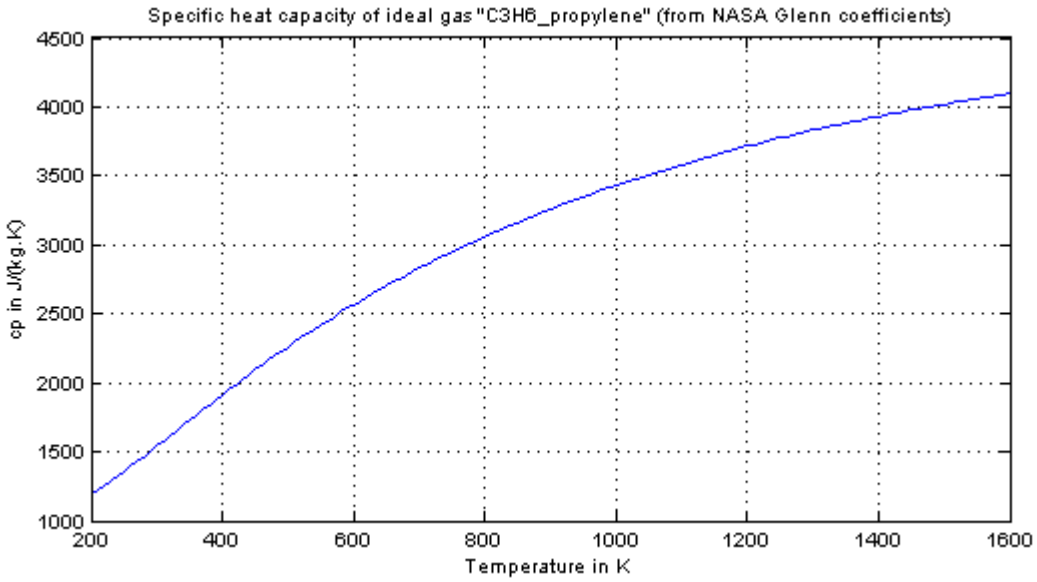
Information



Modelica.Media.IdealGases.SingleGases.C3H6_propylene

Ideal gas "C3H6_propylene" from NASA Glenn coefficients

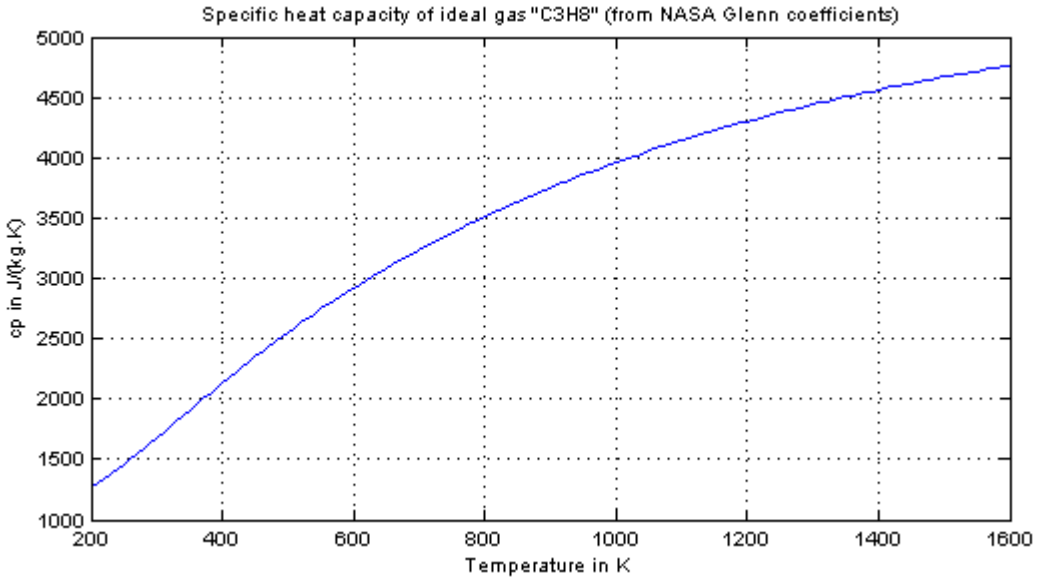
Information



Modelica.Media.IdealGases.SingleGases.C3H8

Ideal gas "C3H8" from NASA Glenn coefficients

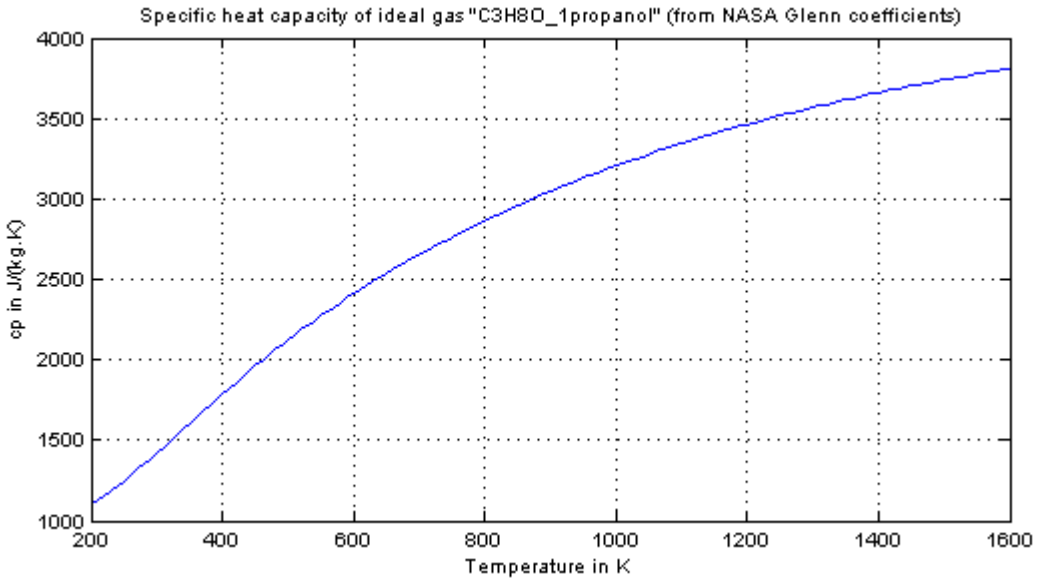
Information



Modelica.Media.IdealGases.SingleGases.C3H8O_1propanol

Ideal gas "C3H8O_1propanol" from NASA Glenn coefficients

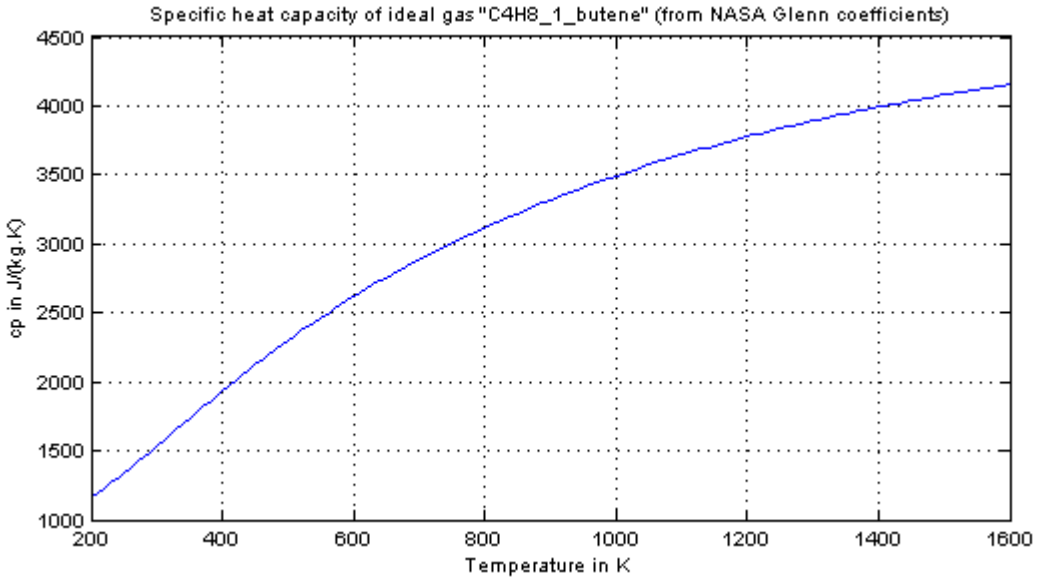
Information



Modelica.Media.IdealGases.SingleGases.C4H8_1_butene

Ideal gas "C4H8_1_butene" from NASA Glenn coefficients

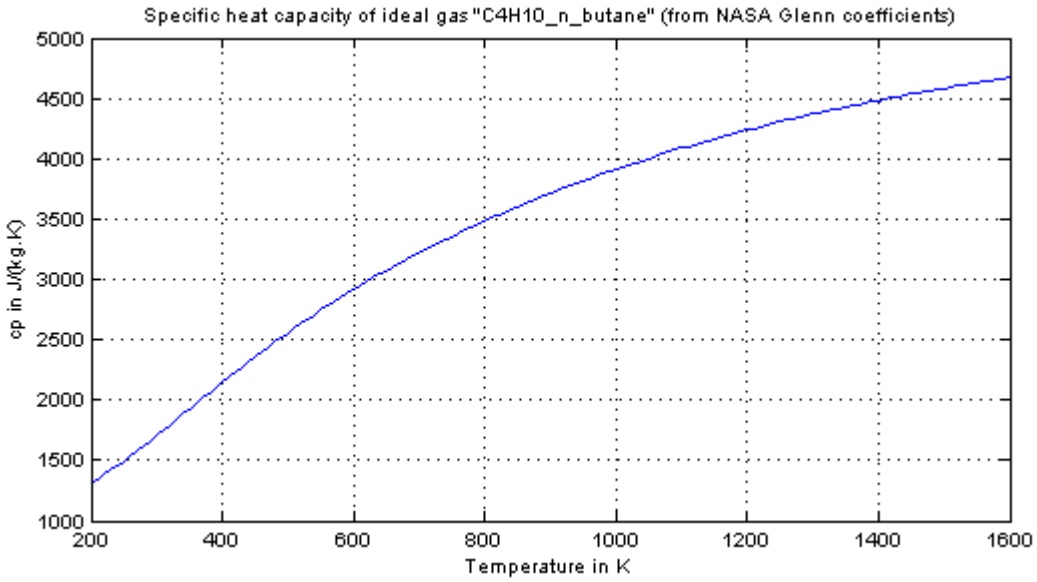
Information



Modelica.Media.IdealGases.SingleGases.C4H10_n_butane

Ideal gas "C4H10_n_butane" from NASA Glenn coefficients

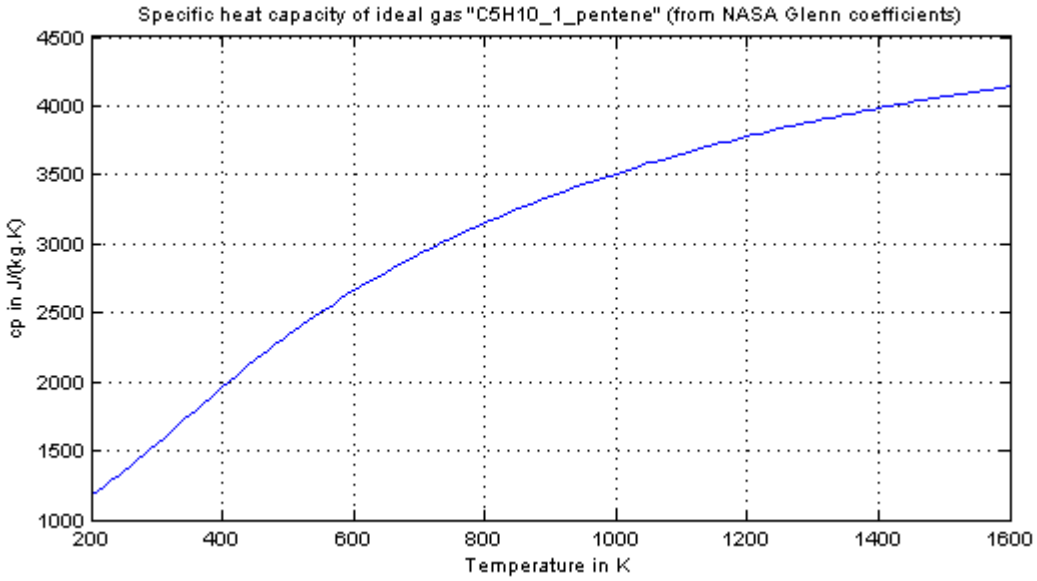
Information



Modelica.Media.IdealGases.SingleGases.C5H10_1_pentene

Ideal gas "C5H10_1_pentene" from NASA Glenn coefficients

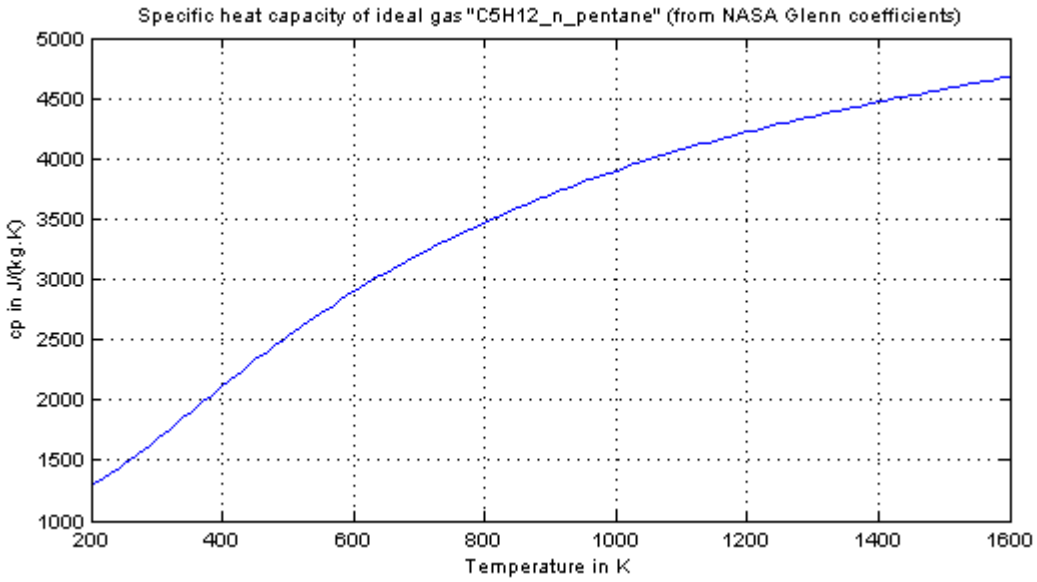
Information



Modelica.Media.IdealGases.SingleGases.C5H12_n_pentane

Ideal gas "C5H12_n_pentane" from NASA Glenn coefficients

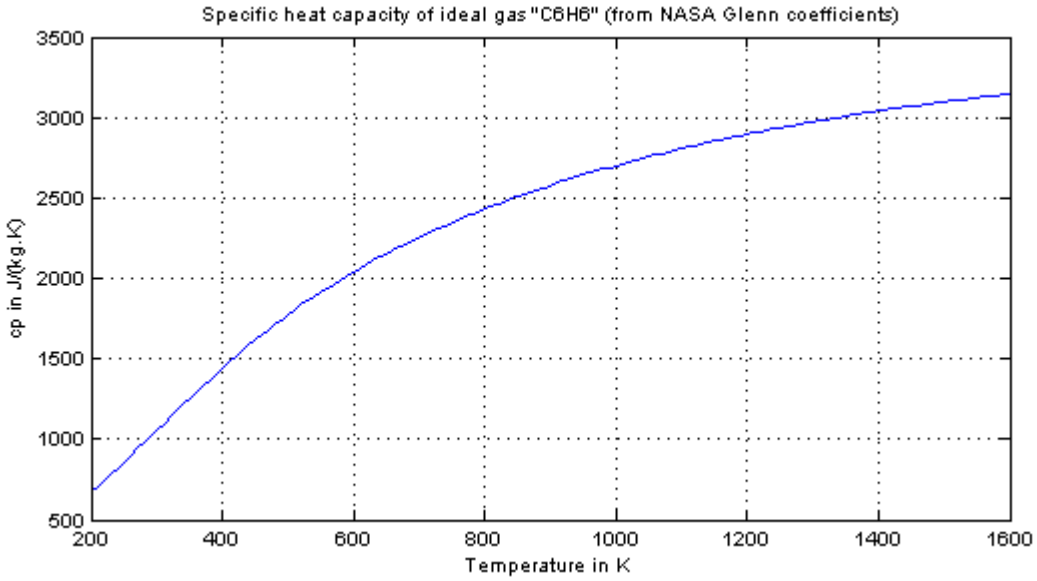
Information



Modelica.Media.IdealGases.SingleGases.C6H6

Ideal gas "C6H6" from NASA Glenn coefficients

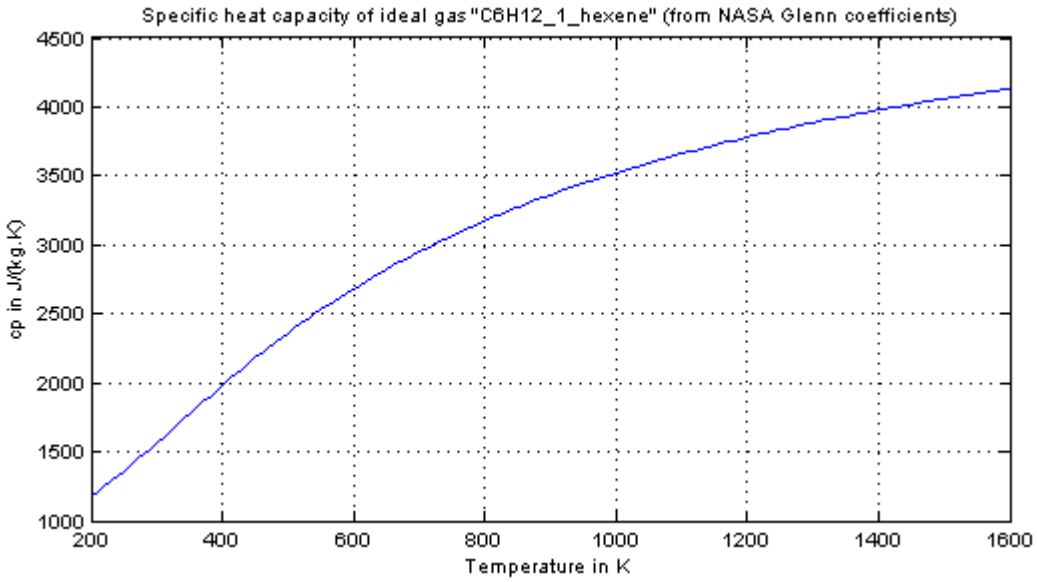
Information



Modelica.Media.IdealGases.SingleGases.C6H12_1_hexene

Ideal gas "C6H12_1_hexene" from NASA Glenn coefficients

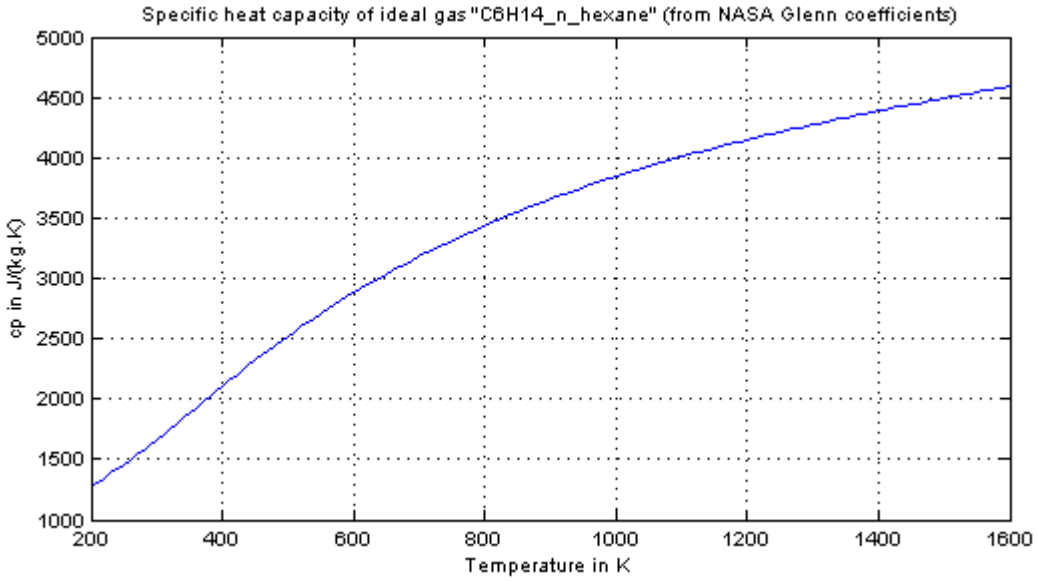
Information



Modelica.Media.IdealGases.SingleGases.C6H14_n_hexane

Ideal gas "C6H14_n_hexane" from NASA Glenn coefficients

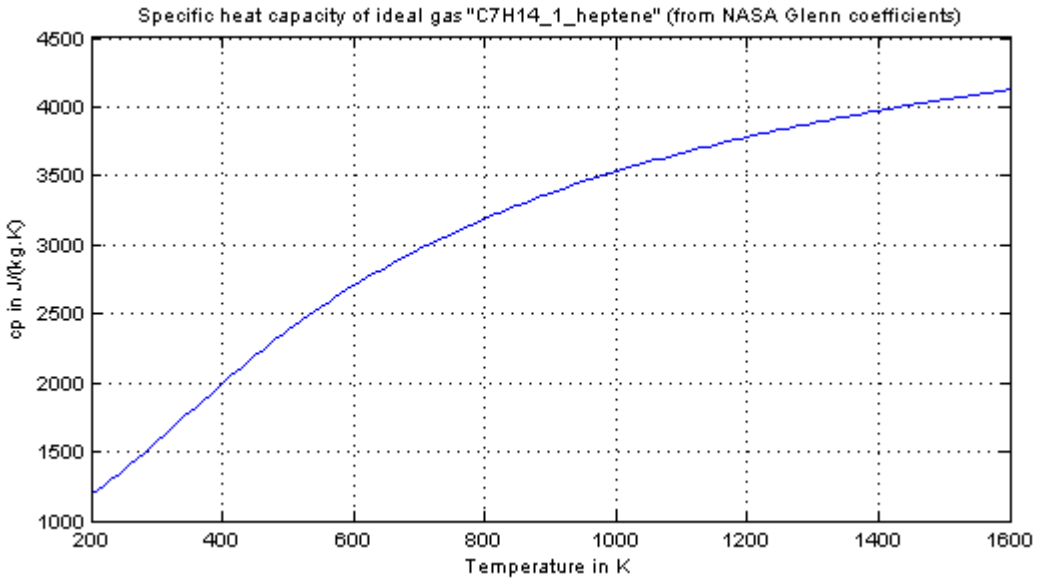
Information



Modelica.Media.IdealGases.SingleGases.C7H14_1_heptene

Ideal gas "C7H14_1_heptene" from NASA Glenn coefficients

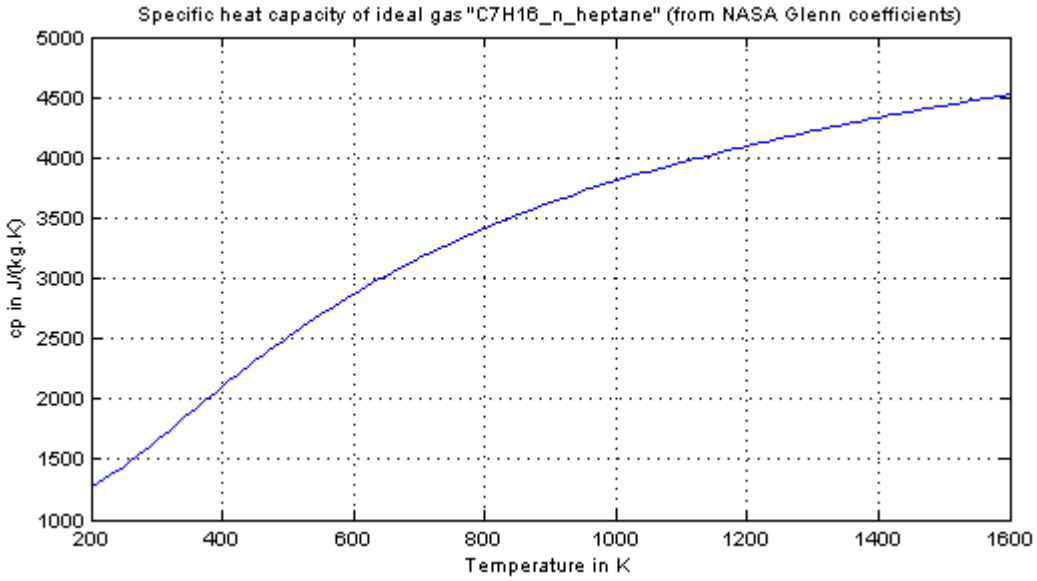
Information



Modelica.Media.IdealGases.SingleGases.C7H16_n_heptane

Ideal gas "C7H16_n_heptane" from NASA Glenn coefficients

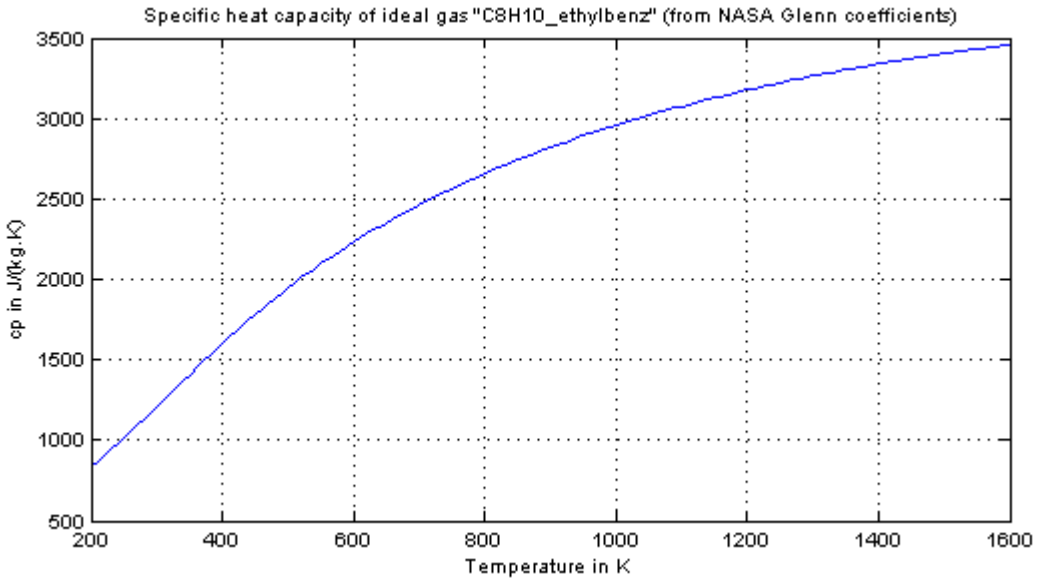
Information



Modelica.Media.IdealGases.SingleGases.C8H10_ethylbenz

Ideal gas "C8H10_ethylbenz" from NASA Glenn coefficients

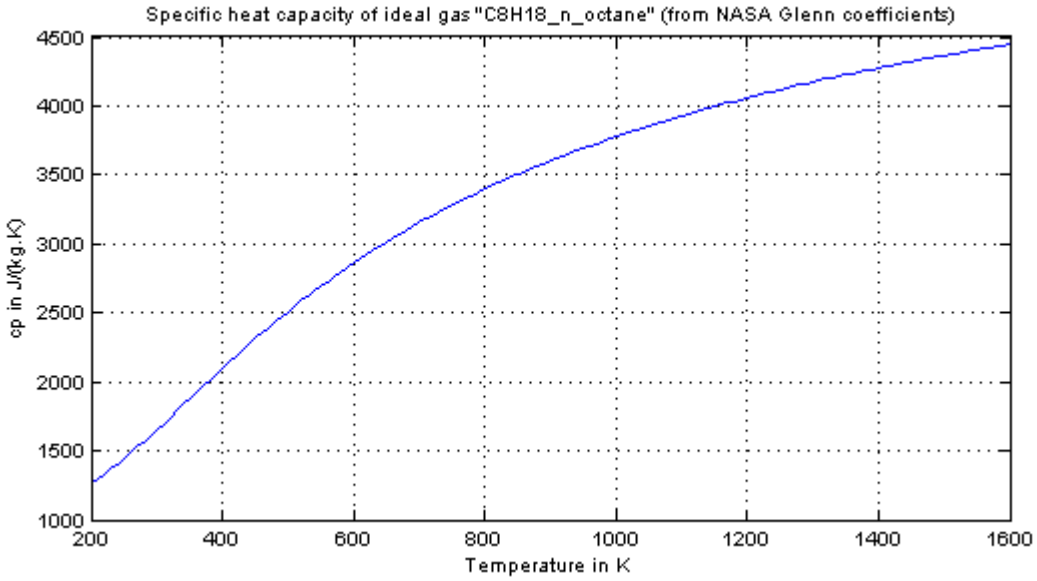
Information



Modelica.Media.IdealGases.SingleGases.C8H18_n_octane

Ideal gas "C8H18_n_octane" from NASA Glenn coefficients

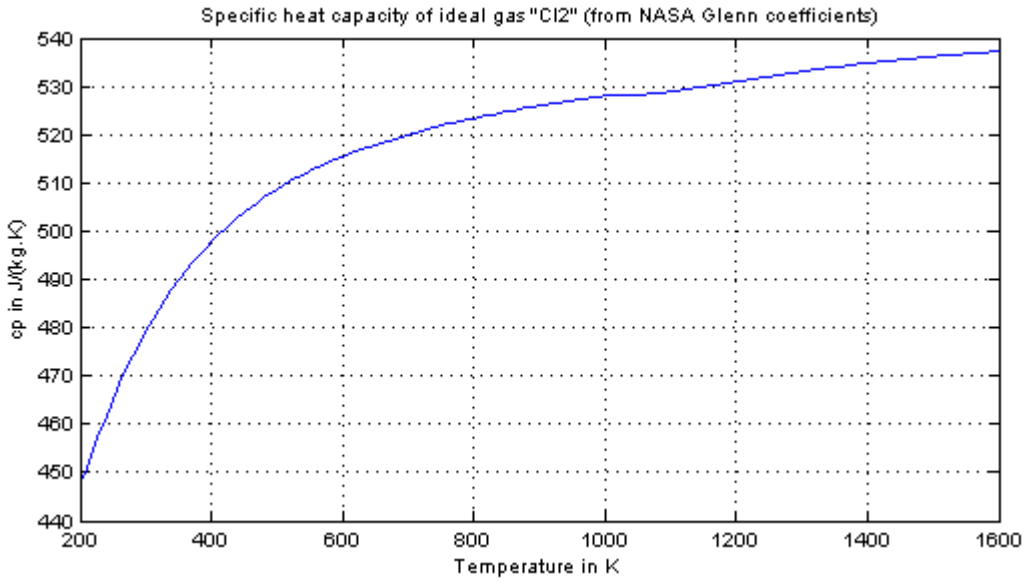
Information



Modelica.Media.IdealGases.SingleGases.CL2

Ideal gas "Cl2" from NASA Glenn coefficients

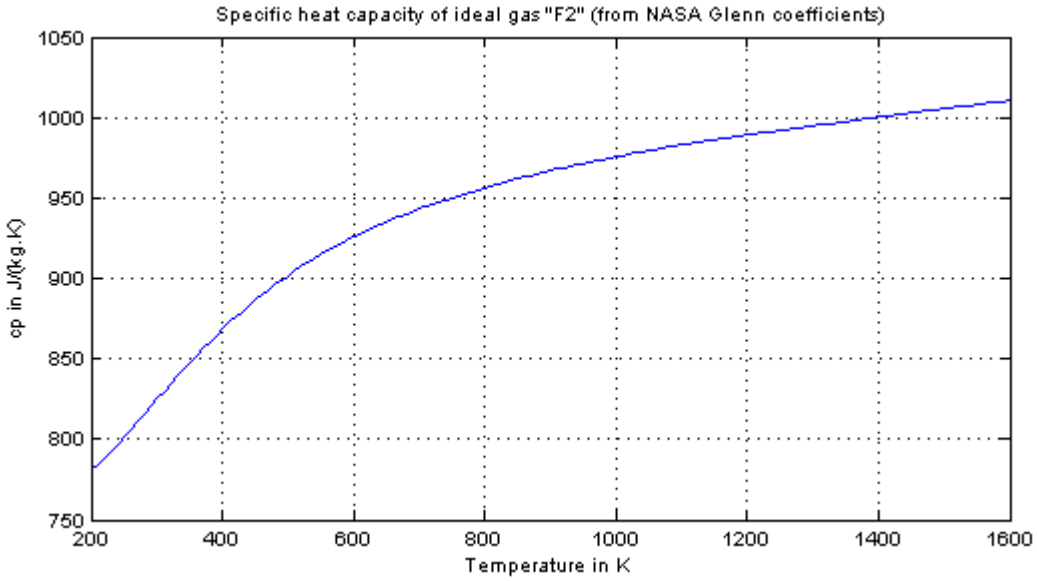
Information



Modelica.Media.IdealGases.SingleGases.F2

Ideal gas "F2" from NASA Glenn coefficients

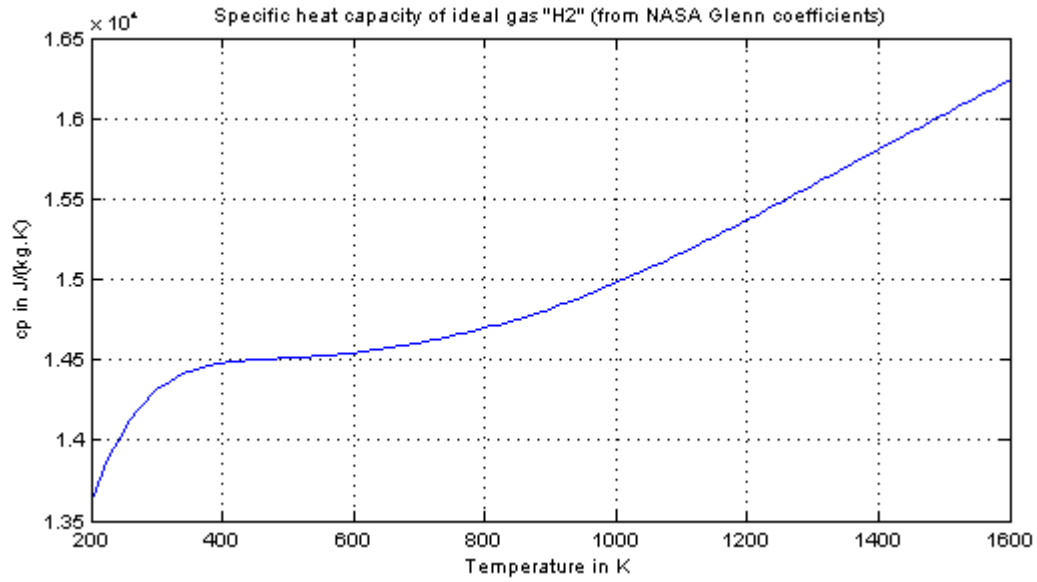
Information



Modelica.Media.IdealGases.SingleGases.H2

Ideal gas "H2" from NASA Glenn coefficients

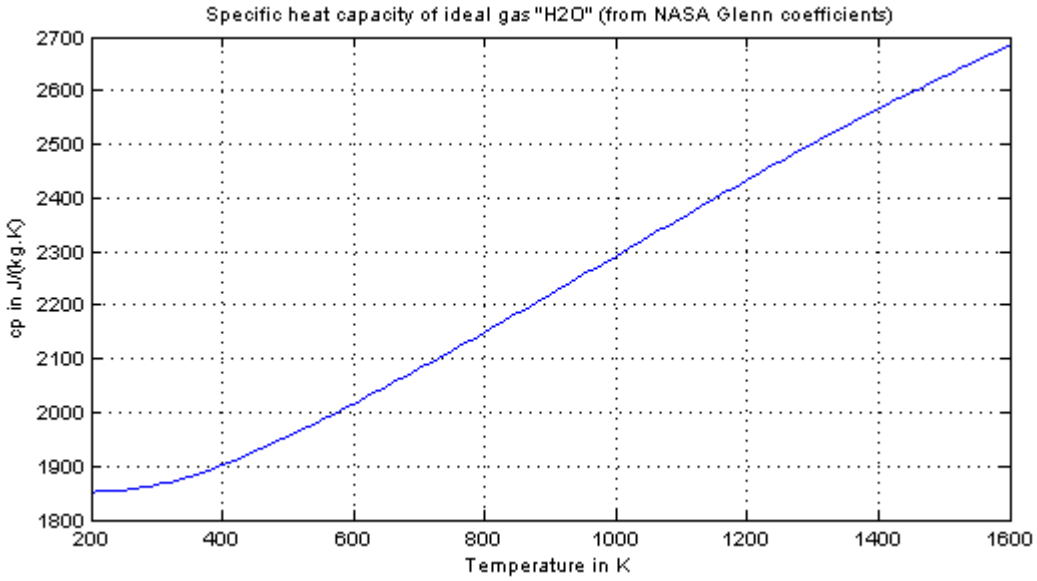
Information



Modelica.Media.IdealGases.SingleGases.H2O

Ideal gas "H2O" from NASA Glenn coefficients

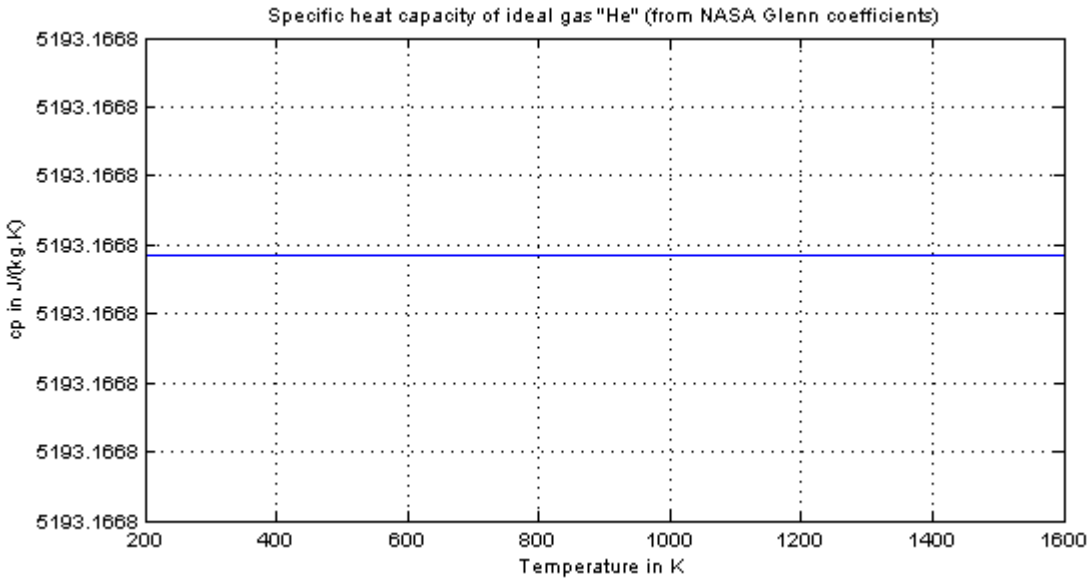
Information



Modelica.Media.IdealGases.SingleGases.He

Ideal gas "He" from NASA Glenn coefficients

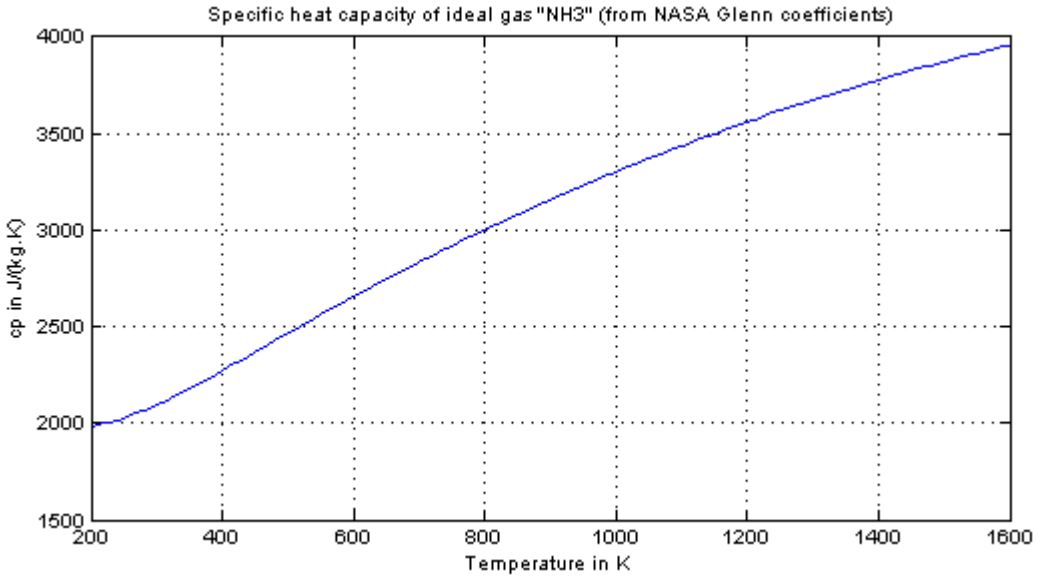
Information



Modelica.Media.IdealGases.SingleGases.NH3

Ideal gas "NH3" from NASA Glenn coefficients

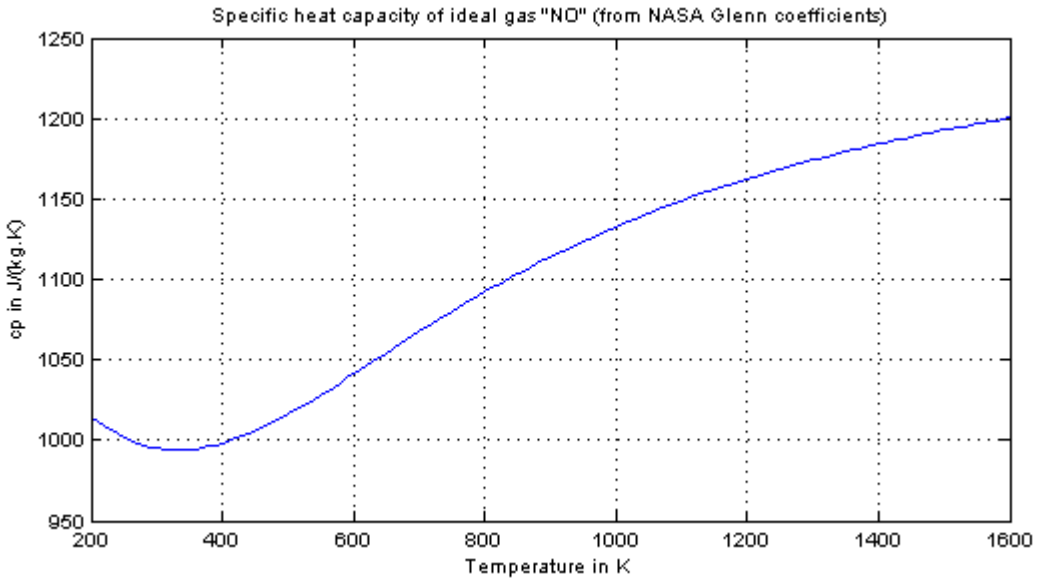
Information



Modelica.Media.IdealGases.SingleGases.NO

Ideal gas "NO" from NASA Glenn coefficients

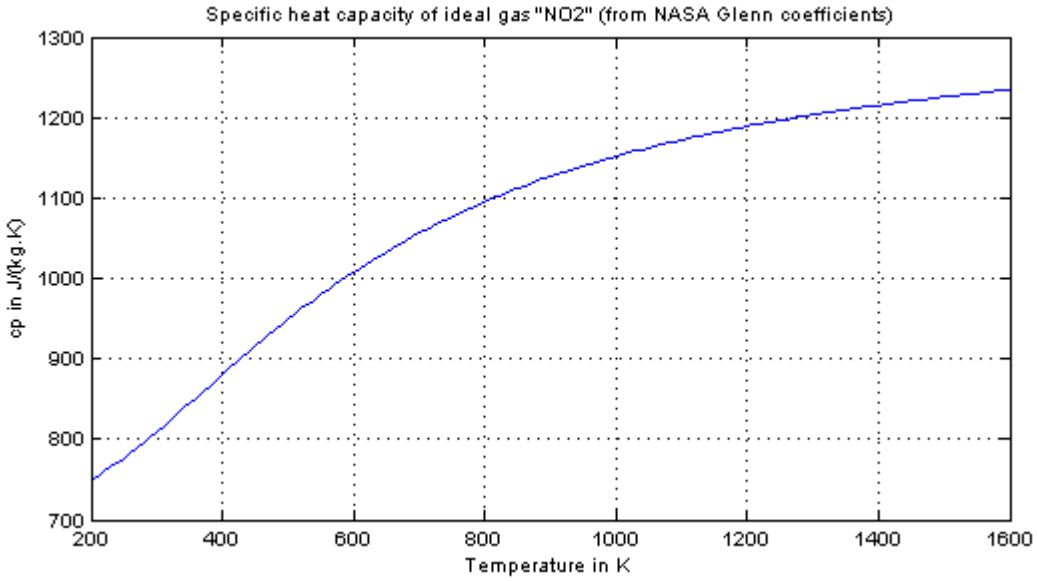
Information



Modelica.Media.IdealGases.SingleGases.NO2

Ideal gas "NO2" from NASA Glenn coefficients

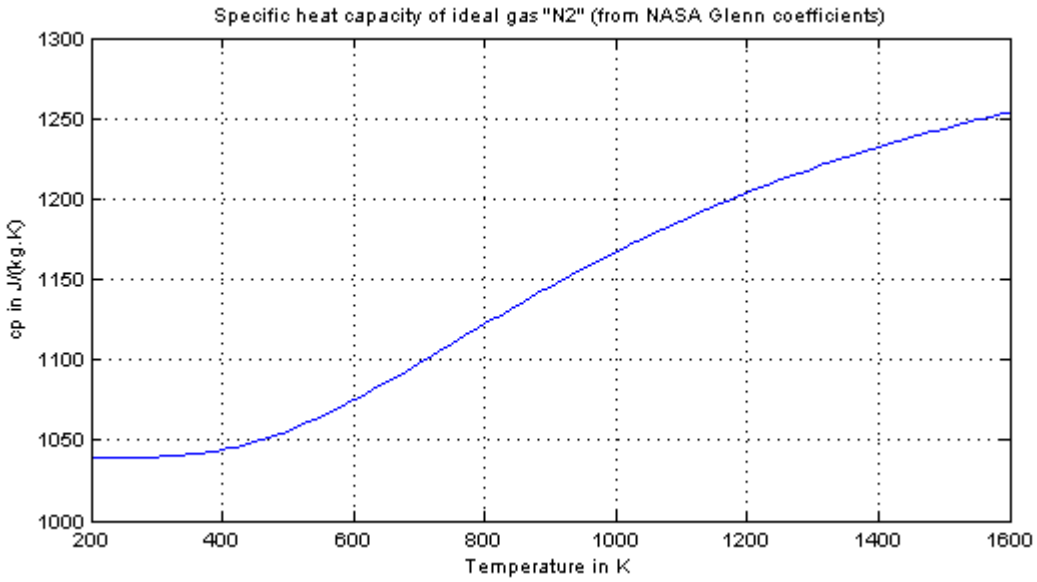
Information



Modelica.Media.IdealGases.SingleGases.N2

Ideal gas "N2" from NASA Glenn coefficients

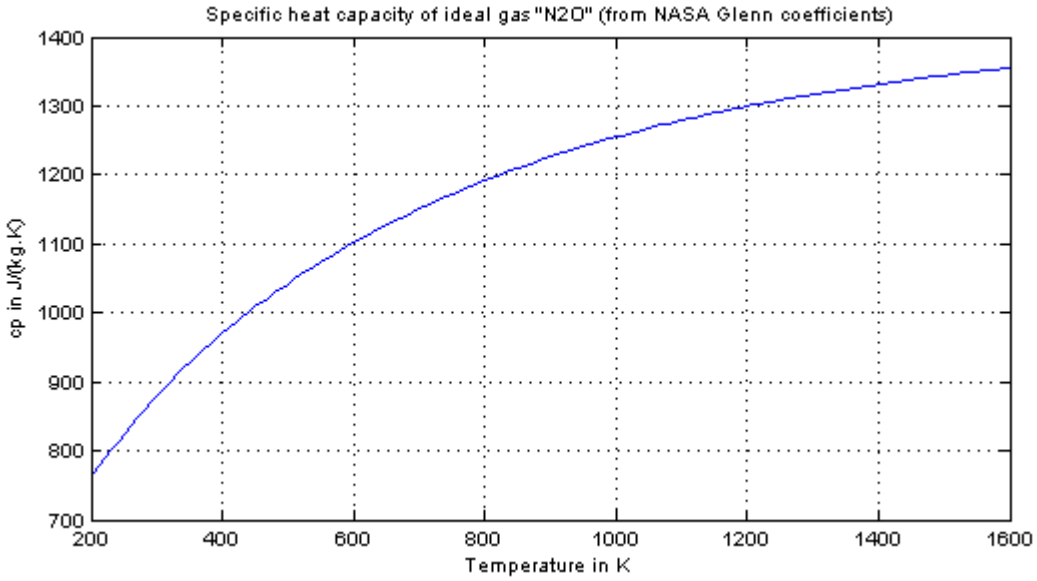
Information



Modelica.Media.IdealGases.SingleGases.N2O

Ideal gas "N2O" from NASA Glenn coefficients

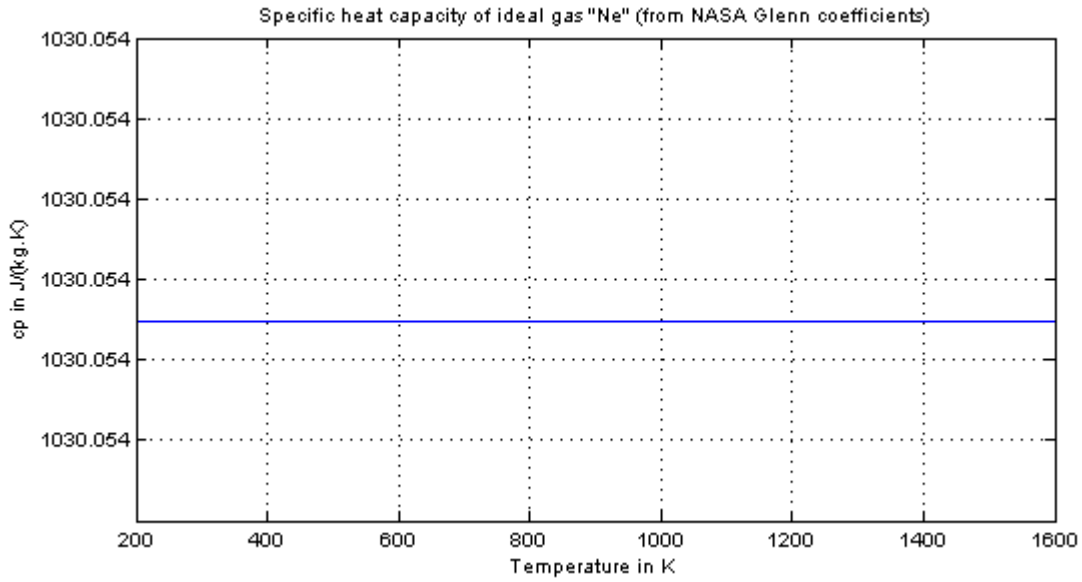
Information



Modelica.Media.IdealGases.SingleGases.Ne

Ideal gas "Ne" from NASA Glenn coefficients

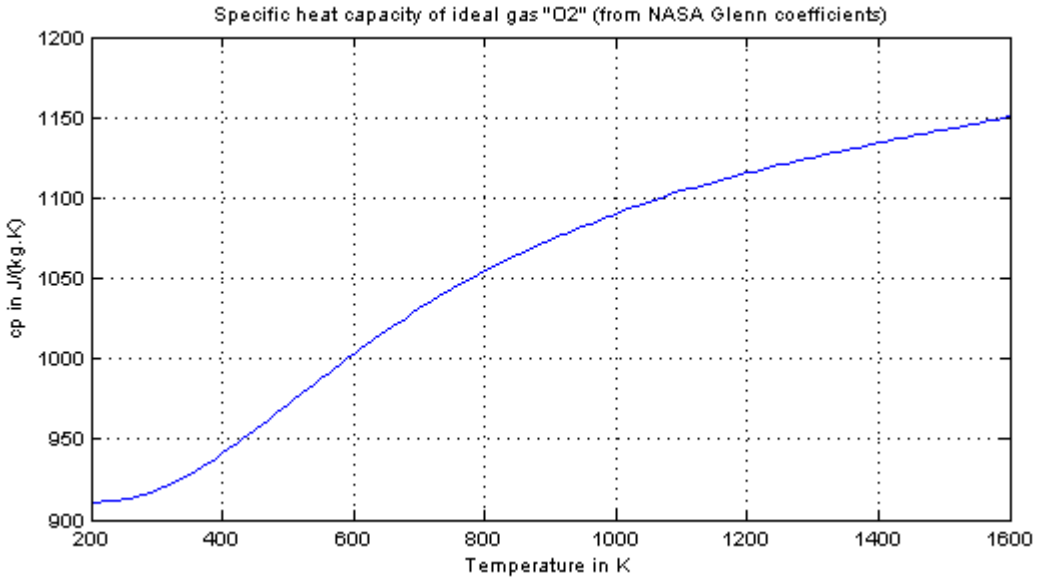
Information



Modelica.Media.IdealGases.SingleGases.O2

Ideal gas "O2" from NASA Glenn coefficients

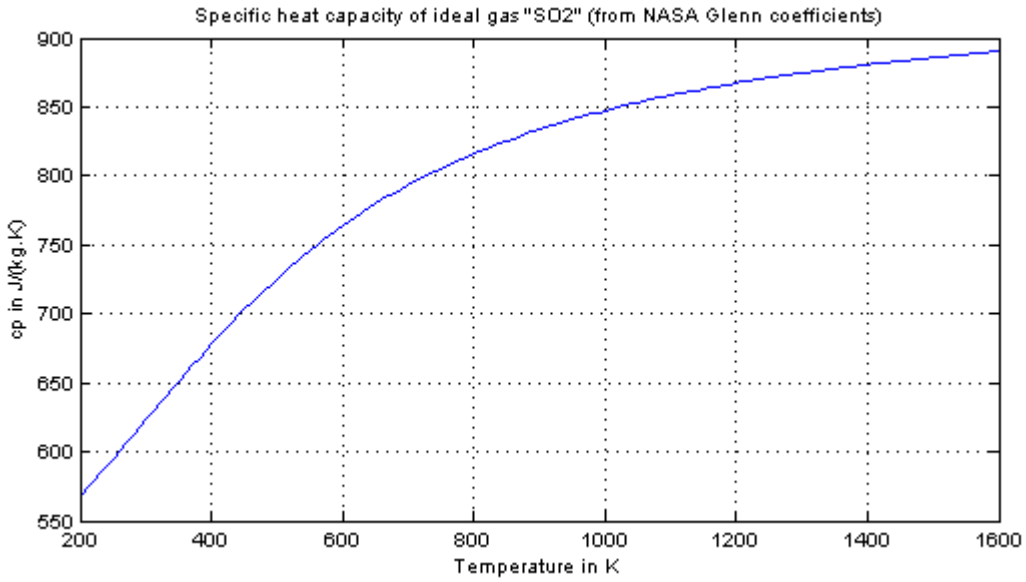
Information



Modelica.Media.IdealGases.SingleGases.SO2

Ideal gas "SO2" from NASA Glenn coefficients

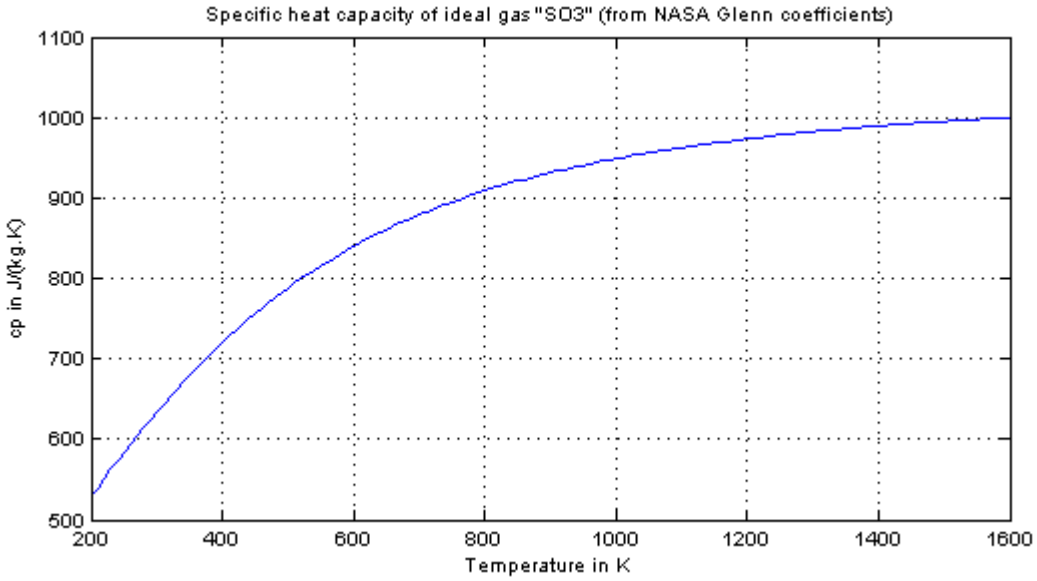
Information



Modelica.Media.IdealGases.SingleGases.SO3

Ideal gas "SO3" from NASA Glenn coefficients

Information



Modelica.Media.IdealGases.MixtureGases

Medium models consisting of mixtures of ideal gases

Information

Package Content

Name	Description
CombustionAir	Air as mixture of N2 and O2
AirSteam	air and steam mixture (no condensation!, pseudo-mixture)
FlueGasLambdaOnePlus	simple flue gas for over0stochiometric O2-fuel ratios
FlueGasSixComponents	simplest flue gas for over-and understochiometric combustion of hydrocarbons
SimpleNaturalGas	simple natural gas mix with 6 components
SimpleNaturalGasFixedComposition	Same as SimpleNaturalGas but with fixed composition

Types and constants

```
package simpleMoistAir = AirSteam(reference_X={0.03,0.97})
"moist air without condensation";
```

Modelica.Media.IdealGases.MixtureGases.CombustionAir

Air as mixture of N2 and O2

Information

Modelica.Media.IdealGases.MixtureGases.AirSteam

air and steam mixture (no condensation!, pseudo-mixture)

Information

Modelica.Media.IdealGases.MixtureGases.FlueGasLambdaOnePlus

simple flue gas for over0stochiometric O2-fuel ratios

Information

Modelica.Media.IdealGases.MixtureGases.FlueGasSixComponents

simplest flue gas for over-and understochiometric combustion of hydrocarbons

Information

Modelica.Media.IdealGases.MixtureGases.SimpleNaturalGas

simple natural gas mix with 6 components

Information

Modelica.Media.IdealGases.MixtureGases.SimpleNaturalGasFixedComposition

Same as SimpleNaturalGas but with fixed composition

Information

Modelica.Media.Incompressible

Medium model for T-dependent properties, defined by tables or polynomials

Information

Incompressible media package

This package provides a structure and examples of how to create simple medium models of incompressible fluids, meaning fluids with very little pressure influence on density. The medium properties is typically described in terms of tables, functions or polynomial coefficients.

Definitions

The common meaning of *incompressible* is that properties like density and enthalpy are independent of pressure. Thus properties are conveniently described as functions of temperature, e.g. as polynomials density(T) and cp(T). However, enthalpy can not be independent of pressure since h = u - p/d. For liquids it is anyway common to neglect this dependence since for constant density the neglected term is (p - p0)/d, which in comparison with cp is very small for most liquids. For water, the equivalent change of temperature to increasing pressure 1 bar is 0.025 Kelvin.

Two boolean flags are used to choose how enthalpy and inner energy is calculated:

- **enthalpyOfT**=true, means assuming that enthalpy is only a function of temperature, neglecting the pressure dependent term.
- **singleState**=true, means also neglect the pressure influence on inner energy, which makes all medium properties pure functions of temperature.

The default setting for both these flags is true, which enables the simulation tool to choose temperature as the only medium state and avoids non-linear equation systems, see the section about Static state selection in the Modelica.Media User's Guide.

Contents

Currently, the package contains the following parts:

1. Table based medium models
2. Example medium models

A few examples are given in the Examples package. The model Examples.Glycol47 shows how the medium models can be used. For more realistic examples of how to implement volume models with medium properties look in the Medium usage section of the User's Guide.

Package Content

Name	Description
Common	Common data structures
TableBased	Incompressible medium properties based on tables
Examples	Examples for incompressible media

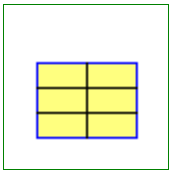
Modelica.Media.Incompressible.Common

Common data structures

Package Content

Name	Description
BaseProps_Tpoly	Fluid state record

Modelica.Media.Incompressible.Common.BaseProps_Tpoly



Fluid state record

Modelica definition

```
record BaseProps_Tpoly "Fluid state record"
  extends Modelica.Media.Interfaces.PartialMedium.ThermodynamicState;
  //      SI.SpecificHeatCapacity cp "specific heat capacity";
  SI.Temperature T "temperature";
  SI.Pressure p "pressure";
  //    SI.Density d "density";
end BaseProps_Tpoly;
```

Modelica.Media.Incompressible.TableBased

Incompressible medium properties based on tables

Information

This is the base package for medium models of incompressible fluids based on tables. The minimal data to provide for a useful medium description is tables of density and heat capacity as functions of temperature.

Using the package TableBased

To implement a new medium model, create a package that **extends** TableBased and provides one or more of the constant tables:

```
tableDensity        = [T, d];
tableHeatCapacity   = [T, Cp];
tableConductivity   = [T, lam];
tableViscosity      = [T, eta];
tableVaporPressure  = [T, pVap];
```

The table data is used to fit constant polynomials of order **npol**, the temperature data points do not need to be same for different properties. Properties like enthalpy, inner energy and entropy are calculated consistently from integrals and derivatives of d(T) and Cp(T). The minimal data for a useful medium model is thus density and heat capacity. Transport properties and vapor pressure are optional, if the data tables are empty the corresponding function calls can not be used.

Package Content

Name	Description
enthalpyOfT=true	true if enthalpy is approximated as a function of T only, (p-dependence neglected)
densityOfT=size(tableDensity, 1) > 1	true if density is a function of temperature
T_min	Minimum temperature valid for medium model
T_max	Maximum temperature valid for medium model
T0=273.15	reference Temperature
h0=0	reference enthalpy at T0, reference_p
s0=0	reference entropy at T0, reference_p
MM_const=0.1	Molar mass
npol=2	degree of polynomial used for fitting
neta=size(tableViscosity, 1)	number of data points for viscosity
tableDensity	Table for rho(T)
tableHeatCapacity	Table for Cp(T)
tableViscosity	Table for eta(T)
tableVaporPressure	Table for pVap(T)
tableConductivity	Table for lambda(T)
TinK	true if T[K],Kelvin used for table temperatures
hasDensity=not (size(tableDensity, 1) == 0)	
hasHeatCapacity=not (size(tableHeatCapacity, 1) == 0)	
hasViscosity=not (size(tableViscosity, 1) == 0)	
hasVaporPressure=not (size(tableVaporPressure, 1) == 0)	
invTK=invertTemp(tableViscosity[:, 1], TinK)	
poly_rho=if hasDensity then Poly.fitting(tableDensity[:, 1], tableDensity[:, 2], npol) else zeros(npol + 1)	
poly_Cp=if hasHeatCapacity then Poly.fitting(tableHeatCapacity[:, 1], tableHeatCapacity[:, 2], npol) else zeros(npol + 1)	
poly_eta=if hasViscosity then Poly.fitting(invTK, Math.log(tableViscosity[:, 2]), npol) else zeros(npol + 1)	
poly_pVap=if hasVaporPressure then Poly.fitting(tableVaporPressure[:, 1], tableVaporPressure[:, 2], npol) else zeros(npol + 1)	
poly_lam=if size(tableConductivity, 1) > 0 then Poly.fitting(tableConductivity[:, 1], tableConductivity[:, 2], npol) else zeros(npol + 1)	

invertTemp	function to invert temperatures
BaseProperties	Base properties of T dependent medium
setState_pTX	Returns state record, given pressure and temperature
setState_dTX	Returns state record, given pressure and temperature
setState_pT	returns state record as function of p and T
setState_phX	Returns state record, given pressure and specific enthalpy
setState_ph	returns state record as function of p and h
setState_psX	Returns state record, given pressure and specific entropy
setState_ps	returns state record as function of p and s
specificHeatCapacityCv	Specific heat capacity at constant volume (or pressure) of medium
specificHeatCapacityCp	Specific heat capacity at constant volume (or pressure) of medium
dynamicViscosity	
thermalConductivity	
s_T	compute specific entropy
specificEntropy	
h_T	Compute specific enthalpy from temperature
h_T_der	Compute specific enthalpy from temperature
h_pT	Compute specific enthalpy from pressure and temperature
density_T	
temperature	
pressure	
density	
specificEnthalpy	
specificInternalEnergy	
T_ph	Compute temperature from pressure and specific enthalpy
T_ps	Compute temperature from pressure and specific enthalpy
Polynomials_Temp	Temporary Functions operating on polynomials (including polynomial fitting); only to be used in Modelica.Media.Incompressible.TableBased
Inherited	
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set

	reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
FluidConstants	critical, triple, molecular and other standard data of fluid
ThermodynamicState	Minimal variable set that is available as input argument to every medium function
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
isentropicExponent	Return isentropic exponent
isentropicEnthalpy	Return isentropic enthalpy
velocityOfSound	Return velocity of sound
isobaricExpansionCoefficient	Return overall the isobaric expansion coefficient beta
beta	alias for isobaricExpansionCoefficient for user convenience
isothermalCompressibility	Return overall the isothermal compressibility factor
kappa	alias of isothermalCompressibility for user convenience
density_derp_h	Return density derivative wrt pressure at const specific enthalpy
density_derh_p	Return density derivative wrt specific enthalpy at constant pressure
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure

density_derX	Return density derivative wrt mass fraction
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from p, T, and X or Xi
density_pTX	Return density from p, T, and X or Xi
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p,s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes

DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
  constant Boolean enthalpyOfT=true
  "true if enthalpy is approximated as a function of T only, (p-dependence
neglected)";

  constant Boolean densityOfT = size(tableDensity,1) > 1
  "true if density is a function of temperature";

  constant Temperature T_min "Minimum temperature valid for medium model";

  constant Temperature T_max "Maximum temperature valid for medium model";

  constant Temperature T0=273.15 "reference Temperature";

  constant SpecificEnthalpy h0=0 "reference enthalpy at T0, reference_p";

  constant SpecificEntropy s0=0 "reference entropy at T0, reference_p";

  constant MolarMass MM_const=0.1 "Molar mass";

  constant Integer npol=2 "degree of polynomial used for fitting";

  constant Integer neta=size(tableViscosity,1)
  "number of data points for viscosity";

  constant Real[:,:] tableDensity "Table for rho(T)";

  constant Real[:,:] tableHeatCapacity "Table for Cp(T)";

  constant Real[:,:] tableViscosity "Table for eta(T)";

  constant Real[:,:] tableVaporPressure "Table for pVap(T)";

  constant Real[:,:] tableConductivity "Table for lambda(T)";

  constant Boolean TinK "true if T[K],Kelvin used for table temperatures";

  constant Boolean hasDensity = not (size(tableDensity,1)==0);

  constant Boolean hasHeatCapacity = not (size(tableHeatCapacity,1)==0);

  constant Boolean hasViscosity = not (size(tableViscosity,1)==0);

  constant Boolean hasVaporPressure = not (size(tableVaporPressure,1)==0);
```

```
  final constant Real invTK[neta] = invertTemp(tableViscosity[:,1],TinK);

  final constant Real poly_rho[:] = if hasDensity then
                                      Poly.fitting(tableDensity[:,1],tableDensi
ty[:,2],npol) else
                                        zeros(npol+1);

  final constant Real poly_Cp[:] = if hasHeatCapacity then
                                      Poly.fitting(tableHeatCapacity[:,1],table
HeatCapacity[:,2],npol) else
                                        zeros(npol+1);

  final constant Real poly_eta[:] = if hasViscosity then
                                      Poly.fitting(invTK,
Math.log(tableViscosity[:,2]),npol) else
                                        zeros(npol+1);

  final constant Real poly_pVap[:] = if hasVaporPressure then
                                      Poly.fitting(tableVaporPressure[:,1],tabl
eVaporPressure[:,2],npol) else
                                       zeros(npol+1);

  final constant Real poly_lam[:] = if size(tableConductivity,1)>0 then
                                      Poly.fitting(tableConductivity[:,1],table
Conductivity[:,2],npol) else
                                        zeros(npol+1);
```

Modelica.Media.Incompressible.TableBased.invertTemp

function to invert temperatures

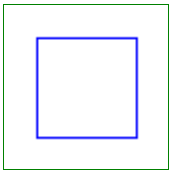
Inputs

Type	Name	Default	Description
Real	table[:]		table temperature data
Boolean	Tink		flag for Celsius or Kelvin

Outputs

Type	Name	Description
Real	invTable[size(table, 1)]	inverted temperatures

Modelica.Media.Incompressible.TableBased.BaseProperties



Base properties of T dependent medium

Information

Note that the inner energy neglects the pressure dependence, which is only true for an incompressible medium with d = constant. The neglected term is p-reference_p)/rho*(T/rho)*(partial rho /partial T). This is very small for liquids due to proportionality to 1/d^2, but can be problematic for gases that are modeled incompressible.

Enthalpy is never a function of T only (h = h(T) + (p-reference_p)/d), but the error is also small and non-linear systems can be avoided. In particular, non-linear systems are small and local as opposed to large and over all volumes.

Entropy is calculated as

```
s = s0 + integral(Cp(T)/T,dt)
```

which is only exactly true for a fluid with constant density d=d0.

Parameters

Type	Name	Default	Description
Temperature	T_start	298.15	initial temperature [K]
SpecificHeatCapacity	R	Modelica.Constants.R	Gas constant (of mixture if applicable) [J/(kg.K)]
Pressure_bar	p_bar	Cv.to_bar(p)	Absolute pressure of medium in [bar] [bar]
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

---

Modelica.Media.Incompressible.TableBased.setState_pTX

Returns state record, given pressure and temperature

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Incompressible.TableBased.setState_dTX

Returns state record, given pressure and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]

Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Incompressible.TableBased.setState_pT

returns state record as function of p and T

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]
Temperature	T		temperature [K]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state

Modelica.Media.Incompressible.TableBased.setState_phX

Returns state record, given pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Incompressible.TableBased.setState_ph

returns state record as function of p and h

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state

Modelica.Media.Incompressible.TableBased.setState_psX

Returns state record, given pressure and specific entropy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Incompressible.TableBased.setState_ps

returns state record as function of p and s

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state

Modelica.Media.Incompressible.TableBased.specificHeatCapacityCv

Specific heat capacity at constant volume (or pressure) of medium

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Incompressible.TableBased.specificHeatCapacityCp

Specific heat capacity at constant volume (or pressure) of medium

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Incompressible.TableBased.dynamicViscosity

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Incompressible.TableBased.thermalConductivity

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Incompressible.TableBased.s_T

compute specific entropy

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description

SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Incompressible.TableBased.specificEntropy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.Incompressible.TableBased.h_T

Compute specific enthalpy from temperature

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy at p, T [J/kg]

Modelica.Media.Incompressible.TableBased.h_T_der

Compute specific enthalpy from temperature

Inputs

Type	Name	Default	Description
Temperature	T		Temperature [K]
Real	dT		temperature derivative

Outputs

Type	Name	Description
Real	dh	derivative of Specific enthalpy at T

Modelica.Media.Incompressible.TableBased.h_pT

Compute specific enthalpy from pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
Boolean	densityOfT	false	include or neglect density derivative dependence of enthalpy

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy at p, T [J/kg]

Modelica.Media.Incompressible.TableBased.density_T

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Incompressible.TableBased.temperature

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Incompressible.TableBased.pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Incompressible.TableBased.density

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Incompressible.TableBased.specificEnthalpy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Incompressible.TableBased.specificInternalEnergy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	u	Specific internal energy [J/kg]

Modelica.Media.Incompressible.TableBased.T_ph

Compute temperature from pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
Temperature	T	temperature [K]

Modelica.Media.Incompressible.TableBased.T_ps

Compute temperature from pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	temperature [K]

Modelica.Media.Incompressible.TableBased.Polynomials_Temp

Temporary Functions operating on polynomials (including polynomial fitting); only to be used in Modelica.Media.Incompressible.TableBased

Information

This package contains functions to operate on polynomials, in particular to determine the derivative and the integral of a polynomial and to use a polynomial to fit a given set of data points.

Copyright © 2004-2007, Modelica Association and DLR.

*This package is **free** software. It can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** in the documentation of package Modelica in file "Modelica/package.mo".*

Package Content

Name	Description
evaluate	Evaluate polynomial at a given abszissa value
derivative	Derivative of polynomial
derivativeValue	Value of derivative of polynomial at abszissa value u
secondDerivativeValue	Value of 2nd derivative of polynomial at abszissa value u
integral	Indefinite integral of polynomial p(u)
integralValue	Integral of polynomial p(u) from u_low to u_high
fitting	Computes the coefficients of a polynomial that fits a set of data points in a least-squares sense
evaluate_der	Evaluate polynomial at a given abszissa value
integralValue_der	Time derivative of integral of polynomial p(u) from u_low to u_high, assuming only u_high as time-dependent (Leibnitz rule)
derivativeValue_der	time derivative of derivative of polynomial

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.evaluate

Evaluate polynomial at a given abszissa value

Inputs

Type	Name	Default	Description
Real	p[:]		Polynomial coefficients (p[1] is coefficient of highest power)
Real	u		Abszissa value

Outputs

Type	Name	Description
Real	y	Value of polynomial at u

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.derivative

Derivative of polynomial

Inputs

Type	Name	Default	Description
Real	p1[:]		Polynomial coefficients (p1[1] is coefficient of highest power)

Outputs

Type	Name	Description
Real	p2[size(p1, 1) - 1]	Derivative of polynomial p1

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.derivativeValue

Value of derivative of polynomial at abszissa value u

Inputs

Type	Name	Default	Description
Real	p[:]		Polynomial coefficients (p[1] is coefficient of highest power)
Real	u		Abszissa value

Outputs

Type	Name	Description
Real	y	Value of derivative of polynomial at u

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.secondDerivativeValue

Value of 2nd derivative of polynomial at abszissa value u

Inputs

Type	Name	Default	Description
Real	p[:]		Polynomial coefficients (p[1] is coefficient of highest power)
Real	u		Abszissa value

Outputs

Type	Name	Description
Real	y	Value of 2nd derivative of polynomial at u

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.integral

Indefinite integral of polynomial p(u)

Inputs

Type	Name	Default	Description
Real	p1[:]		Polynomial coefficients (p1[1] is coefficient of highest power)

Outputs

Type	Name	Description
Real	p2[size(p1, 1) + 1]	Polynomial coefficients of indefinite integral of polynomial p1 (polynomial p2 + C is the indefinite integral of p1, where C is an arbitrary constant)

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.integralValue

Integral of polynomial p(u) from u_low to u_high

Inputs

Type	Name	Default	Description
Real	p[:]		Polynomial coefficients
Real	u_high		High integrand value
Real	u_low	0	Low integrand value, default 0

Outputs

Type	Name	Description
Real	integral	Integral of polynomial p from u_low to u_high

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.fitting

Computes the coefficients of a polynomial that fits a set of data points in a least-squares sense

Information

Polynomials.fitting(u,y,n) computes the coefficients of a polynomial p(u) of degree "n" that fits the data "p(u[i]) - y[i]" in a least squares sense. The polynomial is returned as a vector p[n+1] that has the following definition:

```
p(u) = p[1]*u^n + p[2]*u^(n-1) + ... + p[n]*u + p[n+1];
```

Inputs

Type	Name	Default	Description
Real	u[:]		Abscissa data values
Real	y[size(u, 1)]		Ordinate data values
Integer	n		Order of desired polynomial that fits the data points (u,y)

Outputs

Type	Name	Description
Real	p[n + 1]	Polynomial coefficients of polynomial that fits the date points

---

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.evaluate_der

Evaluate polynomial at a given abszissa value

Inputs

Type	Name	Default	Description
Real	p[:]		Polynomial coefficients (p[1] is coefficient of highest power)
Real	u		Abszissa value
Real	du		Abszissa value

Outputs

Type	Name	Description
Real	dy	Value of polynomial at u

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.integralValue_der

Time derivative of integral of polynomial p(u) from u_low to u_high, assuming only u_high as time-dependent (Leibnitz rule)

Inputs

Type	Name	Default	Description
Real	p[:]		Polynomial coefficients
Real	u_high		High integrand value
Real	u_low	0	Low integrand value, default 0
Real	du_high		High integrand value
Real	du_low	0	Low integrand value, default 0

Outputs

Type	Name	Description
Real	dintegral	Integral of polynomial p from u_low to u_high

Modelica.Media.Incompressible.TableBased.Polynomials_Temp.derivativeValue_der

time derivative of derivative of polynomial

Inputs

Type	Name	Default	Description
Real	p[:]		Polynomial coefficients (p[1] is coefficient of highest power)
Real	u		Abszissa value
Real	du		delta of abszissa value

Outputs

Type	Name	Description
Real	dy	time-derivative of derivative of polynomial w.r.t. input variable at u

---

Modelica.Media.Incompressible.Examples

Examples for incompressible media

Information

This package provides a few examples of how to construct medium models for incompressible fluids. The package contains:

- **Glycol47**, a model of 47% glycol water mixture, based on tables of density and heat capacity as functions of temperature.
- **Essotherm650**, a medium model for thermal oil, also based on tables.

Package Content

Name	Description
Glycol47	1,2-Propylene glycol, 47% mixture with water
Essotherm650	Essotherm thermal oil
TestGlycol	Test Glycol47 Medium model

Modelica.Media.Incompressible.Examples.Glycol47

1,2-Propylene glycol, 47% mixture with water

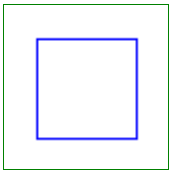
Information

Modelica.Media.Incompressible.Examples.Essotherm650

Essotherm thermal oil

Information

Modelica.Media.Incompressible.Examples.TestGlycol



Test Glycol47 Medium model

Parameters

Type	Name	Default	Description
Temperature	T_start	298.15	initial temperature [K]
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

Modelica.Media.Water

Medium models for water

Information

This package contains different medium models for water:

- **ConstantPropertyLiquidWater**
  Simple liquid water medium (incompressible, constant data).
- **IdealSteam**
  Steam water medium as ideal gas from Media.IdealGases.SingleGases.H2O
- **WaterIF97 derived models**
  High precision water model according to the IAPWS/IF97 standard (liquid, steam, two phase region). Models with different independent variables are provided as well as models valid only for particular regions. The **WaterIF97_ph** model is valid in all regions and is the recommended one to use.

Overview of WaterIF97 derived water models

The WaterIF97 models calculate medium properties for water in the **liquid**, **gas** and **two phase** regions according to the IAPWS/IF97 standard, i.e., the accepted industrial standard and best compromise between accuracy and computation time. It has been part of the ThermoFluid Modelica library and been extended, reorganized and documented to become part of the Modelica Standard library.

An important feature that distinguishes this implementation of the IF97 steam property standard is that this implementation has been explicitly designed to work well in dynamic simulations. Computational performance has been of high importance. This means that there often exist several ways to get the same result from different functions if one of the functions is called often but can be optimized for that purpose.

Three variable pairs can be the independent variables of the model:

1. Pressure **p** and specific enthalpy **h** are the most natural choice for general applications. This is the recommended choice for most general purpose applications, in particular for power plants.
2. Pressure **p** and temperature **T** are the most natural choice for applications where water is always in the same phase, both for liquid water and steam.
3. Density **d** and temperature **T** are explicit variables of the Helmholtz function in the near-critical region and can be the best choice for applications with super-critical or near-critial states.

The following quantities are always computed in Medium.Baseproperties:

Variable	Unit	Description
T	K	temperature

u	J/kg	specific internal energy
d	kg/m^3	density
p	Pa	pressure
h	J/kg	specific enthalpy

In some cases additional medium properties are needed. A component that needs these optional properties has to call one of the following functions:

Function call	Unit	Description
Medium.dynamicViscosity(medium.state)	Pa.s	dynamic viscosity
Medium.thermalConductivity(medium.state)	W/(m.K)	thermal conductivity
Medium.prandtlNumber(medium.state)	1	Prandtl number
Medium.specificEntropy(medium.state)	J/(kg.K)	specific entropy
Medium.heatCapacity_cp(medium.state)	J/(kg.K)	specific heat capacity at constant pressure
Medium.heatCapacity_cv(medium.state)	J/(kg.K)	specific heat capacity at constant density
Medium.isentropicExponent(medium.state)	1	isentropic exponent
Medium.isentropicEnthalpy(pressure, medium.state)	J/kg	isentropic enthalpy
Medium.velocityOfSound(medium.state)	m/s	velocity of sound
Medium.isobaricExpansionCoefficient(medium.state)	1/K	isobaric expansion coefficient
Medium.isothermalCompressibility(medium.state)	1/Pa	isothermal compressibility
Medium.density_derp_h(medium.state)	kg/(m3.Pa)	derivative of density by pressure at constant enthalpy
Medium.density_derh_p(medium.state)	kg2/(m3.J)	derivative of density by enthalpy at constant pressure
Medium.density_derp_T(medium.state)	kg/(m3.Pa)	derivative of density by pressure at constant temperature
Medium.density_derT_p(medium.state)	kg/(m3.K)	derivative of density by temperature at constant pressure
Medium.density_derX(medium.state)	kg/m3	derivative of density by mass fraction
Medium.molarMass(medium.state)	kg/mol	molar mass

More details are given in Modelica.Media.UsersGuide.MediumUsage.OptionalProperties. Many additional optional functions are defined to compute properties of saturated media, either liquid (bubble point) or vapour (dew point). The argument to such functions is a SaturationProperties record, which can be set starting from either the saturation pressure or the saturation temperature. With reference to a model defining a pressure p, a temperature T, and a SaturationProperties record sat, the following functions are provided:

Function call	Unit	Description
Medium.saturationPressure(T)	Pa	Saturation pressure at temperature T
Medium.saturationTemperature(p)	K	Saturation temperature at pressure p
Medium.saturationTemperature_derp(p)	K/Pa	Derivative of saturation temperature with respect to pressure
Medium.bubbleEnthalpy(sat)	J/kg	Specific enthalpy at bubble point
Medium.dewEnthalpy(sat)	J/kg	Specific enthalpy at dew point
Medium.bubbleEntropy(sat)	J/(kg.K)	Specific entropy at bubble point
Medium.dewEntropy(sat)	J/(kg.K)	Specific entropy at dew point
Medium.bubbleDensity(sat)	kg/m3	Density at bubble point
Medium.dewDensity(sat)	kg/m3	Density at dew point
Medium.dBubbleDensity_dPressure(sat)	kg/(m3.Pa)	Derivative of density at bubble point with respect to pressure

Medium.dDewDensity_dPressure(sat)	kg/(m3.Pa)	Derivative of density at dew point with respect to pressure
Medium.dBubbleEnthalpy_dPressure(sat)	J/(kg.Pa)	Derivative of specific enthalpy at bubble point with respect to pressure
Medium.dDewEnthalpy_dPressure(sat)	J/(kg.Pa)	Derivative of specific enthalpy at dew point with respect to pressure
Medium.surfaceTension(sat)	N/m	Surface tension between liquid and vapour phase

Details on usage and some examples are given in: Modelica.Media.UsersGuide.MediumUsage.TwoPhase.

Many further properties can be computed. Using the well-known Bridgman's Tables, all first partial derivatives of the standard thermodynamic variables can be computed easily.

The documentation of the IAPWS/IF97 steam properties can be freely distributed with computer implementations and are included here (in directory Modelica\help\Documentation\IF97documentation):

- IF97.pdf The standards document for the main part of the IF97.
- Back3.pdf The backwards equations for region 3.
- crits.pdf The critical point data.
- meltsub.pdf The melting- and sublimation line formulation (not implemented)
- surf.pdf The surface tension standard definition
- thcond.pdf The thermal conductivity standard definition
- visc.pdf The viscosity standard definition

Package Content

Name	Description
waterConstants	
simpleWaterConstants	
ConstantPropertyLiquidWater	Water: Simple liquid water medium (incompressible, constant data)
IdealSteam	Water: Steam as ideal gas from NASA source
WaterIF97OnePhase_ph	Water using the IF97 standard, explicit in p and h, and only valid outside the two-phase dome
WaterIF97_ph	Water using the IF97 standard, explicit in p and h
WaterIF97_base	Water: Steam properties as defined by IAPWS/IF97 standard
WaterIF97_fixedregion	Water: Steam properties as defined by IAPWS/IF97 standard
WaterIF97_R1ph	region 1 (liquid) water according to IF97 standard
WaterIF97_R2ph	region 2 (steam) water according to IF97 standard
WaterIF97_R3ph	region 3 water according to IF97 standard
WaterIF97_R4ph	region 4 water according to IF97 standard
WaterIF97_R5ph	region 5 water according to IF97 standard
WaterIF97_R1pT	region 1 (liquid) water according to IF97 standard
WaterIF97_R2pT	region 2 (steam) water according to IF97 standard
IF97_Utilities	Low level and utility computation for high accuracy water properties according to the IAPWS/IF97 standard

Types and constants

```
constant Interfaces.PartialTwoPhaseMedium.FluidConstants[1] waterConstants(
   each chemicalFormula = "H2O",
   each structureFormula="H2O",
```

```
      each casRegistryNumber="7732-18-5",
      each iupacName="oxidane",
      each molarMass=0.018015268,
      each criticalTemperature=647.096,
      each criticalPressure=22064.0e3,
      each criticalMolarVolume=1/322.0*0.018015268,
      each normalBoilingPoint=373.124,
      each meltingPoint=273.15,
      each triplePointTemperature=273.16,
      each triplePointPressure=611.657,
      each acentricFactor = 0.344,
      each dipoleMoment = 1.8,
      each hasCriticalData=true);

  constant Interfaces.PartialMedium.FluidConstants[1] simpleWaterConstants(
      each chemicalFormula = "H2O",
      each structureFormula="H2O",
      each casRegistryNumber="7732-18-5",
      each iupacName="oxidane",
      each molarMass=0.018015268);

package StandardWater = WaterIF97_ph
  "Water using the IF97 standard, explicit in p and h. Recommended for most
applications";

package StandardWaterOnePhase = WaterIF97_pT
  "Water using the IF97 standard, explicit in p and T. Recommended for one-phase
applications";

package WaterIF97_pT
  "Water using the IF97 standard, explicit in p and T"
  extends WaterIF97_base(
    final ph_explicit=false,
    final dT_explicit=false,
    final pT_explicit=true,
    final smoothModel=true,
    final onePhase=true);
end WaterIF97_pT;
```

Modelica.Media.Water.ConstantPropertyLiquidWater

Water: Simple liquid water medium (incompressible, constant data)

Information

Modelica.Media.Water.IdealSteam

Water: Steam as ideal gas from NASA source

Information

Modelica.Media.Water.WaterIF97OnePhase_ph

Water using the IF97 standard, explicit in p and h, and only valid outside the two-phase dome

Information

Modelica.Media.Water.WaterIF97_ph

Water using the IF97 standard, explicit in p and h

Information

Modelica.Media.Water.WaterIF97_base

Water: Steam properties as defined by IAPWS/IF97 standard

Information

This model calculates medium properties for water in the **liquid**, **gas** and **two phase** regions according to the IAPWS/IF97 standard, i.e., the accepted industrial standard and best compromise between accuracy and computation time. For more details see Modelica.Media.Water.IF97_Utilities. Three variable pairs can be the independent variables of the model:

1. Pressure **p** and specific enthalpy **h** are the most natural choice for general applications. This is the recommended choice for most general purpose applications, in particular for power plants.
2. Pressure **p** and temperature **T** are the most natural choice for applications where water is always in the same phase, both for liquid water and steam.
3. Density **d** and temperature **T** are explicit variables of the Helmholtz function in the near-critical region and can be the best choice for applications with super-critical or near-critial states.

The following quantities are always computed:

Variable	Unit	Description
T	K	temperature
u	J/kg	specific internal energy
d	kg/m^3	density
p	Pa	pressure
h	J/kg	specific enthalpy

In some cases additional medium properties are needed. A component that needs these optional properties has to call one of the functions listed in Modelica.Media.UsersGuide.MediumUsage.OptionalProperties and in Modelica.Media.UsersGuide.MediumUsage.TwoPhase.

Many further properties can be computed. Using the well-known Bridgman's Tables, all first partial derivatives of the standard thermodynamic variables can be computed easily.

Package Content

Name	Description
ThermodynamicState	thermodynamic state
ph_explicit	true if explicit in pressure and specific enthalpy
dT_explicit	true if explicit in density and temperature
pT_explicit	true if explicit in pressure and temperature

BaseProperties	Base properties of water
density_ph	Computes density as a function of pressure and specific enthalpy
temperature_ph	Computes temperature as a function of pressure and specific enthalpy
temperature_ps	Compute temperature from pressure and specific enthalpy
density_ps	Computes density as a function of pressure and specific enthalpy
pressure_dT	Computes pressure as a function of density and temperature
specificEnthalpy_dT	Computes specific enthalpy as a function of density and temperature
specificEnthalpy_pT	Computes specific enthalpy as a function of pressure and temperature
specificEnthalpy_ps	Computes specific enthalpy as a function of pressure and temperature
density_pT	Computes density as a function of pressure and temperature
setDewState	set the thermodynamic state on the dew line
setBubbleState	set the thermodynamic state on the bubble line
dynamicViscosity	Dynamic viscosity of water
thermalConductivity	Thermal conductivity of water
surfaceTension	Surface tension in two phase region of water
pressure	return pressure of ideal gas
temperature	return temperature of ideal gas
density	return density of ideal gas
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificEntropy	specific entropy of water
specificHeatCapacityCp	specific heat capacity at constant pressure of water
specificHeatCapacityCv	specific heat capacity at constant volume of water
isentropicExponent	Return isentropic exponent
isothermalCompressibility	Isothermal compressibility of water
isobaricExpansionCoefficient	isobaric expansion coefficient of water
velocityOfSound	
isentropicEnthalpy	compute h(p,s)
density_derh_p	density derivative by specific enthalpy
density_derp_h	density derivative by pressure
bubbleEnthalpy	boiling curve specific enthalpy of water
dewEnthalpy	dew curve specific enthalpy of water

bubbleEntropy	boiling curve specific entropy of water
dewEntropy	dew curve specific entropy of water
bubbleDensity	boiling curve specific density of water
dewDensity	dew curve specific density of water
saturationTemperature	saturation temperature of water
saturationTemperature_derp	derivative of saturation temperature w.r.t. pressure
saturationPressure	saturation pressure of water
dBubbleDensity_dPressure	bubble point density derivative
dDewDensity_dPressure	dew point density derivative
dBubbleEnthalpy_dPressure	bubble point specific enthalpy derivative
dDewEnthalpy_dPressure	dew point specific enthalpy derivative
setState_dTX	
setState_phX	
setState_psX	
setState_pTX	
Inherited	
smoothModel	true if the (derived) model should not generate state events
onePhase	true if the (derived) model should never be called with two-phase inputs
FluidLimits	validity limits for fluid model
FluidConstants	extended fluid constants
fluidConstants	constant data for the fluid
SaturationProperties	Saturation properties of two phase medium
FixedPhase	phase of the fluid: 1 for 1-phase, 2 for two-phase, 0 for not known, e.g. interactive use
setSat_T	Return saturation property record from temperature
setSat_p	Return saturation property record from pressure
saturationPressure_sat	Return saturation temperature
saturationTemperature_sat	Return saturation temperature
saturationTemperature_derp_sat	Return derivative of saturation temperature w.r.t. pressure
molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from pressure, temperature and mass fraction
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p, s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
setState_pT	Return thermodynamic state from p and T

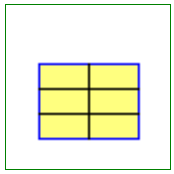
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
setState_px	Return thermodynamic state from pressure and vapour quality
setState_Tx	Return thermodynamic state from temperature and vapour quality
vapourQuality	Return vapour quality
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
density_pTX	Return density from p, T, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes

Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes
Choices	Types, constants to define menu choices

Types and constants

```
constant Boolean ph_explicit
"true if explicit in pressure and specific enthalpy";

constant Boolean dT_explicit "true if explicit in density and temperature";

constant Boolean pT_explicit "true if explicit in pressure and temperature";
```

Modelica.Media.Water.WaterIF97_base.ThermodynamicState

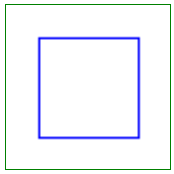


thermodynamic state

Modelica definition

```
redeclare record extends ThermodynamicState "thermodynamic state"
  SpecificEnthalpy h "specific enthalpy";
  Density d "density";
  Temperature T "temperature";
  AbsolutePressure p "pressure";
end ThermodynamicState;
```

Modelica.Media.Water.WaterIF97_base.BaseProperties



Base properties of water

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

---

Modelica.Media.Water.WaterIF97_base.density_ph

Computes density as a function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Water.WaterIF97_base.temperature_ph

Computes temperature as a function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description

AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Water.WaterIF97_base.temperature_ps

Compute temperature from pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Water.WaterIF97_base.density_ps

Computes density as a function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Water.WaterIF97_base.pressure_dT

Computes pressure as a function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		Temperature [K]

FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Water.WaterIF97_base.specificEnthalpy_dT

Computes specific enthalpy as a function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_base.specificEnthalpy_pT

Computes specific enthalpy as a function of pressure and temperature

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_base.specificEnthalpy_ps

Computes specific enthalpy as a function of pressure and temperature

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_base.density_pT

Computes density as a function of pressure and temperature

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Water.WaterIF97_base.setDewState

set the thermodynamic state on the dew line

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation point
FixedPhase	phase	1	phase: default is one phase

Outputs

Type	Name	Description
ThermodynamicState	state	complete thermodynamic state info

Modelica.Media.Water.WaterIF97_base.setBubbleState

set the thermodynamic state on the bubble line

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation point
FixedPhase	phase	1	phase: default is one phase

Outputs

Type	Name	Description
ThermodynamicState	state	complete thermodynamic state info

Modelica.Media.Water.WaterIF97_base.dynamicViscosity

Dynamic viscosity of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Water.WaterIF97_base.thermalConductivity

Thermal conductivity of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Water.WaterIF97_base.surfaceTension

Surface tension in two phase region of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SurfaceTension	sigma	Surface tension sigma in the two phase region [N/m]

Modelica.Media.Water.WaterIF97_base.pressure

return pressure of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Water.WaterIF97_base.temperature

return temperature of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Water.WaterIF97_base.density

return density of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Water.WaterIF97_base.specificEnthalpy

Return specific enthalpy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_base.specificInternalEnergy

Return specific internal energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	u	Specific internal energy [J/kg]

Modelica.Media.Water.WaterIF97_base.specificGibbsEnergy

Return specific Gibbs energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	g	Specific Gibbs energy [J/kg]

Modelica.Media.Water.WaterIF97_base.specificHelmholtzEnergy

Return specific Helmholtz energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	f	Specific Helmholtz energy [J/kg]

Modelica.Media.Water.WaterIF97_base.specificEntropy

specific entropy of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.Water.WaterIF97_base.specificHeatCapacityCp

specific heat capacity at constant pressure of water

Information

In the two phase region this function returns the interpolated heat capacity between the liquid and vapour state heat capacities.

Error:Found no end-tag in HTML-documentation

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Water.WaterIF97_base.specificHeatCapacityCv

specific heat capacity at constant volume of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Water.WaterIF97_base.isentropicExponent

Return isentropic exponent

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.Water.WaterIF97_base.isothermalCompressibility

Isothermal compressibility of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	Isothermal compressibility [1/Pa]

Modelica.Media.Water.WaterIF97_base.isobaricExpansionCoefficient

isobaric expansion coefficient of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsobaricExpansionCoefficient	beta	Isobaric expansion coefficient [1/K]

Modelica.Media.Water.WaterIF97_base.velocityOfSound

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.Water.WaterIF97_base.isentropicEnthalpy

compute h(p,s)

Inputs

Type	Name	Default	Description
AbsolutePressure	p_downstream		downstream pressure [Pa]
ThermodynamicState	refState		reference state for entropy

Outputs

Type	Name	Description
SpecificEnthalpy	h_is	Isentropic enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_base.density_derh_p

density derivative by specific enthalpy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByEnthalpy	ddhp	Density derivative wrt specific enthalpy [kg.s2/m5]

Modelica.Media.Water.WaterIF97_base.density_derp_h

density derivative by pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByPressure	ddph	Density derivative wrt pressure [s2/m2]

Modelica.Media.Water.WaterIF97_base.bubbleEnthalpy

boiling curve specific enthalpy of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEnthalpy	hl	boiling curve specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_base.dewEnthalpy

dew curve specific enthalpy of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEnthalpy	hv	dew curve specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_base.bubbleEntropy

boiling curve specific entropy of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEntropy	sl	boiling curve specific entropy [J/(kg.K)]

Modelica.Media.Water.WaterIF97_base.dewEntropy

dew curve specific entropy of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEntropy	sv	dew curve specific entropy [J/(kg.K)]

Modelica.Media.Water.WaterIF97_base.bubbleDensity

boiling curve specific density of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
Density	dl	boiling curve density [kg/m3]

Modelica.Media.Water.WaterIF97_base.dewDensity

dew curve specific density of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
Density	dv	dew curve density [kg/m3]

Modelica.Media.Water.WaterIF97_base.saturationTemperature

saturation temperature of water

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]

Outputs

Type	Name	Description
Temperature	T	saturation temperature [K]

Modelica.Media.Water.WaterIF97_base.saturationTemperature_derp

derivative of saturation temperature w.r.t. pressure

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]

Outputs

Type	Name	Description
Real	dTp	derivative of saturation temperature w.r.t. pressure

Modelica.Media.Water.WaterIF97_base.saturationPressure

saturation pressure of water

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
AbsolutePressure	p	saturation pressure [Pa]

Modelica.Media.Water.WaterIF97_base.dBubbleDensity_dPressure

bubble point density derivative

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerDensityByPressure	ddldp	boiling curve density derivative [s2/m2]

Modelica.Media.Water.WaterIF97_base.dDewDensity_dPressure

dew point density derivative

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerDensityByPressure	ddvdp	saturated steam density derivative [s2/m2]

Modelica.Media.Water.WaterIF97_base.dBubbleEnthalpy_dPressure

bubble point specific enthalpy derivative

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerEnthalpyByPressure	dhldp	boiling curve specific enthalpy derivative [J.m.s2/kg2]

Modelica.Media.Water.WaterIF97_base.dDewEnthalpy_dPressure

dew point specific enthalpy derivative

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerEnthalpyByPressure	dhvdp	saturated steam specific enthalpy derivative [J.m.s2/kg2]

Modelica.Media.Water.WaterIF97_base.setState_dTX

Inputs

Type	Name	Default	Description
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Water.WaterIF97_base.setState_phX

Inputs

Type	Name	Default	Description
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Water.WaterIF97_base.setState_psX

Inputs

Type	Name	Default	Description
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]

MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Water.WaterIF97_base.setState_pTX

Inputs

Type	Name	Default	Description
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Water.WaterIF97_fixedregion

Water: Steam properties as defined by IAPWS/IF97 standard

Information

This model calculates medium properties for water in the **liquid**, **gas** and **two phase** regions according to the IAPWS/IF97 standard, i.e., the accepted industrial standard and best compromise between accuracy and computation time. For more details see Modelica.Media.Water.IF97_Utilities. Three variable pairs can be the independent variables of the model:

1. Pressure **p** and specific enthalpy **h** are the most natural choice for general applications. This is the recommended choice for most general purpose applications, in particular for power plants.
2. Pressure **p** and temperature **T** are the most natural choice for applications where water is always in the same phase, both for liquid water and steam.
3. Density **d** and temperature **T** are explicit variables of the Helmholtz function in the near-critical region and can be the best choice for applications with super-critical or near-critial states.

The following quantities are always computed:

Variable	Unit	Description
T	K	temperature
u	J/kg	specific internal energy
d	kg/m^3	density
p	Pa	pressure
h	J/kg	specific enthalpy

In some cases additional medium properties are needed. A component that needs these optional properties has to call one of the functions listed in Modelica.Media.UsersGuide.MediumUsage.OptionalProperties and in Modelica.Media.UsersGuide.MediumUsage.TwoPhase.

Many further properties can be computed. Using the well-known Bridgman's Tables, all first partial derivatives

of the standard thermodynamic variables can be computed easily.

Package Content

Name	Description
ThermodynamicState	thermodynamic state
Region	region of IF97, if known
ph_explicit	true if explicit in pressure and specific enthalpy
dT_explicit	true if explicit in density and temperature
pT_explicit	true if explicit in pressure and temperature
BaseProperties	Base properties of water
density_ph	Computes density as a function of pressure and specific enthalpy
temperature_ph	Computes temperature as a function of pressure and specific enthalpy
temperature_ps	Compute temperature from pressure and specific enthalpy
density_ps	Computes density as a function of pressure and specific enthalpy
pressure_dT	Computes pressure as a function of density and temperature
specificEnthalpy_dT	Computes specific enthalpy as a function of density and temperature
specificEnthalpy_pT	Computes specific enthalpy as a function of pressure and temperature
specificEnthalpy_ps	Computes specific enthalpy as a function of pressure and temperature
density_pT	Computes density as a function of pressure and temperature
setDewState	set the thermodynamic state on the dew line
setBubbleState	set the thermodynamic state on the bubble line
dynamicViscosity	Dynamic viscosity of water
thermalConductivity	Thermal conductivity of water
surfaceTension	Surface tension in two phase region of water
pressure	return pressure of ideal gas
temperature	return temperature of ideal gas
density	return density of ideal gas
specificEnthalpy	Return specific enthalpy
specificInternalEnergy	Return specific internal energy
specificGibbsEnergy	Return specific Gibbs energy
specificHelmholtzEnergy	Return specific Helmholtz energy
specificEntropy	specific entropy of water
specificHeatCapacityCp	specific heat capacity at constant pressure of water
specificHeatCapacityCv	specific heat capacity at constant volume of water
isentropicExponent	Return isentropic exponent

isothermalCompressibility	Isothermal compressibility of water
isobaricExpansionCoefficient	isobaric expansion coefficient of water
velocityOfSound	
isentropicEnthalpy	compute h(s,p)
density_derh_p	density derivative by specific enthalpy
density_derp_h	density derivative by pressure
bubbleEnthalpy	boiling curve specific enthalpy of water
dewEnthalpy	dew curve specific enthalpy of water
bubbleEntropy	boiling curve specific entropy of water
dewEntropy	dew curve specific entropy of water
bubbleDensity	boiling curve specific density of water
dewDensity	dew curve specific density of water
saturationTemperature	saturation temperature of water
saturationTemperature_derp	derivative of saturation temperature w.r.t. pressure
saturationPressure	saturation pressure of water
dBubbleDensity_dPressure	bubble point density derivative
dDewDensity_dPressure	dew point density derivative
dBubbleEnthalpy_dPressure	bubble point specific enthalpy derivative
dDewEnthalpy_dPressure	dew point specific enthalpy derivative
setState_dTX	
setState_phX	
setState_psX	
setState_pTX	
Inherited	
smoothModel	true if the (derived) model should not generate state events
onePhase	true if the (derived) model should never be called with two-phase inputs
FluidLimits	validity limits for fluid model
FluidConstants	extended fluid constants
fluidConstants	constant data for the fluid
SaturationProperties	Saturation properties of two phase medium
FixedPhase	phase of the fluid: 1 for 1-phase, 2 for two-phase, 0 for not known, e.g. interactive use
setSat_T	Return saturation property record from temperature
setSat_p	Return saturation property record from pressure
saturationPressure_sat	Return saturation temperature
saturationTemperature_sat	Return saturation temperature
saturationTemperature_derp_sat	Return derivative of saturation temperature w.r.t. pressure

molarMass	Return the molar mass of the medium
specificEnthalpy_pTX	Return specific enthalpy from pressure, temperature and mass fraction
temperature_phX	Return temperature from p, h, and X or Xi
density_phX	Return density from p, h, and X or Xi
temperature_psX	Return temperature from p, s, and X or Xi
density_psX	Return density from p, s, and X or Xi
specificEnthalpy_psX	Return specific enthalpy from p, s, and X or Xi
setState_pT	Return thermodynamic state from p and T
setState_ph	Return thermodynamic state from p and h
setState_ps	Return thermodynamic state from p and s
setState_dT	Return thermodynamic state from d and T
setState_px	Return thermodynamic state from pressure and vapour quality
setState_Tx	Return thermodynamic state from temperature and vapour quality
vapourQuality	Return vapour quality
mediumName="unusablePartialMedium"	Name of the medium
substanceNames={mediumName}	Names of the mixture substances. Set substanceNames={mediumName} if only one substance.
extraPropertiesNames=fill("", 0)	Names of the additional (extra) transported properties. Set extraPropertiesNames=fill("",0) if unused
singleState	= true, if u and d are not a function of pressure
reducedX=true	= true if medium contains the equation sum(X) = 1.0; set reducedX=true if only one substance (see docu for details)
fixedX=false	= true if medium contains the equation X = reference_X
reference_p=101325	Reference pressure of Medium: default 1 atmosphere
reference_T=298.15	Reference temperature of Medium: default 25 deg Celsius
reference_X=if nX == 0 then fill(0, nX) else fill(1/nX, nX)	Default mass fractions of medium
p_default=101325	Default value for pressure of medium (for initialization)
T_default=Modelica.SIunits.Conversions.from_degC(20)	Default value for temperature of medium (for initialization)
h_default=specificEnthalpy_pTX(p_default, T_default, X_default)	Default value for specific enthalpy of medium (for initialization)
X_default=reference_X	Default value for mass fractions of medium (for initialization)
nS=size(substanceNames, 1)	Number of substances
nX=if nS == 1 then 0 else nS	Number of mass fractions (= 0, if only one substance)
nXi=if fixedX then 0 else if reducedX then nS - 1 else nX	Number of structurally independent mass fractions (see docu for details)
nC=size(extraPropertiesNames, 1)	Number of extra (outside of standard mass-balance) transported properties
BasePropertiesRecord	Variables contained in every instance of BaseProperties
prandtlNumber	Return the Prandtl number
heatCapacity_cp	alias for deprecated name

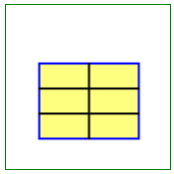
heatCapacity_cv	alias for deprecated name
beta	alias for isobaricExpansionCoefficient for user convenience
kappa	alias of isothermalCompressibility for user convenience
density_derp_T	Return density derivative wrt pressure at const temperature
density_derT_p	Return density derivative wrt temperature at constant pressure
density_derX	Return density derivative wrt mass fraction
density_pTX	Return density from p, T, and X or Xi
AbsolutePressure	Type for absolute pressure with medium specific attributes
Density	Type for density with medium specific attributes
DynamicViscosity	Type for dynamic viscosity with medium specific attributes
EnthalpyFlowRate	Type for enthalpy flow rate with medium specific attributes
MassFlowRate	Type for mass flow rate with medium specific attributes
MassFraction	Type for mass fraction with medium specific attributes
MoleFraction	Type for mole fraction with medium specific attributes
MolarMass	Type for molar mass with medium specific attributes
MolarVolume	Type for molar volume with medium specific attributes
IsentropicExponent	Type for isentropic exponent with medium specific attributes
SpecificEnergy	Type for specific energy with medium specific attributes
SpecificInternalEnergy	Type for specific internal energy with medium specific attributes
SpecificEnthalpy	Type for specific enthalpy with medium specific attributes
SpecificEntropy	Type for specific entropy with medium specific attributes
SpecificHeatCapacity	Type for specific heat capacity with medium specific attributes
SurfaceTension	Type for surface tension with medium specific attributes
Temperature	Type for temperature with medium specific attributes
ThermalConductivity	Type for thermal conductivity with medium specific attributes
PrandtlNumber	Type for Prandtl number with medium specific attributes
VelocityOfSound	Type for velocity of sound with medium specific attributes
ExtraProperty	Type for unspecified, mass-specific property transported by flow
CumulativeExtraProperty	Type for conserved integral of unspecified, mass specific property
ExtraPropertyFlowRate	Type for flow rate of unspecified, mass-specific property
IsobaricExpansionCoefficient	Type for isobaric expansion coefficient with medium specific attributes
DipoleMoment	Type for dipole moment with medium specific attributes
DerDensityByPressure	Type for partial derivative of density with resect to pressure with medium specific attributes
DerDensityByEnthalpy	Type for partial derivative of density with resect to enthalpy with medium specific attributes
DerEnthalpyByPressure	Type for partial derivative of enthalpy with resect to pressure with medium specific attributes
DerDensityByTemperature	Type for partial derivative of density with resect to temperature with medium specific attributes

Choices	Types, constants to define menu choices

Types and constants

```
constant Integer Region "region of IF97, if known";

constant Boolean ph_explicit
"true if explicit in pressure and specific enthalpy";

constant Boolean dT_explicit "true if explicit in density and temperature";

constant Boolean pT_explicit "true if explicit in pressure and temperature";
```

Modelica.Media.Water.WaterIF97_fixedregion.ThermodynamicState

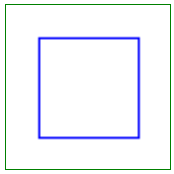


thermodynamic state

Modelica definition

```
redeclare record extends ThermodynamicState "thermodynamic state"
  SpecificEnthalpy h "specific enthalpy";
  Density d "density";
  Temperature T "temperature";
  AbsolutePressure p "pressure";
end ThermodynamicState;
```

Modelica.Media.Water.WaterIF97_fixedregion.BaseProperties



Base properties of water

Parameters

Type	Name	Default	Description
Boolean	standardOrderComponents	true	if true, last element in components is computed from 1-sum(Xi)
Advanced			
Boolean	preferredMediumStates	false	= true if StateSelect.prefer shall be used for the independent property variables of the medium

---

Modelica.Media.Water.WaterIF97_fixedregion.density_ph

Computes density as a function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]

FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Water.WaterIF97_fixedregion.temperature_ph

Computes temperature as a function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Water.WaterIF97_fixedregion.temperature_ps

Compute temperature from pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Water.WaterIF97_fixedregion.density_ps

Computes density as a function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]

SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Water.WaterIF97_fixedregion.pressure_dT

Computes pressure as a function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Water.WaterIF97_fixedregion.specificEnthalpy_dT

Computes specific enthalpy as a function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.specificEnthalpy_pT

Computes specific enthalpy as a function of pressure and temperature

Inputs

Type	Name	Default	Description

AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.specificEnthalpy_ps

Computes specific enthalpy as a function of pressure and temperature

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.density_pT

Computes density as a function of pressure and temperature

Inputs

Type	Name	Default	Description
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Water.WaterIF97_fixedregion.setDewState

set the thermodynamic state on the dew line

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation point
FixedPhase	phase	1	phase: default is one phase

Outputs

Type	Name	Description
ThermodynamicState	state	complete thermodynamic state info

Modelica.Media.Water.WaterIF97_fixedregion.setBubbleState

set the thermodynamic state on the bubble line

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation point
FixedPhase	phase	1	phase: default is one phase

Outputs

Type	Name	Description
ThermodynamicState	state	complete thermodynamic state info

Modelica.Media.Water.WaterIF97_fixedregion.dynamicViscosity

Dynamic viscosity of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DynamicViscosity	eta	Dynamic viscosity [Pa.s]

Modelica.Media.Water.WaterIF97_fixedregion.thermalConductivity

Thermal conductivity of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
ThermalConductivity	lambda	Thermal conductivity [W/(m.K)]

Modelica.Media.Water.WaterIF97_fixedregion.surfaceTension

Surface tension in two phase region of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SurfaceTension	sigma	Surface tension sigma in the two phase region [N/m]

Modelica.Media.Water.WaterIF97_fixedregion.pressure

return pressure of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
AbsolutePressure	p	Pressure [Pa]

Modelica.Media.Water.WaterIF97_fixedregion.temperature

return temperature of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Water.WaterIF97_fixedregion.density

return density of ideal gas

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
Density	d	Density [kg/m3]

Modelica.Media.Water.WaterIF97_fixedregion.specificEnthalpy

Return specific enthalpy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnthalpy	h	Specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.specificInternalEnergy

Return specific internal energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	u	Specific internal energy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.specificGibbsEnergy

Return specific Gibbs energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	g	Specific Gibbs energy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.specificHelmholtzEnergy

Return specific Helmholtz energy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEnergy	f	Specific Helmholtz energy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.specificEntropy

specific entropy of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificEntropy	s	Specific entropy [J/(kg.K)]

Modelica.Media.Water.WaterIF97_fixedregion.specificHeatCapacityCp

specific heat capacity at constant pressure of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	Specific heat capacity at constant pressure [J/(kg.K)]

Modelica.Media.Water.WaterIF97_fixedregion.specificHeatCapacityCv

specific heat capacity at constant volume of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	Specific heat capacity at constant volume [J/(kg.K)]

Modelica.Media.Water.WaterIF97_fixedregion.isentropicExponent

Return isentropic exponent

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsentropicExponent	gamma	Isentropic exponent [1]

Modelica.Media.Water.WaterIF97_fixedregion.isothermalCompressibility

Isothermal compressibility of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	Isothermal compressibility [1/Pa]

Modelica.Media.Water.WaterIF97_fixedregion.isobaricExpansionCoefficient

isobaric expansion coefficient of water

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
IsobaricExpansionCoefficient	beta	Isobaric expansion coefficient [1/K]

Modelica.Media.Water.WaterIF97_fixedregion.velocityOfSound

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
VelocityOfSound	a	Velocity of sound [m/s]

Modelica.Media.Water.WaterIF97_fixedregion.isentropicEnthalpy

compute h(s,p)

Inputs

Type	Name	Default	Description
AbsolutePressure	p_downstream		downstream pressure [Pa]
ThermodynamicState	refState		reference state for entropy

Outputs

Type	Name	Description
SpecificEnthalpy	h_is	Isentropic enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.density_derh_p

density derivative by specific enthalpy

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByEnthalpy	ddhp	Density derivative wrt specific enthalpy [kg.s2/m5]

Modelica.Media.Water.WaterIF97_fixedregion.density_derp_h

density derivative by pressure

Inputs

Type	Name	Default	Description
ThermodynamicState	state		thermodynamic state record

Outputs

Type	Name	Description
DerDensityByPressure	ddph	Density derivative wrt pressure [s2/m2]

Modelica.Media.Water.WaterIF97_fixedregion.bubbleEnthalpy

boiling curve specific enthalpy of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEnthalpy	hl	boiling curve specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.dewEnthalpy

dew curve specific enthalpy of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEnthalpy	hv	dew curve specific enthalpy [J/kg]

Modelica.Media.Water.WaterIF97_fixedregion.bubbleEntropy

boiling curve specific entropy of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEntropy	sl	boiling curve specific entropy [J/(kg.K)]

Modelica.Media.Water.WaterIF97_fixedregion.dewEntropy

dew curve specific entropy of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
SpecificEntropy	sv	dew curve specific entropy [J/(kg.K)]

Modelica.Media.Water.WaterIF97_fixedregion.bubbleDensity

boiling curve specific density of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
Density	dl	boiling curve density [kg/m3]

Modelica.Media.Water.WaterIF97_fixedregion.dewDensity

dew curve specific density of water

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
Density	dv	dew curve density [kg/m3]

Modelica.Media.Water.WaterIF97_fixedregion.saturationTemperature

saturation temperature of water

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]

Outputs

Type	Name	Description
Temperature	T	saturation temperature [K]

Modelica.Media.Water.WaterIF97_fixedregion.saturationTemperature_derp

derivative of saturation temperature w.r.t. pressure

Inputs

Type	Name	Default	Description
AbsolutePressure	p		pressure [Pa]

Outputs

Type	Name	Description
Real	dTp	derivative of saturation temperature w.r.t. pressure

Modelica.Media.Water.WaterIF97_fixedregion.saturationPressure

saturation pressure of water

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
AbsolutePressure	p	saturation pressure [Pa]

Modelica.Media.Water.WaterIF97_fixedregion.dBubbleDensity_dPressure

bubble point density derivative

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerDensityByPressure	ddldp	boiling curve density derivative [s2/m2]

Modelica.Media.Water.WaterIF97_fixedregion.dDewDensity_dPressure

dew point density derivative

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerDensityByPressure	ddvdp	saturated steam density derivative [s2/m2]

Modelica.Media.Water.WaterIF97_fixedregion.dBubbleEnthalpy_dPressure

bubble point specific enthalpy derivative

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerEnthalpyByPressure	dhldp	boiling curve specific enthalpy derivative [J.m.s2/kg2]

Modelica.Media.Water.WaterIF97_fixedregion.dDewEnthalpy_dPressure

dew point specific enthalpy derivative

Inputs

Type	Name	Default	Description
SaturationProperties	sat		saturation property record

Outputs

Type	Name	Description
DerEnthalpyByPressure	dhvdp	saturated steam specific enthalpy derivative [J.m.s2/kg2]

Modelica.Media.Water.WaterIF97_fixedregion.setState_dTX

Inputs

Type	Name	Default	Description
Integer	region	0	if 0, region is unknown, otherwise known and this input
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Water.WaterIF97_fixedregion.setState_phX

Inputs

Type	Name	Default	Description
Integer	region	0	if 0, region is unknown, otherwise known and this input
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
SpecificEnthalpy	h		Specific enthalpy [J/kg]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Water.WaterIF97_fixedregion.setState_psX

Inputs

Type	Name	Default	Description
Integer	region	0	if 0, region is unknown, otherwise known and this input
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
SpecificEntropy	s		Specific entropy [J/(kg.K)]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Water.WaterIF97_fixedregion.setState_pTX

Inputs

Type	Name	Default	Description
Integer	region	0	if 0, region is unknown, otherwise known and this input
FixedPhase	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
AbsolutePressure	p		Pressure [Pa]
Temperature	T		Temperature [K]
MassFraction	X[:]	reference_X	Mass fractions [kg/kg]

Outputs

Type	Name	Description
ThermodynamicState	state	thermodynamic state record

Modelica.Media.Water.WaterIF97_R1ph

region 1 (liquid) water according to IF97 standard

Information

Modelica.Media.Water.WaterIF97_R2ph

region 2 (steam) water according to IF97 standard

Information

Modelica.Media.Water.WaterIF97_R3ph

region 3 water according to IF97 standard

Information

Modelica.Media.Water.WaterIF97_R4ph

region 4 water according to IF97 standard

Information

Modelica.Media.Water.WaterIF97_R5ph

region 5 water according to IF97 standard

Information

Modelica.Media.Water.WaterIF97_R1pT

region 1 (liquid) water according to IF97 standard

Information

Modelica.Media.Water.WaterIF97_R2pT

region 2 (steam) water according to IF97 standard

Information

Modelica.Media.Water.IF97_Utilities

Low level and utility computation for high accuracy water properties according to the IAPWS/IF97 standard

Information

Package description:

This package provides high accuracy physical properties for water according to the IAPWS/IF97 standard. It has been part of the ThermoFluid Modelica library and been extended, reorganized and documented to become part of the Modelica Standard library.

An important feature that distinguishes this implementation of the IF97 steam property standard is that this implementation has been explicitly designed to work well in dynamic simulations. Computational performance has been of high importance. This means that there often exist several ways to get the same result from different functions if one of the functions is called often but can be optimized for that purpose.

The original documentation of the IAPWS/IF97 steam properties can freely be distributed with computer implementations, so for curious minds the complete standard documentation is provided with the Modelica properties library. The following documents are included (in directory Modelica\help\Documentation\IF97documentation):

- IF97.pdf The standards document for the main part of the IF97.
- Back3.pdf The backwards equations for region 3.
- crits.pdf The critical point data.
- meltsub.pdf The melting- and sublimation line formulation (in IF97_Utilities.BaseIF97.IceBoundaries)
- surf.pdf The surface tension standard definition
- thcond.pdf The thermal conductivity standard definition
- visc.pdf The viscosity standard definition

Package contents

- Package **BaseIF97** contains the implementation of the IAPWS-IF97 as described in IF97.pdf. The explicit backwards equations for region 3 from Back3.pdf are implemented as initial values for an inverse iteration of the exact function in IF97 for the input pairs (p,h) and (p,s). The low-level functions in BaseIF97 are not needed for standard simulation usage, but can be useful for experts and some special purposes.
- Function **water_ph** returns all properties needed for a dynamic control volume model and properties of general interest using pressure p and specific entropy enthalpy h as dynamic states in the record ThermoProperties_ph.
- Function **water_ps** returns all properties needed for a dynamic control volume model and properties of general interest using pressure p and specific entropy s as dynamic states in the record ThermoProperties_ps.
- Function **water_dT** returns all properties needed for a dynamic control volume model and properties of general interest using density d and temperature T as dynamic states in the record ThermoProperties_dT.
- Function **water_pT** returns all properties needed for a dynamic control volume model and properties of general interest using pressure p and temperature T as dynamic states in the record ThermoProperties_pT. Due to the coupling of pressure and temperature in the two-phase region, this model can obviously only be used for one-phase models or models treating both phases independently.
- Function **hl_p** computes the liquid specific enthalpy as a function of pressure. For overcritical pressures, the critical specific enthalpy is returned
- Function **hv_p** computes the vapour specific enthalpy as a function of pressure. For overcritical pressures, the critical specific enthalpy is returned
- Function **sl_p** computes the liquid specific entropy as a function of pressure. For overcritical pressures, the critical specific entropy is returned
- Function **sv_p** computes the vapour specific entropy as a function of pressure. For overcritical

pressures, the critical specific entropyis returned
- Function **rhol_T** computes the liquid density as a function of temperature. For overcritical temperatures, the critical density is returned
- Function **rhol_T** computes the vapour density as a function of temperature. For overcritical temperatures, the critical density is returned
- Function **dynamicViscosity** computes the dynamic viscosity as a function of density and temperature.
- Function **thermalConductivity** computes the thermal conductivity as a function of density, temperature and pressure. **Important note**: Obviously only two of the three inputs are really needed, but using three inputs speeds up the computation and the three variables are known in most models anyways. The inputs d,T and p have to be consistent.
- Function **surfaceTension** computes the surface tension between vapour and liquid water as a function of temperature.
- Function **isentropicEnthalpy** computes the specific enthalpy h(p,s,phase) in all regions. The phase input is needed due to discontinuous derivatives at the phase boundary.
- Function **dynamicIsentropicEnthalpy** computes the specific enthalpy h(p,s,,dguess,Tguess,phase) in all regions. The phase input is needed due to discontinuous derivatives at the phase boundary. Tguess and dguess are initial guess values for the density and temperature consistent with p and s. This function should be preferred in dynamic simulations where good guesses are often available.

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit for the ThermoFluid Library with help from Jonas Eborn and Falko Jens Wagner
- Code reorganization, enhanced documentation, additional functions: *December, 2002* by Hubertus Tummescheit and moved to Modelica properties library.

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

Package Content

Name	Description
BaseIF97	Modelica Physical Property Model: the new industrial formulation IAPWS-IF97
iter	
waterBaseProp_ph	intermediate property record for water
waterBaseProp_ps	intermediate property record for water
rho_props_ps	density as function of pressure and specific entropy
rho_ps	density as function of pressure and specific entropy
T_props_ps	temperature as function of pressure and specific entropy
T_ps	temperature as function of pressure and specific entropy
h_props_ps	specific enthalpy as function or pressure and temperature
h_ps	specific enthalpy as function or pressure and temperature
phase_ps	phase as a function of pressure and specific entropy
phase_ph	phase as a function of pressure and specific enthalpy
phase_dT	phase as a function of pressure and temperature

rho_props_ph	density as function of pressure and specific enthalpy
rho_ph	density as function of pressure and specific enthalpy
rho_ph_der	derivative function of rho_ph
T_props_ph	temperature as function of pressure and specific enthalpy
T_ph	temperature as function of pressure and specific enthalpy
T_ph_der	derivative function of T_ph
s_props_ph	specific entropy as function of pressure and specific enthalpy
s_ph	specific entropy as function of pressure and specific enthalpy
s_ph_der	specific entropy as function of pressure and specific enthalpy
cv_props_ph	specific heat capacity at constant volume as function of pressure and specific enthalpy
cv_ph	specific heat capacity at constant volume as function of pressure and specific enthalpy
regionAssertReal	assert function for inlining
cp_props_ph	specific heat capacity at constant pressure as function of pressure and specific enthalpy
cp_ph	specific heat capacity at constant pressure as function of pressure and specific enthalpy
beta_props_ph	isobaric expansion coefficient as function of pressure and specific enthalpy
beta_ph	isobaric expansion coefficient as function of pressure and specific enthalpy
kappa_props_ph	isothermal compressibility factor as function of pressure and specific enthalpy
kappa_ph	isothermal compressibility factor as function of pressure and specific enthalpy
velocityOfSound_props_ph	speed of sound as function of pressure and specific enthalpy
velocityOfSound_ph	
isentropicExponent_props_ph	isentropic exponent as function of pressure and specific enthalpy
isentropicExponent_ph	isentropic exponent as function of pressure and specific enthalpy
ddph_props	density derivative by pressure
ddph	density derivative by pressure
ddhp_props	density derivative by specific enthalpy
ddhp	density derivative by specific enthalpy
waterBaseProp_pT	intermediate property record for water (p and T prefered states)
rho_props_pT	density as function or pressure and temperature
rho_pT	density as function or pressure and temperature
h_props_pT	specific enthalpy as function or pressure and temperature
h_pT	specific enthalpy as function or pressure and temperature
h_pT_der	derivative function of h_pT
rho_pT_der	derivative function of rho_pT

s_props_pT	specific entropy as function of pressure and temperature
s_pT	temperature as function of pressure and temperature
cv_props_pT	specific heat capacity at constant volume as function of pressure and temperature
cv_pT	specific heat capacity at constant volume as function of pressure and temperature
cp_props_pT	specific heat capacity at constant pressure as function of pressure and temperature
cp_pT	specific heat capacity at constant pressure as function of pressure and temperature
beta_props_pT	isobaric expansion coefficient as function of pressure and temperature
beta_pT	isobaric expansion coefficient as function of pressure and temperature
kappa_props_pT	isothermal compressibility factor as function of pressure and temperature
kappa_pT	isothermal compressibility factor as function of pressure and temperature
velocityOfSound_props_pT	speed of sound as function of pressure and temperature
velocityOfSound_pT	speed of sound as function of pressure and temperature
isentropicExponent_props_pT	isentropic exponent as function of pressure and temperature
isentropicExponent_pT	isentropic exponent as function of pressure and temperature
waterBaseProp_dT	intermediate property record for water (d and T prefered states)
h_props_dT	specific enthalpy as function of density and temperature
h_dT	specific enthalpy as function of density and temperature
h_dT_der	derivative function of h_dT
p_props_dT	pressure as function of density and temperature
p_dT	pressure as function of density and temperature
p_dT_der	derivative function of p_dT
s_props_dT	specific entropy as function of density and temperature
s_dT	temperature as function of density and temperature
cv_props_dT	specific heat capacity at constant volume as function of density and temperature
cv_dT	specific heat capacity at constant volume as function of density and temperature
cp_props_dT	specific heat capacity at constant pressure as function of density and temperature
cp_dT	specific heat capacity at constant pressure as function of density and temperature
beta_props_dT	isobaric expansion coefficient as function of density and temperature
beta_dT	isobaric expansion coefficient as function of density and temperature
kappa_props_dT	isothermal compressibility factor as function of density and temperature
kappa_dT	isothermal compressibility factor as function of density and temperature
velocityOfSound_props_dT	speed of sound as function of density and temperature
velocityOfSound_dT	speed of sound as function of density and temperature

isentropicExponent_props_dT	isentropic exponent as function of density and temperature
isentropicExponent_dT	isentropic exponent as function of density and temperature
hl_p	compute the saturated liquid specific h(p)
hv_p	compute the saturated vapour specific h(p)
sl_p	compute the saturated liquid specific s(p)
sv_p	compute the saturated vapour specific s(p)
rhol_T	compute the saturated liquid d(T)
rhov_T	compute the saturated vapour d(T)
rhol_p	compute the saturated liquid d(p)
rhov_p	compute the saturated vapour d(p)
dynamicViscosity	compute eta(d,T) in the one-phase region
thermalConductivity	compute lambda(d,T,p) in the one-phase region
surfaceTension	compute sigma(T) at saturation T
isentropicEnthalpy	isentropic specific enthalpy from p,s (preferably use dynamicIsentropicEnthalpy in dynamic simulation!)
isentropicEnthalpy_props	
isentropicEnthalpy_der	derivative of isentropic specific enthalpy from p,s
dynamicIsentropicEnthalpy	isentropic specific enthalpy from p,s and good guesses of d and T

Modelica.Media.Water.IF97_Utilities.BaseIF97

Modelica Physical Property Model: the new industrial formulation IAPWS-IF97

Information

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit for the ThermoFluid Library with help from Jonas Eborn and Falko Jens Wagner
- Code reorganization, enhanced documentation, additional functions: *December, 2002* by Hubertus Tummescheit and moved to Modelica properties library.

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

In September 1997, the International Association for the Properties of Water and Steam (IAPWS) adopted a new formulation for the thermodynamic properties of water and steam for industrial use. This new industrial standard is called "IAPWS Industrial Formulation for the Thermodynamic Properties of Water and Steam" (IAPWS-IF97). The formulation IAPWS-IF97 replaces the previous industrial standard IFC-67.

Based on this new formulation, a new steam table, titled "Properties of Water and Steam" by W. Wagner and A. Kruse, was published by the Springer-Verlag, Berlin - New-York - Tokyo in April 1998. This steam table, ref. [1] is bilingual (English / German) and contains a complete description of the equations of IAPWS-IF97.

This reference is the authoritative source of information for this implementation. A mostly identical version has been published by the International Association for the Properties of Water and Steam (IAPWS) with permission granted to re-publish the information if credit is given to IAPWS. This document is distributed with this library as IF97.pdf. In addition, the equations published by IAPWS for the transport properties dynamic viscosity (standards document: visc.pdf) and thermal conductivity (standards document: thcond.pdf) and equations for the surface tension (standards document: surf.pdf) are also implemented in this library and included for reference.

The functions in BaseIF97.mo are low level functions which should only be used in those exceptions when the standard user level functions in Water.mo do not contain the wanted properties.

Based on IAPWS-IF97, Modelica functions are available for calculating the most common thermophysical properties (thermodynamic and transport properties). The implementation requires part of the common medium property infrastructure of the Modelica.Thermal.Properties library in the file Common.mo. There are a few extensions from the version of IF97 as documented in IF97.pdf in order to improve performance for dynamic simulations. Input variables for calculating the properties are only implemented for a limited number of variable pairs which make sense as dynamic states: (p,h), (p,T), (p,s) and (d,T).

---

1. Structure and Regions of IAPWS-IF97

The IAPWS Industrial Formulation 1997 consists of a set of equations for different regions which cover the following range of validity:

$$273{,}15\ \mathrm{K} < T < 1073{,}15\ \mathrm{K} \qquad p < 100\ \mathrm{MPa}$$

$$1073{,}15\ \mathrm{K} < T < 2273{,}15\ \mathrm{K} \qquad p < 10\ \mathrm{MPa}$$

Figure 1 shows the 5 regions into which the entire range of validity of IAPWS-IF97 is divided. The boundaries of the regions can be directly taken from Fig. 1 except for the boundary between regions 2 and 3; this boundary, which corresponds approximately to the isentropic line s = 5.047 kJ kg $^{-1}$ K^{-1}, is defined by a corresponding auxiliary equation. Both regions 1 and 2 are individually covered by a fundamental equation for the specific Gibbs free energy $g(p,T)$, region 3 by a fundamental equation for the specific Helmholtz free energy $f(\rho,T)$, and the saturation curve, corresponding to region 4, by a saturation-pressure equation $p_s(T)$. The high-temperature region 5 is also covered by a $g(p,T)$ equation. These 5 equations, shown in rectangular boxes in Fig. 1, form the so-called *basic equations*.

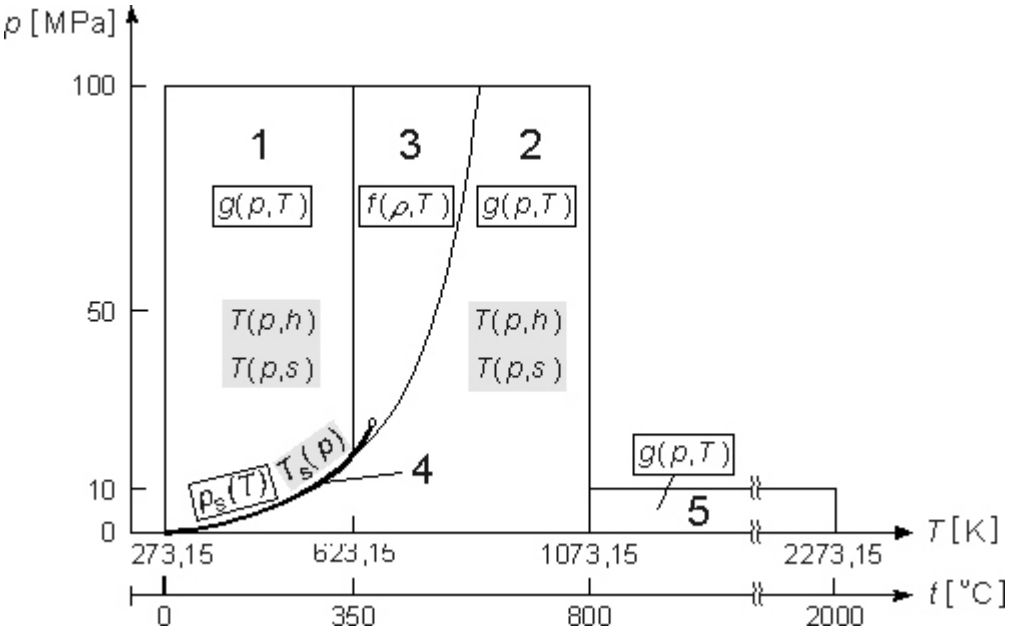


Figure 1: Regions and equations of IAPWS-IF97

In addition to these basic equations, so-called *backward equations* are provided for regions 1, 2, and 4 in form of T (p,h) and T (p,s) for regions 1 and 2, and T_s (p) for region 4. These backward equations, marked in grey in Fig. 1, were developed in such a way that they are numerically very consistent with the corresponding basic equation. Thus, properties as functions of p,h and of p,s for regions 1 and 2, and of p for region 4 can be calculated without any iteration. As a result of this special concept for the development of the new industrial standard IAPWS-IF97, the most important properties can be calculated extremely quickly. All modelica functions are optimized with regard to short computing times.

The complete description of the individual equations of the new industrial formulation IAPWS-IF97 is given in IF97.pdf. Comprehensive information on IAPWS-IF97 (requirements, concept, accuracy, consistency along region boundaries, and the increase of computing speed in comparison with IFC-67, etc.) can be taken from IF97.pdf or [2].

[1]*Wagner, W., Kruse, A.* Properties of Water and Steam / Zustandsgrößen von Wasser und Wasserdampf / IAPWS-IF97. Springer-Verlag, Berlin, 1998.

[2] *Wagner, W., Cooper, J. R., Dittmann, A., Kijima, J., Kretzschmar, H.-J., Kruse, A., Mareï¿½ R., Oguchi, K., Sato, H., Stöcker, I., ï¿½fner, O., Takaishi, Y., Tanishita, I., Trübenbach, J., and Willkommen, Th.* The IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam. ASME Journal of Engineering for Gas Turbines and Power 122 (2000), 150 - 182.

---

2. Calculable Properties

	Common name	Abbreviation	Unit
1	Pressure	p	Pa
2	Temperature	T	K
3	Density	d	kg/m^3
4	Specific volume	v	m^3/kg
5	Specific enthalpy	h	J/kg
6	Specific entropy	s	J/(kg K)
7	Specific internal energy	u	J/kg
8	Specific isobaric heat capacity	c_p	J/(kg K)
9	Specific isochoric heat capacity	c_v	J/(kg K)
10	Isentropic exponent, kappa= $-(v/p)\,(dp/dv)_s$	kappa (κ)	1
11	Speed of sound	a	m/s
12	Dryness fraction	x	kg/kg
13	Specific Helmholtz free energy, f = u - Ts	f	J/kg
14	Specific Gibbs free energy, g = h - Ts	g	J/kg
15	Isenthalpic exponent, theta = $-(v/p)(dp/dv)_h$	theta (θ)	1
16	Isobaric volume expansion coefficient, alpha = $v^{-1}\,(dv/dT)_p$	alpha (α)	1/K
17	Isochoric pressure coefficient, beta = $p^{-1}\,(dp/dT)_v$	beta (β)	1/K
18	Isothermal compressibility, gamma = $-v^{-1}(dv/dp)_T$	gamma (γ)	1/Pa
19	Dynamic viscosity	eta (η)	Pa s
20	Kinematic viscosity	nu (ν)	m^2/s
21	Thermal conductivity	lambda (λ)	W/(m K)
22	Surface tension	sigma (σ)	N/m

The properties 1-11 are calculated by default with the functions for dynamic simulation, 2 of these variables

are the dynamic states and are the inputs to calculate all other properties. In addition to these properties of general interest, the entries to the thermodynamic Jacobian matrix which render the mass- and energy balances explicit in the input variables to the property calculation are also calculated. For an explanatory example using pressure and specific enthalpy as states, see the Examples sub-package.

The high-level calls to steam properties are grouped into records comprising both the properties of general interest and the entries to the thermodynamic Jacobian. If additional properties are needed the low level functions in BaseIF97 provide more choice.

Additional functions

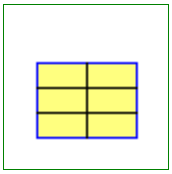
- Function **boundaryvals_p** computes the temperature and the specific enthalpy and entropy on both phase boundaries as a function of p
- Function **boundaryderivs_p** is the Modelica derivative function of **boundaryvals_p**
- Function **extraDerivs_ph** computes all entries to Bridgmans tables for all one-phase regions of IF97 using inputs (p,h). All 336 directional derivatives of the thermodynamic surface can be computed as a ratio of two entries in the return data, see package Common for details.
- Function **extraDerivs_pT** computes all entries to Bridgmans tables for all one-phase regions of IF97 using inputs (p,T).

Package Content

Name	Description
IterationData	constants for iterations internal to some functions
data	constant IF97 data and region limits
getTstar	get normalization temperature for region 1, 2 or 5
getpstar	get normalization pressure for region 1, 2 or 5
critical	critical point data
triple	triple point data
Regions	functions to find the current region for given pairs of input variables
Basic	Base functions as described in IAWPS/IF97
IceBoundaries	the melting line and sublimation line curves from IAPWS
Transport	transport properties for water according to IAPWS/IF97
Isentropic	functions for calculating the isentropic enthalpy from pressure p and specific entropy s
Inverses	efficient inverses for selected pairs of variables
ByRegion	simple explicit functions for one region only
TwoPhase	steam properties in the two-phase rgion and on the phase boundaries
extraDerivs_ph	function to calculate some extra thermophysical properties in regions 1, 2, 3 and 5 as f(p,h)
extraDerivs_pT	function to calculate some extra thermophysical properties in regions 1, 2, 3 and 5 as f(p,T)

Modelica.Media.Water.IF97_Utilities.BaseIF97.IterationData

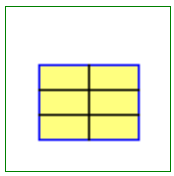
constants for iterations internal to some functions



Modelica definition

```
record IterationData
  "constants for iterations internal to some functions"

  extends Modelica.Icons.Record;
  constant Integer IMAX=50 "maximum number of iterations for inverse functions";
  constant Real DELP=1.0e-6 "maximum iteration error in pressure, Pa";
  constant Real DELS=1.0e-8
    "maximum iteration error in specific entropy, J/{kg.K}";
  constant Real DELH=1.0e-8
    "maximum iteration error in specific entthalpy, J/kg";
  constant Real DELD=1.0e-8 "maximum iteration error in density, kg/m^3";
end IterationData;
```

Modelica.Media.Water.IF97_Utilities.BaseIF97.data



constant IF97 data and region limits

Information

Record description

Constants needed in the international steam properties IF97. SCRIT and HCRIT are calculated from Helmholtz function for region 3.

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Documentation added: December 2002

Modelica definition

```
record data "constant IF97 data and region limits"
  extends Modelica.Icons.Record;
  constant SI.SpecificHeatCapacity RH2O=461.526
    "specific gas constant of water vapour";
  constant SI.MolarMass MH2O=0.01801528 "molar weight of water";
  constant SI.Temperature TSTAR1=1386.0
    "normalization temperature for region 1 IF97";
  constant SI.Pressure PSTAR1=16.53e6
    "normalization pressure for region 1 IF97";
  constant SI.Temperature TSTAR2=540.0
    "normalization temperature for region 2 IF97";
  constant SI.Pressure PSTAR2=1.0e6 "normalization pressure for region 2 IF97";
  constant SI.Temperature TSTAR5=1000.0
    "normalization temperature for region 5 IF97";
```

```
  constant SI.Pressure PSTAR5=1.0e6 "normalization pressure for region 5 IF97";
  constant SI.SpecificEnthalpy HSTAR1=2.5e6
    "normalization specific enthalpy for region 1 IF97";
  constant Real IPSTAR=1.0e-6
    "normalization pressure for inverse function in region 2 IF97";
  constant Real IHSTAR=5.0e-7
    "normalization specific enthalpy for inverse function in region 2 IF97";
  constant SI.Temperature TLIMIT1=623.15
    "temperature limit between regions 1 and 3";
  constant SI.Temperature TLIMIT2=1073.15
    "temperature limit between regions 2 and 5";
  constant SI.Temperature TLIMIT5=2273.15 "upper temperature limit of 5";
  constant SI.Pressure PLIMIT1=100.0e6
    "upper pressure limit for regions 1, 2 and 3";
  constant SI.Pressure PLIMIT4A=16.5292e6
    "pressure limit between regions 1 and 2, important for for two-phase (region
4)";
  constant SI.Pressure PLIMIT5=10.0e6
    "upper limit of valid pressure in region 5";
  constant SI.Pressure PCRIT=22064000.0 "the critical pressure";
  constant SI.Temperature TCRIT=647.096 "the critical temperature";
  constant SI.Density DCRIT=322.0 "the critical density";
  constant SI.SpecificEntropy SCRIT=4412.02148223476
    "the calculated specific entropy at the critical point";
  constant SI.SpecificEnthalpy HCRIT=2087546.84511715
    "the calculated specific enthalpy at the critical point";
  constant Real[5] n=array(0.34805185628969e3, -0.11671859879975e1,
      0.10192970039326e-2, 0.57254459862746e3, 0.13918839778870e2)
    "polynomial coefficients for boundary between regions 2 and 3";
end data;
```

Modelica.Media.Water.IF97_Utilities.BaseIF97.getTstar

get normalization temperature for region 1, 2 or 5

Inputs

Type	Name	Default	Description
Integer	region		IF 97 region

Outputs

Type	Name	Description
Temperature	Tstar	normalization temperature [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.getpstar

get normalization pressure for region 1, 2 or 5

Inputs

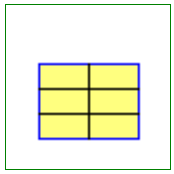
Type	Name	Default	Description

Integer	region		IF 97 region

Outputs

Type	Name	Description
Pressure	pstar	normalization pressure [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.critical



critical point data

Information

Record description

Critical point data for IF97 steam properties. SCRIT and HCRIT are calculated from helmholtz function for region 3

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

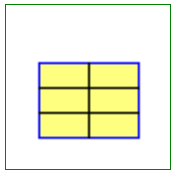
Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Documentation added: December 2002

Modelica definition

```
record critical "critical point data"
  extends Modelica.Icons.Record;
  constant SI.Pressure PCRIT=22064000.0 "the critical pressure";
  constant SI.Temperature TCRIT=647.096 "the critical temperature";
  constant SI.Density DCRIT=322.0 "the critical density";
  constant SI.SpecificEnthalpy HCRIT=2087546.84511715
    "the calculated specific enthalpy at the critical point";
  constant SI.SpecificEntropy SCRIT=4412.02148223476
    "the calculated specific entropy at the critical point";
end critical;
```

Modelica.Media.Water.IF97_Utilities.BaseIF97.triple



triple point data

Information

Record description

Vapour/liquid/ice triple point data for IF97 steam properties.

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Documentation added: December 2002

Modelica definition

```
record triple "triple point data"
  extends Modelica.Icons.Record;
  constant SI.Temperature Ttriple=273.16 "the triple point temperature";
  constant SI.Pressure ptriple=611.657 "the triple point temperature";
  constant SI.Density dltriple=999.792520031617642
    "the triple point liquid density";
  constant SI.Density dvtriple=0.485457572477861372e-2
    "the triple point vapour density";
end triple;
```

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions

functions to find the current region for given pairs of input variables

Information

Package description

Package **Regions** contains a large number of auxiliary functions which are neede to compute the current region of the IAPWS/IF97 for a given pair of input variables as quickly as possible. The focus of this implementation was on computational efficiency, not on compact code. Many of the function values calulated in these functions could be obtained using the fundamental functions of IAPWS/IF97, but with considerable overhead. If the region of IAPWS/IF97 is known in advance, the input variable mode can be set to the region, then the somewhat costly region checks are omitted. The checking for the phase has to be done outside the region functions because many properties are not differentiable at the region boundary. If the input phase is 2, the output region will be set to 4 immediately.

Package contents

The main 4 functions in this package are the functions returning the appropriate region for two input variables.

- Function **region_ph** compute the region of IAPWS/IF97 for input pair pressure and specific enthalpy.
- Function **region_ps** compute the region of IAPWS/IF97 for input pair pressure and specific entropy

- Function **region_dT** compute the region of IAPWS/IF97 for input pair density and temperature.
- Function **region_pT** compute the region of IAPWS/IF97 for input pair pressure and temperature (only ine phase region).

In addition, functions of the boiling and condensation curves compute the specific enthalpy, specific entropy, or density on these curves. The functions for the saturation pressure and temperature are included in the package **Basic** because they are part of the original IAPWS/IF97 standards document. These functions are also aliased to be used directly from package **Water**.

- Function **hl_p** computes the liquid specific enthalpy as a function of pressure. For overcritical pressures, the critical specific enthalpy is returned. An approximation is used for temperatures > 623.15 K.
- Function **hv_p** computes the vapour specific enthalpy as a function of pressure. For overcritical pressures, the critical specific enthalpy is returned. An approximation is used for temperatures > 623.15 K.
- Function **sl_p** computes the liquid specific entropy as a function of pressure. For overcritical pressures, the critical specific entropy is returned. An approximation is used for temperatures > 623.15 K.
- Function **sv_p** computes the vapour specific entropy as a function of pressure. For overcritical pressures, the critical specific entropyis returned. An approximation is used for temperatures > 623.15 K.
- Function **rhol_T** computes the liquid density as a function of temperature. For overcritical temperatures, the critical density is returned. An approximation is used for temperatures > 623.15 K.
- Function **rhol_T** computes the vapour density as a function of temperature. For overcritical temperatures, the critical density is returned. An approximation is used for temperatures > 623.15 K.

All other functions are auxiliary functions called from the region functions to check a specific boundary.

- Function **boundary23ofT** computes the boundary pressure between regions 2 and 3 (input temperature)
- Function **boundary23ofp** computes the boundary temperature between regions 2 and 3 (input pressure)
- Function **hlowerofp5** computes the lower specific enthalpy limit of region 5 (input p, T=1073.15 K)
- Function **hupperofp5** computes the upper specific enthalpy limit of region 5 (input p, T=2273.15 K)
- Function **slowerofp5** computes the lower specific entropy limit of region 5 (input p, T=1073.15 K)
- Function **supperofp5** computes the upper specific entropy limit of region 5 (input p, T=2273.15 K)
- Function **hlowerofp1** computes the lower specific enthalpy limit of region 1 (input p, T=273.15 K)
- Function **hupperofp1** computes the upper specific enthalpy limit of region 1 (input p, T=623.15 K)
- Function **slowerofp1** computes the lower specific entropy limit of region 1 (input p, T=273.15 K)
- Function **supperofp1** computes the upper specific entropy limit of region 1 (input p, T=623.15 K)
- Function **hlowerofp2** computes the lower specific enthalpy limit of region 2 (input p, T=623.15 K)
- Function **hupperofp2** computes the upper specific enthalpy limit of region 2 (input p, T=1073.15 K)
- Function **slowerofp2** computes the lower specific entropy limit of region 2 (input p, T=623.15 K)
- Function **supperofp2** computes the upper specific entropy limit of region 2 (input p, T=1073.15 K)
- Function **d1n** computes the density in region 1 as function of pressure and temperature
- Function **d2n** computes the density in region 2 as function of pressure and temperature
- Function **dhot1ofp** computes the hot density limit of region 1 (input p, T=623.15 K)
- Function **dupper1ofT**computes the high pressure density limit of region 1 (input T, p=100MPa)
- Function **hl_p_R4b** computes a high accuracy approximation to the liquid enthalpy for temperatures > 623.15 K (input p)
- Function **hv_p_R4b** computes a high accuracy approximation to the vapour enthalpy for temperatures > 623.15 K (input p)
- Function **sl_p_R4b** computes a high accuracy approximation to the liquid entropy for temperatures > 623.15 K (input p)
- Function **sv_p_R4b** computes a high accuracy approximation to the vapour entropy for temperatures > 623.15 K (input p)
- Function **rhol_p_R4b** computes a high accuracy approximation to the liquid density for temperatures > 623.15 K (input p)
- Function **rhov_p_R4b** computes a high accuracy approximation to the vapour density for temperatures > 623.15 K (input p)

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

Authors: Hubertus Tummescheit, Jonas Eborn and Falko Jens Wagner
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Revised and extended for inclusion in Modelica.Thermal: December 2002

Package Content

Name	Description
boundary23ofT	boundary function for region boundary between regions 2 and 3 (input temperature)
boundary23ofp	boundary function for region boundary between regions 2 and 3 (input pressure)
hlowerofp5	explicit lower specific enthalpy limit of region 5 as function of pressure
hupperofp5	explicit upper specific enthalpy limit of region 5 as function of pressure
slowerofp5	explicit lower specific entropy limit of region 5 as function of pressure
supperofp5	explicit upper specific entropy limit of region 5 as function of pressure
hlowerofp1	explicit lower specific enthalpy limit of region 1 as function of pressure
hupperofp1	explicit upper specific enthalpy limit of region 1 as function of pressure (meets region 4 saturation pressure curve at 623.15 K)
slowerofp1	explicit lower specific entropy limit of region 1 as function of pressure
supperofp1	explicit upper specific entropy limit of region 1 as function of pressure (meets region 4 saturation pressure curve at 623.15 K)
hlowerofp2	explicit lower specific enthalpy limit of region 2 as function of pressure (meets region 4 saturation pressure curve at 623.15 K)
hupperofp2	explicit upper specific enthalpy limit of region 2 as function of pressure
slowerofp2	explicit lower specific entropy limit of region 2 as function of pressure (meets region 4 saturation pressure curve at 623.15 K)
supperofp2	explicit upper specific entropy limit of region 2 as function of pressure
d1n	density in region 1 as function of p and T
d2n	density in region 2 as function of p and T
dhot1ofp	density at upper temperature limit of region 1
dupper1ofT	density at upper pressure limit of region 1
hl_p_R4b	explicit approximation of liquid specific enthalpy on the boundary between regions 4 and 3
hv_p_R4b	explicit approximation of vapour specific enthalpy on the boundary between regions 4 and 3
sl_p_R4b	explicit approximation of liquid specific entropy on the boundary between regions 4 and 3
sv_p_R4b	explicit approximation of vapour specific entropy on the boundary between regions 4 and 3
rhol_p_R4b	explicit approximation of liquid density on the boundary between regions 4 and 3

rhov_p_R4b	explicit approximation of vapour density on the boundary between regions 4 and 2
boilingcurve_p	properties on the boiling curve
dewcurve_p	properties on the dew curve
hvl_p	
hl_p	liquid specific enthalpy on the boundary between regions 4 and 3 or 1
hv_p	vapour specific enthalpy on the boundary between regions 4 and 3 or 2
hvl_p_der	derivative function for the specific enthalpy along the phase boundary
rhovl_p	
rhol_p	density of saturated water
rhov_p	density of saturated vapour
rhovl_p_der	
sl_p	liquid specific entropy on the boundary between regions 4 and 3 or 1
sv_p	vapour specific entropy on the boundary between regions 4 and 3 or 2
rhol_T	density of saturated water
rhov_T	density of saturated vapour
region_ph	return the current region (valid values: 1,2,3,4,5) in IF97 for given pressure and specific enthalpy
region_ps	return the current region (valid values: 1,2,3,4,5) in IF97 for given pressure and specific entropy
region_pT	return the current region (valid values: 1,2,3,5) in IF97, given pressure and temperature
region_dT	return the current region (valid values: 1,2,3,4,5) in IF97, given density and temperature
hvl_dp	derivative function for the specific enthalpy along the phase boundary
dhl_dp	derivative of liquid specific enthalpy on the boundary between regions 4 and 3 or 1 w.r.t pressure
dhv_dp	derivative of vapour specific enthalpy on the boundary between regions 4 and 3 or 1 w.r.t pressure
drhovl_dp	
drhol_dp	derivative of density of saturated water w.r.t. pressure
drhov_dp	derivative of density of saturated steam w.r.t. pressure

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.boundary23ofT

boundary function for region boundary between regions 2 and 3 (input temperature)

Inputs

Type	Name	Default	Description
Temperature	t		temperature (K) [K]

Outputs

Type	Name	Description
Pressure	p	pressure [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.boundary23ofp

boundary function for region boundary between regions 2 and 3 (input pressure)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
Temperature	t	temperature (K) [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hlowerofp5

explicit lower specific enthalpy limit of region 5 as function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hupperofp5

explicit upper specific enthalpy limit of region 5 as function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.slowerofp5

explicit lower specific entropy limit of region 5 as function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.supperofp5

explicit upper specific entropy limit of region 5 as function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hlowerofp1

explicit lower specific enthalpy limit of region 1 as function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hupperofp1

explicit upper specific enthalpy limit of region 1 as function of pressure (meets region 4 saturation pressure curve at 623.15 K)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.slowerofp1

explicit lower specific entropy limit of region 1 as function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.supperofp1

explicit upper specific entropy limit of region 1 as function of pressure (meets region 4 saturation pressure curve at 623.15 K)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hlowerofp2

explicit lower specific enthalpy limit of region 2 as function of pressure (meets region 4 saturation pressure curve at 623.15 K)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hupperofp2

explicit upper specific enthalpy limit of region 2 as function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description

SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.slowerofp2

explicit lower specific entropy limit of region 2 as function of pressure (meets region 4 saturation pressure curve at 623.15 K)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.supperofp2

explicit upper specific entropy limit of region 2 as function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.d1n

density in region 1 as function of p and T

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.d2n

density in region 2 as function of p and T

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dhot1ofp

density at upper temperature limit of region 1

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dupper1ofT

density at upper pressure limit of region 1

Inputs

Type	Name	Default	Description
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hl_p_R4b

explicit approximation of liquid specific enthalpy on the boundary between regions 4 and 3

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description

SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hv_p_R4b

explicit approximation of vapour specific enthalpy on the boundary between regions 4 and 3

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.sl_p_R4b

explicit approximation of liquid specific entropy on the boundary between regions 4 and 3

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.sv_p_R4b

explicit approximation of vapour specific entropy on the boundary between regions 4 and 3

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	s	[J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhol_p_R4b

explicit approximation of liquid density on the boundary between regions 4 and 3

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
Density	dl	liquid density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhov_p_R4b

explicit approximation of vapour density on the boundary between regions 4 and 2

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
Density	dv	vapour density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.boilingcurve_p

properties on the boiling curve

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
IF97PhaseBoundaryProperties	bpro	property record

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dewcurve_p

properties on the dew curve

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
IF97PhaseBoundaryProperties	bpro	property record

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hvl_p

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
IF97PhaseBoundaryProperties	bpro		property record

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hl_p

liquid specific enthalpy on the boundary between regions 4 and 3 or 1

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hv_p

vapour specific enthalpy on the boundary between regions 4 and 3 or 2

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hvl_p_der

derivative function for the specific enthalpy along the phase boundary

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

IF97PhaseBoundaryProperties	bpro		property record
Real	p_der		derivative of pressure

Outputs

Type	Name	Description
Real	h_der	time derivative of specific enthalpy along the phase boundary

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhovl_p

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
IF97PhaseBoundaryProperties	bpro		property record

Outputs

Type	Name	Description
Density	rho	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhol_p

density of saturated water

Inputs

Type	Name	Default	Description
Pressure	p		saturation pressure [Pa]

Outputs

Type	Name	Description
Density	rho	density of steam at the condensation point [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhov_p

density of saturated vapour

Inputs

Type	Name	Default	Description
Pressure	p		saturation pressure [Pa]

Outputs

Type	Name	Description
Density	rho	density of steam at the condensation point [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhovl_p_der

Inputs

Type	Name	Default	Description
Pressure	p		saturation pressure [Pa]
IF97PhaseBoundaryProperties	bpro		property record
Real	p_der		derivative of pressure

Outputs

Type	Name	Description
Real	d_der	time derivative of density along the phase boundary

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.sl_p

liquid specific entropy on the boundary between regions 4 and 3 or 1

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.sv_p

vapour specific entropy on the boundary between regions 4 and 3 or 2

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhol_T

density of saturated water

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
Density	d	density of water at the boiling point [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.rhov_T

density of saturated vapour

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
Density	d	density of steam at the condensation point [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.region_ph

return the current region (valid values: 1,2,3,4,5) in IF97 for given pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	phase: 2 for two-phase, 1 for one phase, 0 if not known
Integer	mode	0	mode: 0 means check, otherwise assume region=mode

Outputs

Type	Name	Description
Integer	region	region (valid values: 1,2,3,4,5) in IF97

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.region_ps

return the current region (valid values: 1,2,3,4,5) in IF97 for given pressure and specific entropy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Integer	phase	0	phase: 2 for two-phase, 1 for one phase, 0 if unknown
Integer	mode	0	mode: 0 means check, otherwise assume region=mode

Outputs

Type	Name	Description
Integer	region	region (valid values: 1,2,3,4,5) in IF97

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.region_pT

return the current region (valid values: 1,2,3,5) in IF97, given pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]
Integer	mode	0	mode: 0 means check, otherwise assume region=mode

Outputs

Type	Name	Description
Integer	region	region (valid values: 1,2,3,5) in IF97, region 4 is impossible!

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.region_dT

return the current region (valid values: 1,2,3,4,5) in IF97, given density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]
Integer	phase	0	phase: 2 for two-phase, 1 for one phase, 0 if not known
Integer	mode	0	mode: 0 means check, otherwise assume region=mode

Outputs

Type	Name	Description
Integer	region	(valid values: 1,2,3,4,5) in IF97

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.hvl_dp

derivative function for the specific enthalpy along the phase boundary

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
IF97PhaseBoundaryProperties	bpro		property record

Outputs

Type	Name	Description
Real	dh_dp	derivative of specific enthalpy along the phase boundary

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dhl_dp

derivative of liquid specific enthalpy on the boundary between regions 4 and 3 or 1 w.r.t pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
DerEnthalpyByPressure	dh_dp	specific enthalpy derivative w.r.t. pressure [J.m.s2/kg2]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.dhv_dp

derivative of vapour specific enthalpy on the boundary between regions 4 and 3 or 1 w.r.t pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
DerEnthalpyByPressure	dh_dp	specific enthalpy derivative w.r.t. pressure [J.m.s2/kg2]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.drhovl_dp

Inputs

Type	Name	Default	Description
Pressure	p		saturation pressure [Pa]
IF97PhaseBoundaryProperties	bpro		property record

Outputs

Type	Name	Description
Real	dd_dp	derivative of density along the phase boundary [kg/(m3.Pa)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.drhol_dp

derivative of density of saturated water w.r.t. pressure

Inputs

Type	Name	Default	Description
Pressure	p		saturation pressure [Pa]

Outputs

Type	Name	Description
DerDensityByPressure	dd_dp	derivative of density of water at the boiling point [s2/m2]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Regions.drhov_dp

derivative of density of saturated steam w.r.t. pressure

Inputs

Type	Name	Default	Description
Pressure	p		saturation pressure [Pa]

Outputs

Type	Name	Description
DerDensityByPressure	dd_dp	derivative of density of water at the boiling point [s2/m2]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic

Base functions as described in IAWPS/IF97

Information

Package description

Package BaseIF97/Basic computes the the fundamental functions for the 5 regions of the steam tables as described in the standards document IF97.pdf. The code of these functions has been generated using ***Mathematica*** and the add-on packages "Format" and "Optimize" to generate highly efficient, expression-optimized C-code from a symbolic representation of the thermodynamic functions. The C-code has than been transformed into Modelica code. An important feature of this optimization was to simultaneously optimize the functions and the directional derivatives because they share many common subexpressions.

Package contents

- Function **g1** computes the dimensionless Gibbs function for region 1 and all derivatives up to order 2 w.r.t pi and tau. Inputs: p and T.
- Function **g2** computes the dimensionless Gibbs function for region 2 and all derivatives up to order 2 w.r.t pi and tau. Inputs: p and T.
- Function **g2metastable** computes the dimensionless Gibbs function for metastable vapour (adjacent to region 2 but 2-phase at equilibrium) and all derivatives up to order 2 w.r.t pi and tau. Inputs: p and T.
- Function **f3** computes the dimensionless Helmholtz function for region 3 and all derivatives up to

order 2 w.r.t delta and tau. Inputs: d and T.
- Function **g5**computes the dimensionless Gibbs function for region 5 and all derivatives up to order 2 w.r.t pi and tau. Inputs: p and T.
- Function **tph1** computes the inverse function T(p,h) in region 1.
- Function **tph2** computes the inverse function T(p,h) in region 2.
- Function **tps2a** computes the inverse function T(p,s) in region 2a.
- Function **tps2b** computes the inverse function T(p,s) in region 2b.
- Function **tps2c** computes the inverse function T(p,s) in region 2c.
- Function **tps2** computes the inverse function T(p,s) in region 2.
- Function **tsat** computes the saturation temperature as a function of pressure.
- Function **dtsatofp** computes the derivative of the saturation temperature w.r.t. pressure as a function of pressure.
- Function **tsat_der** computes the Modelica derivative function of tsat.
- Function **psat** computes the saturation pressure as a function of temperature.
- Function **dptofT** computes the derivative of the saturation pressure w.r.t. temperature as a function of temperature.
- Function **psat_der** computes the Modelica derivative function of psat.

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Documentation added: December 2002

Package Content

Name	Description
g1	Gibbs function for region 1: g(p,T)
g2	Gibbs function for region 2: g(p,T)
g2metastable	Gibbs function for metastable part of region 2: g(p,T)
f3	Helmholtz function for region 3: f(d,T)
g5	base function for region 5: g(p,T)
gibbs	Gibbs function for region 1, 2 or 5: g(p,T,region)
g1pitau	derivative of g wrt pi and tau
g2pitau	derivative of g wrt pi and tau
g5pitau	derivative of g wrt pi and tau
f3deltatau	1st derivatives of f wrt delta and tau
tph1	inverse function for region 1: T(p,h)
tps1	inverse function for region 1: T(p,s)
tph2	reverse function for region 2: T(p,h)

tps2a	reverse function for region 2a: T(p,s)
tps2b	reverse function for region 2b: T(p,s)
tps2c	reverse function for region 2c: T(p,s)
tps2	reverse function for region 2: T(p,s)
tsat	region 4 saturation temperature as a function of pressure
dtsatofp	derivative of saturation temperature w.r.t. pressure
tsat_der	derivative function for tsat
psat	region 4 saturation pressure as a functionx of temperature
dptofT	derivative of pressure wrt temperature along the saturation pressure curve
psat_der	derivative function for psat
p1_hs	pressure as a function of ehtnalpy and entropy in region 1
h2ab_s	boundary between regions 2a and 2b
p2a_hs	pressure as a function of enthalpy and entropy in subregion 2a
p2b_hs	pressure as a function of enthalpy and entropy in subregion 2a
p2c_hs	pressure as a function of enthalpy and entropy in subregion 2c
h3ab_p	ergion 3 a b boundary for pressure/enthalpy
T3a_ph	Region 3 a: inverse function T(p,h)
T3b_ph	Region 3 b: inverse function T(p,h)
v3a_ph	Region 3 a: inverse function v(p,h)
v3b_ph	Region 3 b: inverse function v(p,h)
T3a_ps	Region 3 a: inverse function T(p,s)
T3b_ps	Region 3 b: inverse function T(p,s)
v3a_ps	Region 3 a: inverse function v(p,s)
v3b_ps	Region 3 b: inverse function v(p,s)

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g1

Gibbs function for region 1: g(p,T)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
GibbsDerivs	g	dimensionless Gibbs funcion and dervatives wrt pi and tau

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g2

Gibbs function for region 2: g(p,T)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
GibbsDerivs	g	dimensionless Gibbs funcion and dervatives wrt pi and tau

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g2metastable

Gibbs function for metastable part of region 2: g(p,T)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
GibbsDerivs	g	dimensionless Gibbs funcion and dervatives wrt pi and tau

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.f3

Helmholtz function for region 3: f(d,T)

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
HelmholtzDerivs	f	dimensionless Helmholtz function and dervatives wrt delta and tau

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g5

base function for region 5: g(p,T)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
GibbsDerivs	g	dimensionless Gibbs funcion and dervatives wrt pi and tau

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.gibbs

Gibbs function for region 1, 2 or 5: g(p,T,region)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]
Integer	region		IF97 region, 1, 2 or 5

Outputs

Type	Name	Description
Real	g	dimensionless Gibbs funcion

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g1pitau

derivative of g wrt pi and tau

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Real	pi	dimensionless pressure
Real	tau	dimensionless temperature
Real	gpi	dimensionless dervative of Gibbs function wrt pi
Real	gtau	dimensionless dervative of Gibbs function wrt tau

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g2pitau

derivative of g wrt pi and tau

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Real	pi	dimensionless pressure
Real	tau	dimensionless temperature
Real	gpi	dimensionless dervative of Gibbs function wrt pi
Real	gtau	dimensionless dervative of Gibbs function wrt tau

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.g5pitau

derivative of g wrt pi and tau

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Real	pi	dimensionless pressure
Real	tau	dimensionless temperature
Real	gpi	dimensionless dervative of Gibbs function wrt pi
Real	gtau	dimensionless dervative of Gibbs function wrt tau

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.f3deltatau

1st derivatives of f wrt delta and tau

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Real	delta	dimensionless density
Real	tau	dimensionless temperature
Real	fdelta	dimensionless dervative of Helmholtz function wrt delta
Real	ftau	dimensionless dervative of Helmholtz function wrt tau

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tph1

inverse function for region 1: T(p,h)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps1

inverse function for region 1: T(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tph2

reverse function for region 2: T(p,h)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps2a

reverse function for region 2a: T(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps2b

reverse function for region 2b: T(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps2c

reverse function for region 2c: T(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tps2

reverse function for region 2: T(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tsat

region 4 saturation temperature as a function of pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
Temperature	t_sat	temperature [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.dtsatofp

derivative of saturation temperature w.r.t. pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
Real	dtsat	derivative of T w.r.t. p

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.tsat_der

derivative function for tsat

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Real	der_p		pressure derivatrive

Outputs

Type	Name	Description
Real	der_tsat	temperature derivative

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.psat

region 4 saturation pressure as a functionx of temperature

Inputs

Type	Name	Default	Description
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Pressure	p_sat	pressure [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.dptofT

derivative of pressure wrt temperature along the saturation pressure curve

Inputs

Type	Name	Default	Description
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
Real	dpt	temperature derivative of pressure

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.psat_der

derivative function for psat

Inputs

Type	Name	Default	Description
Temperature	T		temperature (K) [K]
Real	der_T		temperature derivative

Outputs

Type	Name	Description
Real	der_psat	pressure

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.p1_hs

pressure as a function of ehtnalpy and entropy in region 1

Information

Equation number 1 from:
The International Association for the Properties of Water and Steam
Gaithersburg, Maryland, USA

September 2001
Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy p(h,s) to the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam

Inputs

Type	Name	Default	Description
SpecificEnthalpy	h		specific enthalpy [J/kg]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Pressure	p	Pressure [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.h2ab_s

boundary between regions 2a and 2b

Information

Equation number 2 from:
The International Association for the Properties of Water and Steam
Gaithersburg, Maryland, USA
September 2001
Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy p(h,s) to the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam

Inputs

Type	Name	Default	Description
SpecificEntropy	s		Entropy [J/(kg.K)]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.p2a_hs

pressure as a function of enthalpy and entropy in subregion 2a

Information

Equation number 3 from:
The International Association for the Properties of Water and Steam
Gaithersburg, Maryland, USA
September 2001
Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy p(h,s)

to the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam

Inputs

Type	Name	Default	Description
SpecificEnthalpy	h		specific enthalpy [J/kg]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Pressure	p	Pressure [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.p2b_hs

pressure as a function of enthalpy and entropy in subregion 2a

Information

Equation number 4 from:
The International Association for the Properties of Water and Steam
Gaithersburg, Maryland, USA
September 2001
Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy p(h,s) to the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam

Inputs

Type	Name	Default	Description
SpecificEnthalpy	h		specific enthalpy [J/kg]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Pressure	p	Pressure [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.p2c_hs

pressure as a function of enthalpy and entropy in subregion 2c

Information

Equation number 5 from:
The International Association for the Properties of Water and Steam
Gaithersburg, Maryland, USA
September 2001
Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy p(h,s) to the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam

Inputs

Type	Name	Default	Description
SpecificEnthalpy	h		specific enthalpy [J/kg]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Pressure	p	Pressure [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.h3ab_p

ergion 3 a b boundary for pressure/enthalpy

Information

Equation number 1 from:

[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s),
v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of
Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	Enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.T3a_ph

Region 3 a: inverse function T(p,h)

Information

Equation number 2 from:

[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s),
v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of
Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]

SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
Temp_K	T	Temperature [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.T3b_ph

Region 3 b: inverse function T(p,h)

Information

Equation number 3 from:

[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s),
v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of
Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
Temp_K	T	Temperature [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.v3a_ph

Region 3 a: inverse function v(p,h)

Information

Equation number 4 from:

[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s),
v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of
Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
Volume	v	specific volume [m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.v3b_ph

Region 3 b: inverse function v(p,h)

Information

Equation number 5 from:

[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s),
v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of
Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
Volume	v	specific volume [m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.T3a_ps

Region 3 a: inverse function T(p,s)

Information

Equation number 6 from:

[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s),
v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of
Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description

Temp_K	T	Temperature [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.T3b_ps

Region 3 b: inverse function T(p,s)

Information

Equation number 7 from:
[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s),
v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of
Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Temp_K	T	Temperature [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.v3a_ps

Region 3 a: inverse function v(p,s)

Information

Equation number 8 from:
[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s),
v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of
Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Volume	v	specific volume [m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Basic.v3b_ps

Region 3 b: inverse function v(p,s)

Information

Equation number 9 from:
[1] The international Association for the Properties of Water and Steam
Vejle, Denmark
August 2003
Supplementary Release on Backward Equations for the Fucnctions T(p,h), v(p,h) and T(p,s), v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam

Inputs

Type	Name	Default	Description
Pressure	p		Pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Volume	v	specific volume [m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries

the melting line and sublimation line curves from IAPWS

Information

The International Association for the Properties of Water and Steam
Milan, Italy
September 1993
Release on the Pressure along the Melting and the Sublimation Curves of Ordinary Water Substance

Package Content

Name	Description
pmIceI_T	Melting pressure of ice I (temperature range from 273.16 to 251.165 K)
pmIceIII_T	Melting pressure of ice III (temperature range from 251.165 to 256.164 K)
pmIceV_T	Melting pressure of ice V (temperature range from 256.164 to 273.31 K)
sublimationPressure_T	Sublimation pressure, valid from 190 to 273.16 K

Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries.pmIceI_T

Melting pressure of ice I (temperature range from 273.16 to 251.165 K)

Information

Equation 1 from:
The International Association for the Properties of Water and Steam

Milan, Italy
September 1993
Release on the Pressure along the Melting and the Sublimation Curves of Ordinary Water Substance

Inputs

Type	Name	Default	Description
Temp_K	T		Temperature [K]

Outputs

Type	Name	Description
Pressure	pm	Melting pressure of iceI(for T from 273.16 to 251.165 K) [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries.pmIceIII_T

Melting pressure of ice III (temperature range from 251.165 to 256.164 K)

Information

Equation 2 from:
The International Association for the Properties of Water and Steam
Milan, Italy
September 1993
Release on the Pressure along the Melting and the Sublimation Curves of Ordinary Water Substance

Inputs

Type	Name	Default	Description
Temp_K	T		Temperature [K]

Outputs

Type	Name	Description
Pressure	pm	Melting pressure of iceIII(for T from 251.165 to 256.164 K) [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries.pmIceV_T

Melting pressure of ice V (temperature range from 256.164 to 273.31 K)

Information

Equation 3 from:
The International Association for the Properties of Water and Steam
Milan, Italy
September 1993
Release on the Pressure along the Melting and the Sublimation Curves of Ordinary Water Substance

Inputs

Type	Name	Default	Description
Temp_K	T		Temperature [K]

Outputs

Type	Name	Description
Pressure	pm	Melting pressure of iceV(for T from 256.164 to 273.31 K) [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.IceBoundaries.sublimationPressure_T

Sublimation pressure, valid from 190 to 273.16 K

Information

Equation 6 from:
The International Association for the Properties of Water and Steam
Milan, Italy
September 1993
Release on the Pressure along the Melting and the Sublimation Curves of Ordinary Water Substance

Inputs

Type	Name	Default	Description
Temp_K	T		Temperature [K]

Outputs

Type	Name	Description
Pressure	psubl	sublimation pressure (for T from 190 to 273.16) [Pa]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Transport

transport properties for water according to IAPWS/IF97

Information

Package description

Package contents

- Function **visc_dTp** implements a function to compute the industrial formulation of the dynamic viscosity of water as a function of density and temperature. The details are described in the document visc.pdf.
- Function **cond_dTp** implements a function to compute the industrial formulation of the thermal conductivity of water as a function of density, temperature and pressure. **Important note**: Obviously only two of the three inputs are really needed, but using three inputs speeds up the computation and the three variables are known in most models anyways. The inputs d,T and p have to be consistent. The details are described in the document surf.pdf.

- Function **surfaceTension** implements a function to compute the surface tension between vapour and liquid water as a function of temperature. The details are described in the document thcond.pdf.

Version Info and Revision history

- First implemented: *October, 2002* by Hubertus Tummescheit

Authors: Hubertus Tummescheit and Jonas Eborn
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: October 2002

Package Content

Name	Description
visc_dTp	dynamic viscosity eta(d,T,p), industrial formulation
cond_dTp	Thermal conductivity lam(d,T,p) (industrial use version) only in one-phase region
surfaceTension	surface tension in region 4 between steam and water

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.Transport.visc_dTp

dynamic viscosity eta(d,T,p), industrial formulation

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]
Pressure	p		pressure (only needed for region of validity) [Pa]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
DynamicViscosity	eta	dynamic viscosity [Pa.s]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Transport.cond_dTp

Thermal conductivity lam(d,T,p) (industrial use version) only in one-phase region

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]
Pressure	p		pressure [Pa]

Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Boolean	industrialMethod	true	if true, the industrial method is used, otherwise the scientific one

Outputs

Type	Name	Description
ThermalConductivity	lambda	thermal conductivity [W/(m.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Transport.surfaceTension

surface tension in region 4 between steam and water

Inputs

Type	Name	Default	Description
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
SurfaceTension	sigma	surface tension in SI units [N/m]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic

functions for calculating the isentropic enthalpy from pressure p and specific entropy s

Information

Package description

Package contents

- Function **hofpT1** computes h(p,T) in region 1.
- Function **handsofpT1** computes (s,h)=f(p,T) in region 1, needed for two-phase properties.
- Function **hofps1** computes h(p,s) in region 1.
- Function **hofpT2** computes h(p,T) in region 2.
- Function **handsofpT2** computes (s,h)=f(p,T) in region 2, needed for two-phase properties.
- Function **hofps2** computes h(p,s) in region 2.
- Function **hofdT3** computes h(d,T) in region 3.
- Function **hofpsdt3** computes h(p,s,dguess,Tguess) in region 3, where dguess and Tguess are initial guess values for the density and temperature consistent with p and s.
- Function **hofps4** computes h(p,s) in region 4.
- Function **hofpT5** computes h(p,T) in region 5.
- Function **water_hisentropic** computes h(p,s,phase) in all regions. The phase input is needed due to discontinuous derivatives at the phase boundary.
- Function **water_hisentropic_dyn** computes h(p,s,dguess,Tguess,phase) in all regions. The phase input is needed due to discontinuous derivatives at the phase boundary. Tguess and dguess are initial guess values for the density and temperature consistent with p and s. This function should be preferred in dynamic simulations where good guesses are often available.

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Documentation added: December 2002

Package Content

Name	Description
hofpT1	intermediate function for isentropic specific enthalpy in region 1
handsofpT1	special function for specific enthalpy and specific entropy in region 1
hofps1	function for isentropic specific enthalpy in region 1
hofpT2	intermediate function for isentropic specific enthalpy in region 2
handsofpT2	function for isentropic specific enthalpy and specific entropy in region 2
hofps2	function for isentropic specific enthalpy in region 2
hofdT3	function for isentropic specific enthalpy in region 3
hofps3	isentropic specific enthalpy in region 3 h(p,s)
hofpsdt3	isentropic specific enthalpy in region 3 h(p,s) with given good guess in d and T
hofps4	isentropic specific enthalpy in region 4 h(p,s)
hofpT5	specific enthalpy in region 5 h(p,T)
water_hisentropic	isentropic specific enthalpy from p,s (preferably use water_hisentropic_dyn in dynamic simulation!)
water_hisentropic_dyn	isentropic specific enthalpy from p,s and good guesses of d and T

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofpT1

intermediate function for isentropic specific enthalpy in region 1

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.handsofpT1

special function for specific enthalpy and specific entropy in region 1

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofps1

function for isentropic specific enthalpy in region 1

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofpT2

intermediate function for isentropic specific enthalpy in region 2

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.handsofpT2

function for isentropic specific enthalpy and specific entropy in region 2

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofps2

function for isentropic specific enthalpy in region 2

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofdT3

function for isentropic specific enthalpy in region 3

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofps3

isentropic specific enthalpy in region 3 h(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofpsdt3

isentropic specific enthalpy in region 3 h(p,s) with given good guess in d and T

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Density	dguess		good guess density, e.g. from adjacent volume [kg/m3]
Temperature	Tguess		good guess temperature, e.g. from adjacent volume [K]
Pressure	delp	IterationData.DELP	relative error in p [Pa]
SpecificEntropy	dels	IterationData.DELS	relative error in s [J/(kg.K)]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofps4

isentropic specific enthalpy in region 4 h(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.hofpT5

specific enthalpy in region 5 h(p,T)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.water_hisentropic

isentropic specific enthalpy from p,s (preferably use water_hisentropic_dyn in dynamic simulation!)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Integer	phase	0	phase: 2 for two-phase, 1 for one phase, 0 if unknown

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Isentropic.water_hisentropic_dyn

isentropic specific enthalpy from p,s and good guesses of d and T

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Density	dguess		good guess density, e.g. from adjacent volume [kg/m3]
Temperature	Tguess		good guess temperature, e.g. from adjacent volume [K]
Integer	phase		1 for one phase, 2 for two phase

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses

efficient inverses for selected pairs of variables

Information

Package description

Package contents

- Function **fixdT** constrains density and temperature to allowed region
- Function **dofp13** computes d as a function of p at boundary between regions 1 and 3
- Function **dofp23** computes d as a function of p at boundary between regions 2 and 3
- Function **dofpt3** iteration to compute d as a function of p and T in region 3
- Function **dtofph3** iteration to compute d and T as a function of p and h in region 3
- Function **dtofps3** iteration to compute d and T as a function of p and s in region 3
- Function **dtofpsdt3** iteration to compute d and T as a function of p and s in region 3, with initial guesses
- Function **pofdt125** iteration to compute p as a function of p and T in regions 1, 2 and 5
- Function **tofph5** iteration to compute T as a function of p and h in region 5
- Function **tofps5** iteration to compute T as a function of p and s in region 5
- Function **tofpst5** iteration to compute T as a function of p and s in region 5, with initial guess in T
- Function

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Documentation added: December 2002

Package Content

Name	Description
fixdT	region limits for inverse iteration in region 3
dofp13	density at the boundary between regions 1 and 3
dofp23	density at the boundary between regions 2 and 3
dofpt3	inverse iteration in region 3: (d) = f(p,T)
dtofph3	inverse iteration in region 3: (d,T) = f(p,h)
dtofps3	inverse iteration in region 3: (d,T) = f(p,s)
dtofpsdt3	inverse iteration in region 3: (d,T) = f(p,s)
pofdt125	inverse iteration in region 1,2 and 5: p = g(d,T)
tofph5	inverse iteration in region 5: (p,T) = f(p,h)
tofps5	inverse iteration in region 5: (p,T) = f(p,s)
tofpst5	inverse iteration in region 5: (p,T) = f(p,s)

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.fixdT

region limits for inverse iteration in region 3

Inputs

Type	Name	Default	Description
Density	din		density [kg/m3]
Temperature	Tin		temperature [K]

Outputs

Type	Name	Description
Density	dout	density [kg/m3]
Temperature	Tout	temperature [K]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dofp13

density at the boundary between regions 1 and 3

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dofp23

density at the boundary between regions 2 and 3

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
Density	d	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dofpt3

inverse iteration in region 3: (d) = f(p,T)

Inputs

Type	Name	Default	Description

Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]
Pressure	delp		iteration converged if (p-pre(p) < delp) [Pa]

Outputs

Type	Name	Description
Density	d	density [kg/m3]
Integer	error	error flag: iteration failed if different from 0

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dtofph3

inverse iteration in region 3: (d,T) = f(p,h)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Pressure	delp		iteration accuracy [Pa]
SpecificEnthalpy	delh		iteration accuracy [J/kg]

Outputs

Type	Name	Description
Density	d	density [kg/m3]
Temperature	T	temperature (K) [K]
Integer	error	error flag: iteration failed if different from 0

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dtofps3

inverse iteration in region 3: (d,T) = f(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Pressure	delp		iteration accuracy [Pa]
SpecificEntropy	dels		iteration accuracy [J/(kg.K)]

Outputs

Type	Name	Description
Density	d	density [kg/m3]
Temperature	T	temperature (K) [K]
Integer	error	error flag: iteration failed if different from 0

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.dtofpsdt3

inverse iteration in region 3: (d,T) = f(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Density	dguess		guess density, e.g. from adjacent volume [kg/m3]
Temperature	Tguess		guess temperature, e.g. from adjacent volume [K]
Pressure	delp		iteration accuracy [Pa]
SpecificEntropy	dels		iteration accuracy [J/(kg.K)]

Outputs

Type	Name	Description
Density	d	density [kg/m3]
Temperature	T	temperature (K) [K]
Integer	error	error flag: iteration failed if different from 0

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.pofdt125

inverse iteration in region 1,2 and 5: p = g(d,T)

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]
Pressure	reldd		relative iteration accuracy of density [Pa]
Integer	region		region in IAPWS/IF97 in which inverse should be calculated

Outputs

Type	Name	Description
Pressure	p	pressure [Pa]
Integer	error	error flag: iteration failed if different from 0

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.tofph5

inverse iteration in region 5: (p,T) = f(p,h)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
SpecificEnthalpy	reldh		iteration accuracy [J/kg]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]
Integer	error	error flag: iteration failed if different from 0

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.tofps5

inverse iteration in region 5: (p,T) = f(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
SpecificEnthalpy	relds		iteration accuracy [J/kg]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]
Integer	error	error flag: iteration failed if different from 0

Modelica.Media.Water.IF97_Utilities.BaseIF97.Inverses.tofpst5

inverse iteration in region 5: (p,T) = f(p,s)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Temperature	Tguess		guess temperature, e.g. from adjacent volume [K]
SpecificEntropy	relds		iteration accuracy [J/(kg.K)]

Outputs

Type	Name	Description
Temperature	T	temperature (K) [K]
Integer	error	error flag: iteration failed if different from 0

Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion

simple explicit functions for one region only

Information

Package description

Package ByRegion provides fast forward calls for dynamic property calculation records for all one phase

regions of IAPWS/IF97

Package contents

- Function **waterR1_pT** computes dynamic properties for region 1 using (p,T) as inputs
- Function **waterR2_pT** computes dynamic properties for region 2 using (p,T) as inputs
- Function **waterR3_dT** computes dynamic properties for region 3 using (d,T) as inputs
- Function **waterR5_pT** computes dynamic properties for region 5 using (p,T) as inputs

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Documented and re-organized: January 2003

Package Content

Name	Description
waterR1_pT	standard properties for region 1, (p,T) as inputs
waterR2_pT	standard properties for region 2, (p,T) as inputs
waterR3_dT	standard properties for region 3, (d,T) as inputs
waterR5_pT	standard properties for region 5, (p,T) as inputs

Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion.waterR1_pT

standard properties for region 1, (p,T) as inputs

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
ThermoProperties_pT	pro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion.waterR2_pT

standard properties for region 2, (p,T) as inputs

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
ThermoProperties_pT	pro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion.waterR3_dT

standard properties for region 3, (d,T) as inputs

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
ThermoProperties_dT	pro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.BaseIF97.ByRegion.waterR5_pT

standard properties for region 5, (p,T) as inputs

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
ThermoProperties_pT	pro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase

steam properties in the two-phase rgion and on the phase boundaries

Information

Package description

Package TwoPhase provides functions to compute the steam properties in the two-phase region and on the phase boundaries

Package contents

- Function **WaterLiq_p** computes properties on the boiling boundary as a function of p
- Function **WaterVap_p** computes properties on the dew line boundary as a function of p
- Function **WaterSat_ph** computes properties on both phase boundaries and in the two phase region as a function of p
- Function **WaterR4_ph** computes dynamic simulation properties in region 4 with (p,h) as inputs
- Function **WaterR4_dT** computes dynamic simulation properties in region 4 with (d,T) as inputs

Version Info and Revision history

- First implemented: *July, 2000* by Hubertus Tummescheit

Author: Hubertus Tummescheit,
Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
email: hubertus@modelon.se

- Initial version: July 2000
- Documented and re-organized: January 2003

Package Content

Name	Description
waterLiq_p	properties on the liquid phase boundary of region 4
waterVap_p	properties on the vapour phase boundary of region 4
waterSat_ph	Water saturation properties in the 2-phase region (4) as f(p,h)
waterR4_ph	Water/Steam properties in region 4 of IAPWS/IF97 (two-phase)
waterR4_dT	Water properties in region 4 as function of d and T

Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterLiq_p

properties on the liquid phase boundary of region 4

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
PhaseBoundaryProperties	liq	liquid thermodynamic property collection

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterVap_p

properties on the vapour phase boundary of region 4

Inputs

Type	Name	Default	Description

Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
PhaseBoundaryProperties	vap	vapour thermodynamic property collection

---

Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterSat_ph

Water saturation properties in the 2-phase region (4) as f(p,h)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
SaturationProperties	pro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterR4_ph

Water/Steam properties in region 4 of IAPWS/IF97 (two-phase)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
ThermoProperties_ph	pro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.BaseIF97.TwoPhase.waterR4_dT

Water properties in region 4 as function of d and T

Inputs

Type	Name	Default	Description
Density	d		Density [kg/m3]
Temperature	T		temperature [K]

Outputs

Type	Name	Description
ThermoProperties_dT	pro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.BaseIF97.extraDerivs_ph

function to calculate some extra thermophysical properties in regions 1, 2, 3 and 5 as f(p,h)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	phase: 2 for two-phase, 1 for one phase, 0 if unknown

Outputs

Type	Name	Description
ExtraDerivatives	dpro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.BaseIF97.extraDerivs_pT

function to calculate some extra thermophysical properties in regions 1, 2, 3 and 5 as f(p,T)

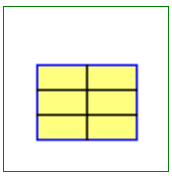
Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]

Outputs

Type	Name	Description
ExtraDerivatives	dpro	thermodynamic property collection

Modelica.Media.Water.IF97_Utilities.iter



Modelica definition

```
replaceable record iter = BaseIF97.IterationData;
```

Modelica.Media.Water.IF97_Utilities.waterBaseProp_ph

intermediate property record for water

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	phase: 2 for two-phase, 1 for one phase, 0 if unknown
Integer	region	0	if 0, do region computation, otherwise assume the region is this input

Outputs

Type	Name	Description
IF97BaseTwoPhase	aux	auxiliary record

Modelica.Media.Water.IF97_Utilities.waterBaseProp_ps

intermediate property record for water

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Integer	phase	0	phase: 2 for two-phase, 1 for one phase, 0 if unknown
Integer	region	0	if 0, do region computation, otherwise assume the region is this input

Outputs

Type	Name	Description
IF97BaseTwoPhase	aux	auxiliary record

Modelica.Media.Water.IF97_Utilities.rho_props_ps

density as function of pressure and specific entropy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
IF97BaseTwoPhase	properties		auxiliary record

Outputs

Type	Name	Description
Density	rho	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.rho_ps

density as function of pressure and specific entropy

Inputs

Type	Name	Default	Description

Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Density	rho	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.T_props_ps

temperature as function of pressure and specific entropy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
IF97BaseTwoPhase	properties		auxiliary record

Outputs

Type	Name	Description
Temperature	T	temperature [K]

Modelica.Media.Water.IF97_Utilities.T_ps

temperature as function of pressure and specific entropy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Water.IF97_Utilities.h_props_ps

specific enthalpy as function or pressure and temperature

Inputs

Type	Name	Default	Description

Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.h_ps

specific enthalpy as function or pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.phase_ps

phase as a function of pressure and specific entropy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]

Outputs

Type	Name	Description
Integer	phase	true if in liquid or gas or supercritical region

---

Modelica.Media.Water.IF97_Utilities.phase_ph

phase as a function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]

Outputs

Type	Name	Description
Integer	phase	true if in liquid or gas or supercritical region

---

Modelica.Media.Water.IF97_Utilities.phase_dT

phase as a function of pressure and temperature

Inputs

Type	Name	Default	Description
Density	rho		density [kg/m3]
Temperature	T		temperature [K]

Outputs

Type	Name	Description
Integer	phase	true if in liquid or gas or supercritical region

---

Modelica.Media.Water.IF97_Utilities.rho_props_ph

density as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	properties		auxiliary record

Outputs

Type	Name	Description
Density	rho	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.rho_ph

density as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Density	rho	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.rho_ph_der

derivative function of rho_ph

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record
Real	p_der		derivative of pressure
Real	h_der		derivative of specific enthalpy

Outputs

Type	Name	Description
Real	rho_der	derivative of density

Modelica.Media.Water.IF97_Utilities.T_props_ph

temperature as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	properties		auxiliary record

Outputs

Type	Name	Description
Temperature	T	temperature [K]

Modelica.Media.Water.IF97_Utilities.T_ph

temperature as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Temperature	T	Temperature [K]

Modelica.Media.Water.IF97_Utilities.T_ph_der

derivative function of T_ph

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record
Real	p_der		derivative of pressure
Real	h_der		derivative of specific enthalpy

Outputs

Type	Name	Description
Real	T_der	derivative of temperature

Modelica.Media.Water.IF97_Utilities.s_props_ph

specific entropy as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	properties		auxiliary record

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.s_ph

specific entropy as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.s_ph_der

specific entropy as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record
Real	p_der		derivative of pressure
Real	h_der		derivative of specific enthalpy

Outputs

Type	Name	Description
Real	s_der	derivative of entropy

Modelica.Media.Water.IF97_Utilities.cv_props_ph

specific heat capacity at constant volume as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cv_ph

specific heat capacity at constant volume as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.regionAssertReal

assert function for inlining

Inputs

Type	Name	Default	Description
Boolean	check		condition to check

Outputs

Type	Name	Description
Real	dummy	dummy output

Modelica.Media.Water.IF97_Utilities.cp_props_ph

specific heat capacity at constant pressure as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cp_ph

specific heat capacity at constant pressure as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.beta_props_ph

isobaric expansion coefficient as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
RelativePressureCoefficient	beta	isobaric expansion coefficient [1/K]

Modelica.Media.Water.IF97_Utilities.beta_ph

isobaric expansion coefficient as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
RelativePressureCoefficient	beta	isobaric expansion coefficient [1/K]

Modelica.Media.Water.IF97_Utilities.kappa_props_ph

isothermal compressibility factor as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	isothermal compressibility factor [1/Pa]

Modelica.Media.Water.IF97_Utilities.kappa_ph

isothermal compressibility factor as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	isothermal compressibility factor [1/Pa]

Modelica.Media.Water.IF97_Utilities.velocityOfSound_props_ph

speed of sound as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
Velocity	v_sound	speed of sound [m/s]

Modelica.Media.Water.IF97_Utilities.velocityOfSound_ph

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Velocity	v_sound	speed of sound [m/s]

Modelica.Media.Water.IF97_Utilities.isentropicExponent_props_ph

isentropic exponent as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
Real	gamma	isentropic exponent

Modelica.Media.Water.IF97_Utilities.isentropicExponent_ph

isentropic exponent as function of pressure and specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Real	gamma	isentropic exponent

Modelica.Media.Water.IF97_Utilities.ddph_props

density derivative by pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
DerDensityByPressure	ddph	density derivative by pressure [s2/m2]

Modelica.Media.Water.IF97_Utilities.ddph

density derivative by pressure

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
DerDensityByPressure	ddph	density derivative by pressure [s2/m2]

Modelica.Media.Water.IF97_Utilities.ddhp_props

density derivative by specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
DerDensityByEnthalpy	ddhp	density derivative by specific enthalpy [kg.s2/m5]

Modelica.Media.Water.IF97_Utilities.ddhp

density derivative by specific enthalpy

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEnthalpy	h		specific enthalpy [J/kg]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
DerDensityByEnthalpy	ddhp	density derivative by specific enthalpy [kg.s2/m5]

Modelica.Media.Water.IF97_Utilities.waterBaseProp_pT

intermediate property record for water (p and T prefered states)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, do region computation, otherwise assume the region is this input

Outputs

Type	Name	Description
IF97BaseTwoPhase	aux	auxiliary record

Modelica.Media.Water.IF97_Utilities.rho_props_pT

density as function or pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
Density	rho	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.rho_pT

density as function or pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Density	rho	density [kg/m3]

Modelica.Media.Water.IF97_Utilities.h_props_pT

specific enthalpy as function or pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.h_pT

specific enthalpy as function or pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		Temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.h_pT_der

derivative function of h_pT

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record
Real	p_der		derivative of pressure
Real	T_der		derivative of temperature

Outputs

Type	Name	Description
Real	h_der	derivative of specific enthalpy

Modelica.Media.Water.IF97_Utilities.rho_pT_der

derivative function of rho_pT

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record
Real	p_der		derivative of pressure
Real	T_der		derivative of temperature

Outputs

Type	Name	Description
Real	rho_der	derivative of density

Modelica.Media.Water.IF97_Utilities.s_props_pT

specific entropy as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.s_pT

temperature as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cv_props_pT

specific heat capacity at constant volume as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cv_pT

specific heat capacity at constant volume as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cp_props_pT

specific heat capacity at constant pressure as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cp_pT

specific heat capacity at constant pressure as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.beta_props_pT

isobaric expansion coefficient as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
RelativePressureCoefficient	beta	isobaric expansion coefficient [1/K]

Modelica.Media.Water.IF97_Utilities.beta_pT

isobaric expansion coefficient as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
RelativePressureCoefficient	beta	isobaric expansion coefficient [1/K]

Modelica.Media.Water.IF97_Utilities.kappa_props_pT

isothermal compressibility factor as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	isothermal compressibility factor [1/Pa]

Modelica.Media.Water.IF97_Utilities.kappa_pT

isothermal compressibility factor as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	isothermal compressibility factor [1/Pa]

Modelica.Media.Water.IF97_Utilities.velocityOfSound_props_pT

speed of sound as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
Velocity	v_sound	speed of sound [m/s]

Modelica.Media.Water.IF97_Utilities.velocityOfSound_pT

speed of sound as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Velocity	v_sound	speed of sound [m/s]

Modelica.Media.Water.IF97_Utilities.isentropicExponent_props_pT

isentropic exponent as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
Real	gamma	isentropic exponent

Modelica.Media.Water.IF97_Utilities.isentropicExponent_pT

isentropic exponent as function of pressure and temperature

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
Temperature	T		temperature [K]
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Real	gamma	isentropic exponent

Modelica.Media.Water.IF97_Utilities.waterBaseProp_dT

intermediate property record for water (d and T prefered states)

Inputs

Type	Name	Default	Description
Density	rho		density [kg/m3]
Temperature	T		temperature [K]
Integer	phase	0	phase: 2 for two-phase, 1 for one phase, 0 if unknown
Integer	region	0	if 0, do region computation, otherwise assume the region is this input

Outputs

Type	Name	Description
IF97BaseTwoPhase	aux	auxiliary record

Modelica.Media.Water.IF97_Utilities.h_props_dT

specific enthalpy as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.h_dT

specific enthalpy as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.h_dT_der

derivative function of h_dT

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record
Real	d_der		derivative of density
Real	T_der		derivative of temperature

Outputs

Type	Name	Description
Real	h_der	derivative of specific enthalpy

Modelica.Media.Water.IF97_Utilities.p_props_dT

pressure as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
Pressure	p	pressure [Pa]

Modelica.Media.Water.IF97_Utilities.p_dT

pressure as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Pressure	p	pressure [Pa]

Modelica.Media.Water.IF97_Utilities.p_dT_der

derivative function of p_dT

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record
Real	d_der		derivative of density
Real	T_der		derivative of temperature

Outputs

Type	Name	Description
Real	p_der	derivative of pressure

Modelica.Media.Water.IF97_Utilities.s_props_dT

specific entropy as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.s_dT

temperature as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		Temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cv_props_dT

specific heat capacity at constant volume as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cv_dT

specific heat capacity at constant volume as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificHeatCapacity	cv	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cp_props_dT

specific heat capacity at constant pressure as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.cp_dT

specific heat capacity at constant pressure as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificHeatCapacity	cp	specific heat capacity [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.beta_props_dT

isobaric expansion coefficient as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
RelativePressureCoefficient	beta	isobaric expansion coefficient [1/K]

Modelica.Media.Water.IF97_Utilities.beta_dT

isobaric expansion coefficient as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
RelativePressureCoefficient	beta	isobaric expansion coefficient [1/K]

Modelica.Media.Water.IF97_Utilities.kappa_props_dT

isothermal compressibility factor as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	isothermal compressibility factor [1/Pa]

Modelica.Media.Water.IF97_Utilities.kappa_dT

isothermal compressibility factor as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
IsothermalCompressibility	kappa	isothermal compressibility factor [1/Pa]

Modelica.Media.Water.IF97_Utilities.velocityOfSound_props_dT

speed of sound as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
Velocity	v_sound	speed of sound [m/s]

Modelica.Media.Water.IF97_Utilities.velocityOfSound_dT

speed of sound as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Velocity	v_sound	speed of sound [m/s]

Modelica.Media.Water.IF97_Utilities.isentropicExponent_props_dT

isentropic exponent as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
Real	gamma	isentropic exponent

Modelica.Media.Water.IF97_Utilities.isentropicExponent_dT

isentropic exponent as function of density and temperature

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
Real	gamma	isentropic exponent

Modelica.Media.Water.IF97_Utilities.hl_p

compute the saturated liquid specific h(p)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.hv_p

compute the saturated vapour specific h(p)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.sl_p

compute the saturated liquid specific s(p)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.sv_p

compute the saturated vapour specific s(p)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]

Outputs

Type	Name	Description
SpecificEntropy	s	specific entropy [J/(kg.K)]

Modelica.Media.Water.IF97_Utilities.rhol_T

compute the saturated liquid d(T)

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
Density	d	density of water at the boiling point [kg/m3]

Modelica.Media.Water.IF97_Utilities.rhov_T

compute the saturated vapour d(T)

Inputs

Type	Name	Default	Description
Temperature	T		temperature [K]

Outputs

Type	Name	Description
Density	d	density of steam at the condensation point [kg/m3]

Modelica.Media.Water.IF97_Utilities.rhol_p

compute the saturated liquid d(p)

Inputs

Type	Name	Default	Description
Pressure	p		saturation pressure [Pa]

Outputs

Type	Name	Description
Density	rho	density of steam at the condensation point [kg/m3]

Modelica.Media.Water.IF97_Utilities.rhov_p

compute the saturated vapour d(p)

Inputs

Type	Name	Default	Description
Pressure	p		saturation pressure [Pa]

Outputs

Type	Name	Description
Density	rho	density of steam at the condensation point [kg/m3]

Modelica.Media.Water.IF97_Utilities.dynamicViscosity

compute eta(d,T) in the one-phase region

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]
Pressure	p		pressure (only needed for region of validity) [Pa]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
DynamicViscosity	eta	dynamic viscosity [Pa.s]

Modelica.Media.Water.IF97_Utilities.thermalConductivity

compute lambda(d,T,p) in the one-phase region

Inputs

Type	Name	Default	Description
Density	d		density [kg/m3]
Temperature	T		temperature (K) [K]
Pressure	p		pressure [Pa]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Boolean	industrialMethod	true	if true, the industrial method is used, otherwise the scientific one

Outputs

Type	Name	Description
ThermalConductivity	lambda	thermal conductivity [W/(m.K)]

Modelica.Media.Water.IF97_Utilities.surfaceTension

compute sigma(T) at saturation T

Inputs

Type	Name	Default	Description
Temperature	T		temperature (K) [K]

Outputs

Type	Name	Description
SurfaceTension	sigma	surface tension in SI units [N/m]

Modelica.Media.Water.IF97_Utilities.isentropicEnthalpy

isentropic specific enthalpy from p,s (preferably use dynamicIsentropicEnthalpy in dynamic simulation!)

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known
Integer	region	0	if 0, region is unknown, otherwise known and this input

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.Media.Water.IF97_Utilities.isentropicEnthalpy_props

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
IF97BaseTwoPhase	aux		auxiliary record

Outputs

Type	Name	Description
SpecificEnthalpy	h	isentropic enthalpay [J/kg]

Modelica.Media.Water.IF97_Utilities.isentropicEnthalpy_der

derivative of isentropic specific enthalpy from p,s

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
IF97BaseTwoPhase	aux		auxiliary record
Real	p_der		pressure derivative
Real	s_der		entropy derivative

Outputs

Type	Name	Description
Real	h_der	specific enthalpy derivative

Modelica.Media.Water.IF97_Utilities.dynamicIsentropicEnthalpy

isentropic specific enthalpy from p,s and good guesses of d and T

Inputs

Type	Name	Default	Description
Pressure	p		pressure [Pa]
SpecificEntropy	s		specific entropy [J/(kg.K)]
Density	dguess		good guess density, e.g. from adjacent volume [kg/m3]
Temperature	Tguess		good guess temperature, e.g. from adjacent volume [K]
Integer	phase	0	2 for two-phase, 1 for one-phase, 0 if not known

Outputs

Type	Name	Description
SpecificEnthalpy	h	specific enthalpy [J/kg]

Modelica.SIunits

Library of type and unit definitions based on SI units according to ISO 31-1992

Information

This package provides predefined types, such as *Mass*, *Angle*, *Time*, based on the international standard on units, e.g.,

```
type Angle = Real(final quantity = "Angle",
                  final unit     = "rad",
                  displayUnit    = "deg");
```

as well as conversion functions from non SI-units to SI-units and vice versa in subpackage Conversions.

For an introduction how units are used in the Modelica standard library with package SIunits, have a look at: How to use SIunits.

Copyright © 1998-2007, Modelica Association and DLR.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
UsersGuide	User's Guide of SIunits Library
Conversions	Conversion functions to/from non SI units and type definitions of non SI units
Angle	
SolidAngle	
Length	
PathLength	
Position	
Distance	
Breadth	
Height	
Thickness	
Radius	
Diameter	
Area	
Volume	
Time	
Duration	
AngularVelocity	
AngularAcceleration	
Velocity	
Acceleration	
Period	
Frequency	
AngularFrequency	
Wavelength	
Wavelenght	
WaveNumber	
CircularWaveNumber	
AmplitudeLevelDifference	
PowerLevelDifference	
DampingCoefficient	
LogarithmicDecrement	
AttenuationCoefficient	
PhaseCoefficient	
PropagationCoefficient	
Damping	
Mass	
Density	
RelativeDensity	
SpecificVolume	
LinearDensity	
SurfaceDensity	

Momentum	
Impulse	
AngularMomentum	
AngularImpulse	
MomentOfInertia	
Inertia	
Force	
Weight	
Torque	
MomentOfForce	
Pressure	
AbsolutePressure	
BulkModulus	
Stress	
NormalStress	
ShearStress	
Strain	
LinearStrain	
ShearStrain	
VolumeStrain	
PoissonNumber	
ModulusOfElasticity	
ShearModulus	
SecondMomentOfArea	
SecondPolarMomentOfArea	
SectionModulus	
CoefficientOfFriction	
DynamicViscosity	
KinematicViscosity	
SurfaceTension	
Work	
Energy	
EnergyDensity	
PotentialEnergy	
KineticEnergy	
Power	
EnergyFlowRate	
EnthalpyFlowRate	
Efficiency	
MassFlowRate	
VolumeFlowRate	
MomentumFlux	
AngularMomentumFlux	
ThermodynamicTemperature	
Temp_K	
Temperature	

TemperatureDifference	
CelsiusTemperature	
Temp_C	
LinearExpansionCoefficient	
CubicExpansionCoefficient	
RelativePressureCoefficient	
PressureCoefficient	
Compressibility	
IsothermalCompressibility	
IsentropicCompressibility	
Heat	
HeatFlowRate	
HeatFlux	
DensityOfHeatFlowRate	
ThermalConductivity	
CoefficientOfHeatTransfer	
SurfaceCoefficientOfHeatTransfer	
ThermalInsulance	
ThermalResistance	
ThermalConductance	
ThermalDiffusivity	
HeatCapacity	
SpecificHeatCapacity	
SpecificHeatCapacityAtConstantPressure	
SpecificHeatCapacityAtConstantVolume	
SpecificHeatCapacityAtSaturation	
RatioOfSpecificHeatCapacities	
IsentropicExponent	
Entropy	
SpecificEntropy	
InternalEnergy	
Enthalpy	
HelmholtzFreeEnergy	
GibbsFreeEnergy	
SpecificEnergy	
SpecificInternalEnergy	
SpecificEnthalpy	
SpecificHelmholtzFreeEnergy	
SpecificGibbsFreeEnergy	
MassieuFunction	
PlanckFunction	
DerDensityByEnthalpy	
DerDensityByPressure	
DerDensityByTemperature	
DerEnthalpyByPressure	
DerEnergyByDensity	

DerEnergyByPressure	
ElectricCurrent	
Current	
ElectricCharge	
Charge	
VolumeDensityOfCharge	
SurfaceDensityOfCharge	
ElectricFieldStrength	
ElectricPotential	
Voltage	
PotentialDifference	
ElectromotiveForce	
ElectricFluxDensity	
ElectricFlux	
Capacitance	
Permittivity	
PermittivityOfVacuum	
RelativePermittivity	
ElectricSusceptibility	
ElectricPolarization	
Electrization	
ElectricDipoleMoment	
CurrentDensity	
LinearCurrentDensity	
MagneticFieldStrength	
MagneticPotential	
MagneticPotentialDifference	
MagnetomotiveForce	
CurrentLinkage	
MagneticFluxDensity	
MagneticFlux	
MagneticVectorPotential	
Inductance	
SelfInductance	
MutualInductance	
CouplingCoefficient	
LeakageCoefficient	
Permeability	
PermeabilityOfVacuum	
RelativePermeability	
MagneticSusceptibility	
ElectromagneticMoment	
MagneticDipoleMoment	
Magnetization	
MagneticPolarization	
ElectromagneticEnergyDensity	

PoyntingVector	
Resistance	
Resistivity	
Conductivity	
Reluctance	
Permeance	
PhaseDifference	
Impedance	
ModulusOfImpedance	
Reactance	
QualityFactor	
LossAngle	
Conductance	
Admittance	
ModulusOfAdmittance	
Susceptance	
InstantaneousPower	
ActivePower	
ApparentPower	
ReactivePower	
PowerFactor	
Transconductance	
InversePotential	
RadiantEnergy	
RadiantEnergyDensity	
SpectralRadiantEnergyDensity	
RadiantPower	
RadiantEnergyFluenceRate	
RadiantIntensity	
Radiance	
RadiantExtiance	
Irradiance	
Emissivity	
SpectralEmissivity	
DirectionalSpectralEmissivity	
LuminousIntensity	
LuminousFlux	
QuantityOfLight	
Luminance	
LuminousExitance	
Illuminance	
LightExposure	
LuminousEfficacy	
SpectralLuminousEfficacy	
LuminousEfficiency	
SpectralLuminousEfficiency	

CIESpectralTristimulusValues	
ChromaticityCoordinates	
SpectralAbsorptionFactor	
SpectralReflectionFactor	
SpectralTransmissionFactor	
SpectralRadianceFactor	
LinearAttenuationCoefficient	
LinearAbsorptionCoefficient	
MolarAbsorptionCoefficient	
RefractiveIndex	
StaticPressure	
SoundPressure	
SoundParticleDisplacement	
SoundParticleVelocity	
SoundParticleAcceleration	
VelocityOfSound	
SoundEnergyDensity	
SoundPower	
SoundIntensity	
AcousticImpedance	
SpecificAcousticImpedance	
MechanicalImpedance	
SoundPressureLevel	
SoundPowerLevel	
DissipationCoefficient	
ReflectionCoefficient	
TransmissionCoefficient	
AcousticAbsorptionCoefficient	
SoundReductionIndex	
EquivalentAbsorptionArea	
ReverberationTime	
LoundnessLevel	
Loundness	
RelativeAtomicMass	
RelativeMolecularMass	
NumberOfMolecules	
AmountOfSubstance	
MolarMass	
MolarVolume	
MolarInternalEnergy	
MolarHeatCapacity	
MolarEntropy	
NumberDensityOfMolecules	
MolecularConcentration	
MassConcentration	
MassFraction	

Concentration	
VolumeFraction	
MoleFraction	
ChemicalPotential	
AbsoluteActivity	
PartialPressure	
Fugacity	
StandardAbsoluteActivity	
ActivityCoefficient	
ActivityOfSolute	
ActivityCoefficientOfSolute	
StandardAbsoluteActivityOfSolute	
ActivityOfSolvent	
OsmoticCoefficientOfSolvent	
StandardAbsoluteActivityOfSolvent	
OsmoticPressure	
StoichiometricNumber	
Affinity	
MassOfMolecule	
ElectricDipoleMomentOfMolecule	
ElectricPolarizabilityOfAMolecule	
MicrocanonicalPartitionFunction	
CanonicalPartitionFunction	
GrandCanonicalPartitionFunction	
MolecularPartitionFunction	
StatisticalWeight	
MeanFreePath	
DiffusionCoefficient	
ThermalDiffusionRatio	
ThermalDiffusionFactor	
ThermalDiffusionCoefficient	
ElementaryCharge	
ChargeNumberOfIon	
FaradayConstant	
IonicStrength	
DegreeOfDissociation	
ElectrolyticConductivity	
MolarConductivity	
TransportNumberOfIonic	
ProtonNumber	
NeutronNumber	
NucleonNumber	
AtomicMassConstant	
MassOfElectron	
MassOfProton	
MassOfNeutron	

HartreeEnergy	
MagneticMomentOfParticle	
BohrMagneton	
NuclearMagneton	
GyromagneticCoefficient	
GFactorOfAtom	
GFactorOfNucleus	
LarmorAngularFrequency	
NuclearPrecessionAngularFrequency	
CyclotronAngularFrequency	
NuclearQuadrupoleMoment	
NuclearRadius	
ElectronRadius	
ComptonWavelength	
MassExcess	
MassDefect	
RelativeMassExcess	
RelativeMassDefect	
PackingFraction	
BindingFraction	
MeanLife	
LevelWidth	
Activity	
SpecificActivity	
DecayConstant	
HalfLife	
AlphaDisintegrationEnergy	
MaximumBetaParticleEnergy	
BetaDisintegrationEnergy	
ReactionEnergy	
ResonanceEnergy	
CrossSection	
TotalCrossSection	
AngularCrossSection	
SpectralCrossSection	
SpectralAngularCrossSection	
MacroscopicCrossSection	
TotalMacroscopicCrossSection	
ParticleFluence	
ParticleFluenceRate	
EnergyFluence	
EnergyFluenceRate	
CurrentDensityOfParticles	
MassAttenuationCoefficient	
MolarAttenuationCoefficient	
AtomicAttenuationCoefficient	

HalfThickness	
TotalLinearStoppingPower	
TotalAtomicStoppingPower	
TotalMassStoppingPower	
MeanLinearRange	
MeanMassRange	
LinearIonization	
TotalIonization	
Mobility	
IonNumberDensity	
RecombinationCoefficient	
NeutronNumberDensity	
NeutronSpeed	
NeutronFluenceRate	
TotalNeutronSourceDensity	
SlowingDownDensity	
ResonanceEscapeProbability	
Lethargy	
SlowingDownArea	
DiffusionArea	
MigrationArea	
SlowingDownLength	
DiffusionLength	
MigrationLength	
NeutronYieldPerFission	
NeutronYieldPerAbsorption	
FastFissionFactor	
ThermalUtilizationFactor	
NonLeakageProbability	
Reactivity	
ReactorTimeConstant	
EnergyImparted	
MeanEnergyImparted	
SpecificEnergyImparted	
AbsorbedDose	
DoseEquivalent	
AbsorbedDoseRate	
LinearEnergyTransfer	
Kerma	
KermaRate	
MassEnergyTransferCoefficient	
Exposure	
ExposureRate	
ReynoldsNumber	
EulerNumber	
FroudeNumber	

GrashofNumber	
WeberNumber	
MachNumber	
KnudsenNumber	
StrouhalNumber	
FourierNumber	
PecletNumber	
RayleighNumber	
NusseltNumber	
BiotNumber	
StantonNumber	
FourierNumberOfMassTransfer	
PecletNumberOfMassTransfer	
GrashofNumberOfMassTransfer	
NusseltNumberOfMassTransfer	
StantonNumberOfMassTransfer	
PrandtlNumber	
SchmidtNumber	
LewisNumber	
MagneticReynoldsNumber	
AlfvenNumber	
HartmannNumber	
CowlingNumber	
BraggAngle	
OrderOfReflexion	
ShortRangeOrderParameter	
LongRangeOrderParameter	
DebyeWallerFactor	
CircularWavenumber	
FermiCircularWavenumber	
DebyeCircularWavenumber	
DebyeCircularFrequency	
DebyeTemperature	
SpectralConcentration	
GrueneisenParameter	
MadelungConstant	
DensityOfStates	
ResidualResistivity	
LorenzCoefficient	
HallCoefficient	
ThermoelectromotiveForce	
SeebeckCoefficient	
PeltierCoefficient	
ThomsonCoefficient	
RichardsonConstant	
FermiEnergy	

GapEnergy	
DonorIonizationEnergy	
AcceptorIonizationEnergy	
FermiTemperature	
ElectronNumberDensity	
HoleNumberDensity	
IntrinsicNumberDensity	
DonorNumberDensity	
AcceptorNumberDensity	
EffectiveMass	
MobilityRatio	
RelaxationTime	
CarrierLifeTime	
ExchangeIntegral	
CurieTemperature	
NeelTemperature	
LondonPenetrationDepth	
CoherenceLength	
LandauGinzburgParameter	
FluxiodQuantum	

Types and constants

```
type Angle = Real (
    final quantity="Angle",
    final unit="rad",
    displayUnit="deg");

type SolidAngle = Real (final quantity="SolidAngle", final unit="sr");

type Length = Real (final quantity="Length", final unit="m");

type PathLength = Length;

type Position = Length;

type Distance = Length (min=0);

type Breadth = Length(min=0);

type Height = Length(min=0);

type Thickness = Length(min=0);

type Radius = Length(min=0);

type Diameter = Length(min=0);

type Area = Real (final quantity="Area", final unit="m2");
```

```
  type Volume = Real (final quantity="Volume", final unit="m3");

  type Time = Real (final quantity="Time", final unit="s");

  type Duration = Time;

  type AngularVelocity = Real (
      final quantity="AngularVelocity",
      final unit="rad/s",
      displayUnit="rev/min");

  type AngularAcceleration = Real (final quantity="AngularAcceleration", final
unit
      =      "rad/s2");

  type Velocity = Real (final quantity="Velocity", final unit="m/s");

  type Acceleration = Real (final quantity="Acceleration", final unit="m/s2");

  type Period = Real (final quantity="Time", final unit="s");

  type Frequency = Real (final quantity="Frequency", final unit="Hz");

  type AngularFrequency = Real (final quantity="AngularFrequency", final unit=
         "s-1");

  type Wavelength = Real (final quantity="Wavelength", final unit="m");

  type Wavelenght = Wavelength;

  type WaveNumber = Real (final quantity="WaveNumber", final unit="m-1");

  type CircularWaveNumber = Real (final quantity="CircularWaveNumber", final
unit
      =      "rad/m");

  type AmplitudeLevelDifference = Real (final quantity=
         "AmplitudeLevelDifference", final unit="dB");

  type PowerLevelDifference = Real (final quantity="PowerLevelDifference",
       final unit="dB");

  type DampingCoefficient = Real (final quantity="DampingCoefficient", final
unit
      =      "s-1");

  type LogarithmicDecrement = Real (final quantity="LogarithmicDecrement",
       final unit="1/S");

  type AttenuationCoefficient = Real (final quantity="AttenuationCoefficient",
       final unit="m-1");

  type PhaseCoefficient = Real (final quantity="PhaseCoefficient", final unit=
         "m-1");
```

```
  type PropagationCoefficient = Real (final quantity="PropagationCoefficient",
        final unit="m-1");

  type Damping = DampingCoefficient;

  type Mass = Real (
      quantity="Mass",
      final unit="kg",
      min=0);

  type Density = Real (
      final quantity="Density",
      final unit="kg/m3",
      displayUnit="g/cm3",
      min=0);

  type RelativeDensity = Real (
      final quantity="RelativeDensity",
      final unit="1",
      min=0);

  type SpecificVolume = Real (
      final quantity="SpecificVolume",
      final unit="m3/kg",
      min=0);

  type LinearDensity = Real (
      final quantity="LinearDensity",
      final unit="kg/m",
      min=0);

  type SurfaceDensity = Real (
      final quantity="SurfaceDensity",
      final unit="kg/m2",
      min=0);

  type Momentum = Real (final quantity="Momentum", final unit="kg.m/s");

  type Impulse = Real (final quantity="Impulse", final unit="N.s");

  type AngularMomentum = Real (final quantity="AngularMomentum", final unit=
          "kg.m2/s");

  type AngularImpulse = Real (final quantity="AngularImpulse", final unit=
          "N.m.s");

  type MomentOfInertia = Real (final quantity="MomentOfInertia", final unit=
          "kg.m2");

  type Inertia = MomentOfInertia;

  type Force = Real (final quantity="Force", final unit="N");

  type Weight = Force;
```

```
  type Torque = Real (final quantity="Torque", final unit="N.m");

  type MomentOfForce = Torque;

  type Pressure = Real (
      final quantity="Pressure",
      final unit="Pa",
      displayUnit="bar");

  type AbsolutePressure = Pressure (min=0);

  type BulkModulus = AbsolutePressure;

  type Stress = Real (final unit="Pa");

  type NormalStress = Stress;

  type ShearStress = Stress;

  type Strain = Real (final quantity="Strain", final unit="1");

  type LinearStrain = Strain;

  type ShearStrain = Strain;

  type VolumeStrain = Real (final quantity="VolumeStrain", final unit="1");

  type PoissonNumber = Real (final quantity="PoissonNumber", final unit="1");

  type ModulusOfElasticity = Stress;

  type ShearModulus = Stress;

  type SecondMomentOfArea = Real (final quantity="SecondMomentOfArea", final
unit
      =       "m4");

  type SecondPolarMomentOfArea = SecondMomentOfArea;

  type SectionModulus = Real (final quantity="SectionModulus", final unit="m3");

  type CoefficientOfFriction = Real (final quantity="CoefficientOfFriction",
       final unit="1");

  type DynamicViscosity = Real (
      final quantity="DynamicViscosity",
      final unit="Pa.s",
      min=0);

  type KinematicViscosity = Real (
      final quantity="KinematicViscosity",
      final unit="m2/s",
      min=0);
```

```
  type SurfaceTension = Real (final quantity="SurfaceTension", final
unit="N/m");

  type Work = Real (final quantity="Work", final unit="J");

  type Energy = Real (final quantity="Energy", final unit="J");

  type EnergyDensity = Real (final quantity="EnergyDensity", final unit="J/m3");

  type PotentialEnergy = Energy;

  type KineticEnergy = Energy;

  type Power = Real (final quantity="Power", final unit="W");

  type EnergyFlowRate = Power;

  type EnthalpyFlowRate = Real (final quantity="EnthalpyFlowRate", final unit=
        "W");

  type Efficiency = Real (
      final quantity="Efficiency",
      final unit="1",
      min=0);

  type MassFlowRate = Real (quantity="MassFlowRate", final unit="kg/s");

  type VolumeFlowRate = Real (final quantity="VolumeFlowRate", final unit=
        "m3/s");

  type MomentumFlux = Real (final quantity="MomentumFlux", final unit="N");

  type AngularMomentumFlux = Real (final quantity="AngularMomentumFlux", final
unit
      =      "N.m");

  type ThermodynamicTemperature = Real (
      final quantity="ThermodynamicTemperature",
      final unit="K",
      min = 0,
      displayUnit="degC");

  type Temp_K = ThermodynamicTemperature;

  type Temperature = ThermodynamicTemperature;

  type TemperatureDifference = Real (
      final quantity="ThermodynamicTemperature",
      final unit="K");

  type CelsiusTemperature = Real (final quantity="CelsiusTemperature", final
unit
      =      "degC");
```

```
  type Temp_C = CelsiusTemperature;

  type LinearExpansionCoefficient = Real (final quantity=
         "LinearExpansionCoefficient", final unit="1/K");

  type CubicExpansionCoefficient = Real (final quantity=
         "CubicExpansionCoefficient", final unit="1/K");

  type RelativePressureCoefficient = Real (final quantity=
         "RelativePressureCoefficient", final unit="1/K");

  type PressureCoefficient = Real (final quantity="PressureCoefficient", final
unit
      =      "Pa/K");

  type Compressibility = Real (final quantity="Compressibility", final unit=
         "1/Pa");

  type IsothermalCompressibility = Compressibility;

  type IsentropicCompressibility = Compressibility;

  type Heat = Real (final quantity="Energy", final unit="J");

  type HeatFlowRate = Real (final quantity="Power", final unit="W");

  type HeatFlux = Real (final quantity="HeatFlux", final unit="W/m2");

  type DensityOfHeatFlowRate = Real (final quantity="DensityOfHeatFlowRate",
       final unit="W/m2");

  type ThermalConductivity = Real (final quantity="ThermalConductivity", final
unit
      =      "W/(m.K)");

  type CoefficientOfHeatTransfer = Real (final quantity=
         "CoefficientOfHeatTransfer", final unit="W/(m2.K)");

  type SurfaceCoefficientOfHeatTransfer = CoefficientOfHeatTransfer;

  type ThermalInsulance = Real (final quantity="ThermalInsulance", final unit=
         "m2.K/W");

  type ThermalResistance = Real (final quantity="ThermalResistance", final unit
      =  "K/W");

  type ThermalConductance = Real (final quantity="ThermalConductance", final
unit
      =      "W/K");

  type ThermalDiffusivity = Real (final quantity="ThermalDiffusivity", final
unit
      =      "m2/s");
```

```
  type HeatCapacity = Real (final quantity="HeatCapacity", final unit="J/K");

  type SpecificHeatCapacity = Real (final quantity="SpecificHeatCapacity",
       final unit="J/(kg.K)");

  type SpecificHeatCapacityAtConstantPressure = SpecificHeatCapacity;

  type SpecificHeatCapacityAtConstantVolume = SpecificHeatCapacity;

  type SpecificHeatCapacityAtSaturation = SpecificHeatCapacity;

  type RatioOfSpecificHeatCapacities = Real (final quantity=
          "RatioOfSpecificHeatCapacities", final unit="1");

  type IsentropicExponent = Real (final quantity="IsentropicExponent", final
unit
      =      "1");

  type Entropy = Real (final quantity="Entropy", final unit="J/K");

  type SpecificEntropy = Real (final quantity="SpecificEntropy", final unit=
          "J/(kg.K)");

  type InternalEnergy = Heat;

  type Enthalpy = Heat;

  type HelmholtzFreeEnergy = Heat;

  type GibbsFreeEnergy = Heat;

  type SpecificEnergy = Real (final quantity="SpecificEnergy", final unit=
          "J/kg");

  type SpecificInternalEnergy = SpecificEnergy;

  type SpecificEnthalpy = SpecificEnergy;

  type SpecificHelmholtzFreeEnergy = SpecificEnergy;

  type SpecificGibbsFreeEnergy = SpecificEnergy;

  type MassieuFunction = Real (final quantity="MassieuFunction", final unit=
          "J/K");

  type PlanckFunction = Real (final quantity="PlanckFunction", final
unit="J/K");

  type DerDensityByEnthalpy = Real (final unit="kg.s2/m5");

  type DerDensityByPressure = Real (final unit="s2/m2");
```

```
  type DerDensityByTemperature = Real (final unit="kg/(m3.K)");

  type DerEnthalpyByPressure = Real (final unit="J.m.s2/kg2");

  type DerEnergyByDensity = Real (final unit="J.m3/kg");

  type DerEnergyByPressure = Real (final unit="J.m.s2/kg");

  type ElectricCurrent = Real (final quantity="ElectricCurrent", final
unit="A");

  type Current = ElectricCurrent;

  type ElectricCharge = Real (final quantity="ElectricCharge", final unit="C");

  type Charge = ElectricCharge;

  type VolumeDensityOfCharge = Real (
      final quantity="VolumeDensityOfCharge",
      final unit="C/m3",
      min=0);

  type SurfaceDensityOfCharge = Real (
      final quantity="SurfaceDensityOfCharge",
      final unit="C/m2",
      min=0);

  type ElectricFieldStrength = Real (final quantity="ElectricFieldStrength",
       final unit="V/m");

  type ElectricPotential = Real (final quantity="ElectricPotential", final unit
      =  "V");

  type Voltage = ElectricPotential;

  type PotentialDifference = ElectricPotential;

  type ElectromotiveForce = ElectricPotential;

  type ElectricFluxDensity = Real (final quantity="ElectricFluxDensity", final
unit
      =      "C/m2");

  type ElectricFlux = Real (final quantity="ElectricFlux", final unit="C");

  type Capacitance = Real (
      final quantity="Capacitance",
      final unit="F",
      min=0);

  type Permittivity = Real (
      final quantity="Permittivity",
      final unit="F/m",
      min=0);
```

```
  type PermittivityOfVacuum = Permittivity;

  type RelativePermittivity = Real (final quantity="RelativePermittivity",
       final unit="1");

  type ElectricSusceptibility = Real (final quantity="ElectricSusceptibility",
       final unit="1");

  type ElectricPolarization = Real (final quantity="ElectricPolarization",
       final unit="C/m2");

  type Electrization = Real (final quantity="Electrization", final unit="V/m");

  type ElectricDipoleMoment = Real (final quantity="ElectricDipoleMoment",
       final unit="C.m");

  type CurrentDensity = Real (final quantity="CurrentDensity", final unit=
         "A/m2");

  type LinearCurrentDensity = Real (final quantity="LinearCurrentDensity",
       final unit="A/m");

  type MagneticFieldStrength = Real (final quantity="MagneticFieldStrength",
       final unit="A/m");

  type MagneticPotential = Real (final quantity="MagneticPotential", final unit
      =
"A");

  type MagneticPotentialDifference = Real (final quantity=
         "MagneticPotential", final unit="A");

  type MagnetomotiveForce = Real (final quantity="MagnetomotiveForce", final
unit
      =      "A");

  type CurrentLinkage = Real (final quantity="CurrentLinkage", final unit="A");

  type MagneticFluxDensity = Real (final quantity="MagneticFluxDensity", final
unit
      =      "T");

  type MagneticFlux = Real (final quantity="MagneticFlux", final unit="Wb");

  type MagneticVectorPotential = Real (final quantity="MagneticVectorPotential",
       final unit="Wb/m");

  type Inductance = Real (
      final quantity="Inductance",
      final unit="H");

  type SelfInductance = Inductance(min=0);

  type MutualInductance = Inductance;
```

```
  type CouplingCoefficient = Real (final quantity="CouplingCoefficient", final
unit
      =      "1");

  type LeakageCoefficient = Real (final quantity="LeakageCoefficient", final
unit
      =      "1");

  type Permeability = Real (final quantity="Permeability", final unit="H/m");

  type PermeabilityOfVacuum = Permeability;

  type RelativePermeability = Real (final quantity="RelativePermeability",
       final unit="1");

  type MagneticSusceptibility = Real (final quantity="MagneticSusceptibility",
       final unit="1");

  type ElectromagneticMoment = Real (final quantity="ElectromagneticMoment",
       final unit="A.m2");

  type MagneticDipoleMoment = Real (final quantity="MagneticDipoleMoment",
       final unit="Wb.m");

  type Magnetization = Real (final quantity="Magnetization", final unit="A/m");

  type MagneticPolarization = Real (final quantity="MagneticPolarization",
       final unit="T");

  type ElectromagneticEnergyDensity = Real (final quantity="EnergyDensity",
       final unit="J/m3");

  type PoyntingVector = Real (final quantity="PoyntingVector", final unit=
         "W/m2");

  type Resistance = Real (
      final quantity="Resistance",
      final unit="Ohm");

  type Resistivity = Real (final quantity="Resistivity", final unit="Ohm.m");

  type Conductivity = Real (final quantity="Conductivity", final unit="S/m");

  type Reluctance = Real (final quantity="Reluctance", final unit="H-1");

  type Permeance = Real (final quantity="Permeance", final unit="H");

  type PhaseDifference = Real (
      final quantity="Angle",
      final unit="rad",
      displayUnit="deg");

  type Impedance = Resistance;
```

```
  type ModulusOfImpedance = Resistance;

  type Reactance = Resistance;

  type QualityFactor = Real (final quantity="QualityFactor", final unit="1");

  type LossAngle = Real (
      final quantity="Angle",
      final unit="rad",
      displayUnit="deg");

  type Conductance = Real (
      final quantity="Conductance",
      final unit="S");

  type Admittance = Conductance;

  type ModulusOfAdmittance = Conductance;

  type Susceptance = Conductance;

  type InstantaneousPower = Real (final quantity="Power", final unit="W");

  type ActivePower = Real (final quantity="Power", final unit="W");

  type ApparentPower = Real (final quantity="Power", final unit="VA");

  type ReactivePower = Real (final quantity="Power", final unit="var");

  type PowerFactor = Real (final quantity="PowerFactor", final unit="1");

  type Transconductance = Real (final quantity="Transconductance", final unit=
         "A/V2");

  type InversePotential = Real (final quantity="InversePotential", final unit=
         "1/V");

  type RadiantEnergy = Real (final quantity="Energy", final unit="J");

  type RadiantEnergyDensity = Real (final quantity="EnergyDensity", final unit=
         "J/m3");

  type SpectralRadiantEnergyDensity = Real (final quantity=
         "SpectralRadiantEnergyDensity", final unit="J/m4");

  type RadiantPower = Real (final quantity="Power", final unit="W");

  type RadiantEnergyFluenceRate = Real (final quantity=
         "RadiantEnergyFluenceRate", final unit="W/m2");

  type RadiantIntensity = Real (final quantity="RadiantIntensity", final unit=
         "W/sr");
```

```
  type Radiance = Real (final quantity="Radiance", final unit="W/(sr.m2)");

  type RadiantExtiance = Real (final quantity="RadiantExtiance", final unit=
         "W/m2");

  type Irradiance = Real (final quantity="Irradiance", final unit="W/m2");

  type Emissivity = Real (final quantity="Emissivity", final unit="1");

  type SpectralEmissivity = Real (final quantity="SpectralEmissivity", final
unit
      =      "1");

  type DirectionalSpectralEmissivity = Real (final quantity=
         "DirectionalSpectralEmissivity", final unit="1");

  type LuminousIntensity = Real (final quantity="LuminousIntensity", final unit
     =  "cd");

  type LuminousFlux = Real (final quantity="LuminousFlux", final unit="lm");

  type QuantityOfLight = Real (final quantity="QuantityOfLight", final unit=
         "lm.s");

  type Luminance = Real (final quantity="Luminance", final unit="cd/m2");

  type LuminousExitance = Real (final quantity="LuminousExitance", final unit=
         "lm/m2");

  type Illuminance = Real (final quantity="Illuminance", final unit="lx");

  type LightExposure = Real (final quantity="LightExposure", final unit="lx.s");

  type LuminousEfficacy = Real (final quantity="LuminousEfficacy", final unit=
         "lm/W");

  type SpectralLuminousEfficacy = Real (final quantity=
         "SpectralLuminousEfficacy", final unit="lm/W");

  type LuminousEfficiency = Real (final quantity="LuminousEfficiency", final
unit
      =      "1");

  type SpectralLuminousEfficiency = Real (final quantity=
         "SpectralLuminousEfficiency", final unit="1");

  type CIESpectralTristimulusValues = Real (final quantity=
         "CIESpectralTristimulusValues", final unit="1");

  type ChromaticityCoordinates = Real (final quantity="CromaticityCoordinates",
         final unit="1");

  type SpectralAbsorptionFactor = Real (final quantity=
         "SpectralAbsorptionFactor", final unit="1");
```

```
  type SpectralReflectionFactor = Real (final quantity=
         "SpectralReflectionFactor", final unit="1");

  type SpectralTransmissionFactor = Real (final quantity=
         "SpectralTransmissionFactor", final unit="1");

  type SpectralRadianceFactor = Real (final quantity="SpectralRadianceFactor",
       final unit="1");

  type LinearAttenuationCoefficient = Real (final quantity=
         "AttenuationCoefficient", final unit="m-1");

  type LinearAbsorptionCoefficient = Real (final quantity=
         "LinearAbsorptionCoefficient", final unit="m-1");

  type MolarAbsorptionCoefficient = Real (final quantity=
         "MolarAbsorptionCoefficient", final unit="m2/mol");

  type RefractiveIndex = Real (final quantity="RefractiveIndex", final
unit="1");

  type StaticPressure = Real (
      final quantity="Pressure",
      final unit="Pa",
      displayUnit="bar",
      min=0);

  type SoundPressure = StaticPressure;

  type SoundParticleDisplacement = Real (final quantity="Length", final unit=
         "m");

  type SoundParticleVelocity = Real (final quantity="Velocity", final unit=
         "m/s");

  type SoundParticleAcceleration = Real (final quantity="Acceleration", final
unit
      =      "m/s2");

  type VelocityOfSound = Real (final quantity="Velocity", final unit="m/s");

  type SoundEnergyDensity = Real (final quantity="EnergyDensity", final unit=
         "J/m3");

  type SoundPower = Real (final quantity="Power", final unit="W");

  type SoundIntensity = Real (final quantity="SoundIntensity", final unit=
         "W/m2");

  type AcousticImpedance = Real (final quantity="AcousticImpedance", final unit
      =  "Pa.s/m3");

  type SpecificAcousticImpedance = Real (final quantity=
         "SpecificAcousticImpedance", final unit="Pa.s/m");
```

```
  type MechanicalImpedance = Real (final quantity="MechanicalImpedance", final
unit
      =      "N.s/m");

  type SoundPressureLevel = Real (final quantity="SoundPressureLevel", final
unit
      =      "dB");

  type SoundPowerLevel = Real (final quantity="SoundPowerLevel", final unit=
         "dB");

  type DissipationCoefficient = Real (final quantity="DissipationCoefficient",
       final unit="1");

  type ReflectionCoefficient = Real (final quantity="ReflectionCoefficient",
       final unit="1");

  type TransmissionCoefficient = Real (final quantity="TransmissionCoefficient",
         final unit="1");

  type AcousticAbsorptionCoefficient = Real (final quantity=
         "AcousticAbsorptionCoefficient", final unit="1");

  type SoundReductionIndex = Real (final quantity="SoundReductionIndex", final
unit
      =      "dB");

  type EquivalentAbsorptionArea = Real (final quantity="Area", final unit="m2");

  type ReverberationTime = Real (final quantity="Time", final unit="s");

  type LoundnessLevel = Real (final quantity="LoundnessLevel", final unit=
         "phon");

  type Loundness = Real (final quantity="Loundness", final unit="sone");

  type RelativeAtomicMass = Real (final quantity="RelativeAtomicMass", final
unit
      =      "1");

  type RelativeMolecularMass = Real (final quantity="RelativeMolecularMass",
       final unit="1");

  type NumberOfMolecules = Real (final quantity="NumberOfMolecules", final unit
      =  "1");

  type AmountOfSubstance = Real (
      final quantity="AmountOfSubstance",
      final unit="mol",
      min=0);

  type MolarMass = Real (final quantity="MolarMass", final unit="kg/mol");

  type MolarVolume = Real (final quantity="MolarVolume", final unit="m3/mol");
```

```
  type MolarInternalEnergy = Real (final quantity="MolarInternalEnergy", final
unit
      =      "J/mol");

  type MolarHeatCapacity = Real (final quantity="MolarHeatCapacity", final unit
      =  "J/(mol.K)");

  type MolarEntropy = Real (final quantity="MolarEntropy", final unit=
         "J/(mol.K)");

  type NumberDensityOfMolecules = Real (final quantity=
         "NumberDensityOfMolecules", final unit="m-3");

  type MolecularConcentration = Real (final quantity="MolecularConcentration",
       final unit="m-3");

  type MassConcentration = Real (final quantity="MassConcentration", final unit
      =  "kg/m3");

  type MassFraction = Real (final quantity="MassFraction", final unit="1");

  type Concentration = Real (final quantity="Concentration", final unit=
         "mol/m3");

  type VolumeFraction = Real (final quantity="VolumeFraction", final unit="1");

  type MoleFraction = Real (final quantity="MoleFraction", final unit="1");

  type ChemicalPotential = Real (final quantity="ChemicalPotential", final unit
      =  "J/mol");

  type AbsoluteActivity = Real (final quantity="AbsoluteActivity", final unit=
         "1");

  type PartialPressure = Real (
      final quantity="Pressure",
      final unit="Pa",
      displayUnit="bar",
      min=0);

  type Fugacity = Real (final quantity="Fugacity", final unit="Pa");

  type StandardAbsoluteActivity = Real (final quantity=
         "StandardAbsoluteActivity", final unit="1");

  type ActivityCoefficient = Real (final quantity="ActivityCoefficient", final
unit
      =      "1");

  type ActivityOfSolute = Real (final quantity="ActivityOfSolute", final unit=
         "1");

  type ActivityCoefficientOfSolute = Real (final quantity=
         "ActivityCoefficientOfSolute", final unit="1");
```

```
type StandardAbsoluteActivityOfSolute = Real (final quantity=
       "StandardAbsoluteActivityOfSolute", final unit="1");

type ActivityOfSolvent = Real (final quantity="ActivityOfSolvent", final unit
    =  "1");

type OsmoticCoefficientOfSolvent = Real (final quantity=
       "OsmoticCoefficientOfSolvent", final unit="1");

type StandardAbsoluteActivityOfSolvent = Real (final quantity=
       "StandardAbsoluteActivityOfSolvent", final unit="1");

type OsmoticPressure = Real (
    final quantity="Pressure",
    final unit="Pa",
    displayUnit="bar",
    min=0);

type StoichiometricNumber = Real (final quantity="StoichiometricNumber",
      final unit="1");

type Affinity = Real (final quantity="Affinity", final unit="J/mol");

type MassOfMolecule = Real (final quantity="Mass", final unit="kg");

type ElectricDipoleMomentOfMolecule = Real (final quantity=
       "ElectricDipoleMomentOfMolecule", final unit="C.m");

type ElectricPolarizabilityOfAMolecule = Real (final quantity=
       "ElectricPolarizabilityOfAMolecule", final unit="C.m2/V");

type MicrocanonicalPartitionFunction = Real (final quantity=
       "MicrocanonicalPartitionFunction", final unit="1");

type CanonicalPartitionFunction = Real (final quantity=
       "CanonicalPartitionFunction", final unit="1");

type GrandCanonicalPartitionFunction = Real (final quantity=
       "GrandCanonicalPartitionFunction", final unit="1");

type MolecularPartitionFunction = Real (final quantity=
       "MolecularPartitionFunction", final unit="1");

type StatisticalWeight = Real (final quantity="StatisticalWeight", final unit
    =  "1");

type MeanFreePath = Length;

type DiffusionCoefficient = Real (final quantity="DiffusionCoefficient",
      final unit="m2/s");

type ThermalDiffusionRatio = Real (final quantity="ThermalDiffusionRatio",
      final unit="1");
```

```
  type ThermalDiffusionFactor = Real (final quantity="ThermalDiffusionFactor",
       final unit="1");

  type ThermalDiffusionCoefficient = Real (final quantity=
          "ThermalDiffusionCoefficient", final unit="m2/s");

  type ElementaryCharge = Real (final quantity="ElementaryCharge", final unit=
          "C");

  type ChargeNumberOfIon = Real (final quantity="ChargeNumberOfIon", final unit
      =  "1");

  type FaradayConstant = Real (final quantity="FaradayConstant", final unit=
          "C/mol");

  type IonicStrength = Real (final quantity="IonicStrength", final unit=
          "mol/kg");

  type DegreeOfDissociation = Real (final quantity="DegreeOfDissociation",
       final unit="1");

  type ElectrolyticConductivity = Real (final quantity=
          "ElectrolyticConductivity", final unit="S/m");

  type MolarConductivity = Real (final quantity="MolarConductivity", final unit
      =  "S.m2/mol");

  type TransportNumberOfIonic = Real (final quantity="TransportNumberOfIonic",
       final unit="1");

  type ProtonNumber = Real (final quantity="ProtonNumber", final unit="1");

  type NeutronNumber = Real (final quantity="NeutronNumber", final unit="1");

  type NucleonNumber = Real (final quantity="NucleonNumber", final unit="1");

  type AtomicMassConstant = Real (final quantity="Mass", final unit="kg");

  type MassOfElectron = Real (final quantity="Mass", final unit="kg");

  type MassOfProton = Real (final quantity="Mass", final unit="kg");

  type MassOfNeutron = Real (final quantity="Mass", final unit="kg");

  type HartreeEnergy = Real (final quantity="Energy", final unit="J");

  type MagneticMomentOfParticle = Real (final quantity=
          "MagneticMomentOfParticle", final unit="A.m2");

  type BohrMagneton = MagneticMomentOfParticle;

  type NuclearMagneton = MagneticMomentOfParticle;
```

```
  type GyromagneticCoefficient = Real (final quantity="GyromagneticCoefficient",
         final unit="A.m2/(J.s)");

  type GFactorOfAtom = Real (final quantity="GFactorOfAtom", final unit="1");

  type GFactorOfNucleus = Real (final quantity="GFactorOfNucleus", final unit=
         "1");

  type LarmorAngularFrequency = Real (final quantity="AngularFrequency", final
unit
      =      "s-1");

  type NuclearPrecessionAngularFrequency = Real (final quantity=
         "AngularFrequency", final unit="s-1");

  type CyclotronAngularFrequency = Real (final quantity="AngularFrequency",
       final unit="s-1");

  type NuclearQuadrupoleMoment = Real (final quantity="NuclearQuadrupoleMoment",
         final unit="m2");

  type NuclearRadius = Real (final quantity="Length", final unit="m");

  type ElectronRadius = Real (final quantity="Length", final unit="m");

  type ComptonWavelength = Real (final quantity="Length", final unit="m");

  type MassExcess = Real (final quantity="Mass", final unit="kg");

  type MassDefect = Real (final quantity="Mass", final unit="kg");

  type RelativeMassExcess = Real (final quantity="RelativeMassExcess", final
unit
      =      "1");

  type RelativeMassDefect = Real (final quantity="RelativeMassDefect", final
unit
      =      "1");

  type PackingFraction = Real (final quantity="PackingFraction", final
unit="1");

  type BindingFraction = Real (final quantity="BindingFraction", final
unit="1");

  type MeanLife = Real (final quantity="Time", final unit="s");

  type LevelWidth = Real (final quantity="LevelWidth", final unit="J");

  type Activity = Real (final quantity="Activity", final unit="Bq");

  type SpecificActivity = Real (final quantity="SpecificActivity", final unit=
         "Bq/kg");
```

```
  type DecayConstant = Real (final quantity="DecayConstant", final unit="s-1");

  type HalfLife = Real (final quantity="Time", final unit="s");

  type AlphaDisintegrationEnergy = Real (final quantity="Energy", final unit=
         "J");

  type MaximumBetaParticleEnergy = Real (final quantity="Energy", final unit=
         "J");

  type BetaDisintegrationEnergy = Real (final quantity="Energy", final
unit="J");

  type ReactionEnergy = Real (final quantity="Energy", final unit="J");

  type ResonanceEnergy = Real (final quantity="Energy", final unit="J");

  type CrossSection = Real (final quantity="Area", final unit="m2");

  type TotalCrossSection = Real (final quantity="Area", final unit="m2");

  type AngularCrossSection = Real (final quantity="AngularCrossSection", final
unit
      =      "m2/sr");

  type SpectralCrossSection = Real (final quantity="SpectralCrossSection",
       final unit="m2/J");

  type SpectralAngularCrossSection = Real (final quantity=
         "SpectralAngularCrossSection", final unit="m2/(sr.J)");

  type MacroscopicCrossSection = Real (final quantity="MacroscopicCrossSection",
         final unit="m-1");

  type TotalMacroscopicCrossSection = Real (final quantity=
         "TotalMacroscopicCrossSection", final unit="m-1");

  type ParticleFluence = Real (final quantity="ParticleFluence", final unit=
         "m-2");

  type ParticleFluenceRate = Real (final quantity="ParticleFluenceRate", final
unit
      =      "s-1.m2");

  type EnergyFluence = Real (final quantity="EnergyFluence", final unit="J/m2");

  type EnergyFluenceRate = Real (final quantity="EnergyFluenceRate", final unit
      =  "W/m2");

  type CurrentDensityOfParticles = Real (final quantity=
         "CurrentDensityOfParticles", final unit="m-2.s-1");

  type MassAttenuationCoefficient = Real (final quantity=
         "MassAttenuationCoefficient", final unit="m2/kg");
```

```
  type MolarAttenuationCoefficient = Real (final quantity=
         "MolarAttenuationCoefficient", final unit="m2/mol");

  type AtomicAttenuationCoefficient = Real (final quantity=
         "AtomicAttenuationCoefficient", final unit="m2");

  type HalfThickness = Real (final quantity="Length", final unit="m");

  type TotalLinearStoppingPower = Real (final quantity=
         "TotalLinearStoppingPower", final unit="J/m");

  type TotalAtomicStoppingPower = Real (final quantity=
         "TotalAtomicStoppingPower", final unit="J.m2");

  type TotalMassStoppingPower = Real (final quantity="TotalMassStoppingPower",
       final unit="J.m2/kg");

  type MeanLinearRange = Real (final quantity="Length", final unit="m");

  type MeanMassRange = Real (final quantity="MeanMassRange", final
unit="kg/m2");

  type LinearIonization = Real (final quantity="LinearIonization", final unit=
         "m-1");

  type TotalIonization = Real (final quantity="TotalIonization", final
unit="1");

  type Mobility = Real (final quantity="Mobility", final unit="m2/(V.s)");

  type IonNumberDensity = Real (final quantity="IonNumberDensity", final unit=
         "m-3");

  type RecombinationCoefficient = Real (final quantity=
         "RecombinationCoefficient", final unit="m3/s");

  type NeutronNumberDensity = Real (final quantity="NeutronNumberDensity",
       final unit="m-3");

  type NeutronSpeed = Real (final quantity="Velocity", final unit="m/s");

  type NeutronFluenceRate = Real (final quantity="NeutronFluenceRate", final
unit
      =      "s-1.m-2");

  type TotalNeutronSourceDensity = Real (final quantity=
         "TotalNeutronSourceDesity", final unit="s-1.m-3");

  type SlowingDownDensity = Real (final quantity="SlowingDownDensity", final
unit
      =      "s-1.m-3");

  type ResonanceEscapeProbability = Real (final quantity=
         "ResonanceEscapeProbability", final unit="1");
```

```
  type Lethargy = Real (final quantity="Lethargy", final unit="1");

  type SlowingDownArea = Real (final quantity="Area", final unit="m2");

  type DiffusionArea = Real (final quantity="Area", final unit="m2");

  type MigrationArea = Real (final quantity="Area", final unit="m2");

  type SlowingDownLength = Real (final quantity="SLength", final unit="m");

  type DiffusionLength = Length;

  type MigrationLength = Length;

  type NeutronYieldPerFission = Real (final quantity="NeutronYieldPerFission",
       final unit="1");

  type NeutronYieldPerAbsorption = Real (final quantity=
          "NeutronYieldPerAbsorption", final unit="1");

  type FastFissionFactor = Real (final quantity="FastFissionFactor", final unit
      =  "1");

  type ThermalUtilizationFactor = Real (final quantity=
          "ThermalUtilizationFactor", final unit="1");

  type NonLeakageProbability = Real (final quantity="NonLeakageProbability",
       final unit="1");

  type Reactivity = Real (final quantity="Reactivity", final unit="1");

  type ReactorTimeConstant = Real (final quantity="Time", final unit="s");

  type EnergyImparted = Real (final quantity="Energy", final unit="J");

  type MeanEnergyImparted = Real (final quantity="Energy", final unit="J");

  type SpecificEnergyImparted = Real (final quantity="SpecificEnergy", final
unit
      =      "Gy");

  type AbsorbedDose = Real (final quantity="AbsorbedDose", final unit="Gy");

  type DoseEquivalent = Real (final quantity="DoseEquivalent", final unit="Sv");

  type AbsorbedDoseRate = Real (final quantity="AbsorbedDoseRate", final unit=
          "Gy/s");

  type LinearEnergyTransfer = Real (final quantity="LinearEnergyTransfer",
       final unit="J/m");

  type Kerma = Real (final quantity="Kerma", final unit="Gy");
```

```
type KermaRate = Real (final quantity="KermaRate", final unit="Gy/s");

type MassEnergyTransferCoefficient = Real (final quantity=
       "MassEnergyTransferCoefficient", final unit="m2/kg");

type Exposure = Real (final quantity="Exposure", final unit="C/kg");

type ExposureRate = Real (final quantity="ExposureRate", final unit=
       "C/(kg.s)");

type ReynoldsNumber = Real (final quantity="ReynoldsNumber", final unit="1");

type EulerNumber = Real (final quantity="EulerNumber", final unit="1");

type FroudeNumber = Real (final quantity="FroudeNumber", final unit="1");

type GrashofNumber = Real (final quantity="GrashofNumber", final unit="1");

type WeberNumber = Real (final quantity="WeberNumber", final unit="1");

type MachNumber = Real (final quantity="MachNumber", final unit="1");

type KnudsenNumber = Real (final quantity="KnudsenNumber", final unit="1");

type StrouhalNumber = Real (final quantity="StrouhalNumber", final unit="1");

type FourierNumber = Real (final quantity="FourierNumber", final unit="1");

type PecletNumber = Real (final quantity="PecletNumber", final unit="1");

type RayleighNumber = Real (final quantity="RayleighNumber", final unit="1");

type NusseltNumber = Real (final quantity="NusseltNumber", final unit="1");

type BiotNumber = NusseltNumber;

type StantonNumber = Real (final quantity="StantonNumber", final unit="1");

type FourierNumberOfMassTransfer = Real (final quantity=
       "FourierNumberOfMassTransfer", final unit="1");

type PecletNumberOfMassTransfer = Real (final quantity=
       "PecletNumberOfMassTransfer", final unit="1");

type GrashofNumberOfMassTransfer = Real (final quantity=
       "GrashofNumberOfMassTransfer", final unit="1");

type NusseltNumberOfMassTransfer = Real (final quantity=
       "NusseltNumberOfMassTransfer", final unit="1");

type StantonNumberOfMassTransfer = Real (final quantity=
       "StantonNumberOfMassTransfer", final unit="1");
```

```
  type PrandtlNumber = Real (final quantity="PrandtlNumber", final unit="1");

  type SchmidtNumber = Real (final quantity="SchmidtNumber", final unit="1");

  type LewisNumber = Real (final quantity="LewisNumber", final unit="1");

  type MagneticReynoldsNumber = Real (final quantity="MagneticReynoldsNumber",
       final unit="1");

  type AlfvenNumber = Real (final quantity="AlfvenNumber", final unit="1");

  type HartmannNumber = Real (final quantity="HartmannNumber", final unit="1");

  type CowlingNumber = Real (final quantity="CowlingNumber", final unit="1");

  type BraggAngle = Angle;

  type OrderOfReflexion = Real (final quantity="OrderOfReflexion", final unit=
         "1");

  type ShortRangeOrderParameter = Real (final quantity="RangeOrderParameter",
       final unit="1");

  type LongRangeOrderParameter = Real (final quantity="RangeOrderParameter",
       final unit="1");

  type DebyeWallerFactor = Real (final quantity="DebyeWallerFactor", final unit
      =  "1");

  type CircularWavenumber = Real (final quantity="CircularWavenumber", final
unit
      =      "m-1");

  type FermiCircularWavenumber = Real (final quantity="FermiCircularWavenumber",
         final unit="m-1");

  type DebyeCircularWavenumber = Real (final quantity="DebyeCircularWavenumber",
         final unit="m-1");

  type DebyeCircularFrequency = Real (final quantity="AngularFrequency", final
unit
      =      "s-1");

  type DebyeTemperature = ThermodynamicTemperature;

  type SpectralConcentration = Real (final quantity="SpectralConcentration",
       final unit="s/m3");

  type GrueneisenParameter = Real (final quantity="GrueneisenParameter", final
unit
      =      "1");

  type MadelungConstant = Real (final quantity="MadelungConstant", final unit=
         "1");
```

```
  type DensityOfStates = Real (final quantity="DensityOfStates", final unit=
         "J-1/m-3");

  type ResidualResistivity = Real (final quantity="ResidualResistivity", final
unit
      =      "Ohm.m");

  type LorenzCoefficient = Real (final quantity="LorenzCoefficient", final unit
      =  "V2/K2");

  type HallCoefficient = Real (final quantity="HallCoefficient", final unit=
         "m3/C");

  type ThermoelectromotiveForce = Real (final quantity=
         "ThermoelectromotiveForce", final unit="V");

  type SeebeckCoefficient = Real (final quantity="SeebeckCoefficient", final
unit
      =      "V/K");

  type PeltierCoefficient = Real (final quantity="PeltierCoefficient", final
unit
      =      "V");

  type ThomsonCoefficient = Real (final quantity="ThomsonCoefficient", final
unit
      =      "V/K");

  type RichardsonConstant = Real (final quantity="RichardsonConstant", final
unit
      =      "A/(m2.K2)");

  type FermiEnergy = Real (final quantity="Energy", final unit="eV");

  type GapEnergy = Real (final quantity="Energy", final unit="eV");

  type DonorIonizationEnergy = Real (final quantity="Energy", final unit="eV");

  type AcceptorIonizationEnergy = Real (final quantity="Energy", final unit=
         "eV");

  type FermiTemperature = ThermodynamicTemperature;

  type ElectronNumberDensity = Real (final quantity="ElectronNumberDensity",
       final unit="m-3");

  type HoleNumberDensity = Real (final quantity="HoleNumberDensity", final unit
      =  "m-3");

  type IntrinsicNumberDensity = Real (final quantity="IntrinsicNumberDensity",
       final unit="m-3");

  type DonorNumberDensity = Real (final quantity="DonorNumberDensity", final
unit
```

```
      =       "m-3");

  type AcceptorNumberDensity = Real (final quantity="AcceptorNumberDensity",
       final unit="m-3");

  type EffectiveMass = Mass;

  type MobilityRatio = Real (final quantity="MobilityRatio", final unit="1");

  type RelaxationTime = Time;

  type CarrierLifeTime = Time;

  type ExchangeIntegral = Real (final quantity="Energy", final unit="eV");

  type CurieTemperature = ThermodynamicTemperature;

  type NeelTemperature = ThermodynamicTemperature;

  type LondonPenetrationDepth = Length;

  type CoherenceLength = Length;

  type LandauGinzburgParameter = Real (final quantity="LandauGinzburgParameter",
          final unit="1");

  type FluxiodQuantum = Real (final quantity="FluxiodQuantum", final unit="Wb");
```

Modelica.SIunits.UsersGuide

User's Guide of SIunits Library

Library **SIunits** is a **free** Modelica package providing predefined types, such as *Mass*, *Length*, *Time*, based on the international standard on units.

Package Content

Name	Description
HowToUseSIunits	How to use SIunits
Conventions	Conventions
Literature	Literature
Contact	Contact

Modelica.SIunits.UsersGuide.HowToUseSIunits

How to use SIunits

When implementing a Modelica model, every variable needs to be declared. Physical variables should be declared with a unit. The basic approach in Modelica is that the unit attribute of a variable is the **unit** in which the **equations** are **written**, for example:

```
model MassOnGround
  parameter Real m(quantity="Mass", unit="kg") "Mass";
  parameter Real f(quantity="Force", unit="N") "Driving force";
  Real s(unit="m") "Position of mass";
  Real v(unit="m/s") "Velocity of mass";
equation
  der(s) = v;
  m*der(v) = f;
end MassOnGround;
```

This means that the equations in the equation section are only correct for the specified units. A different issue is the user interface, i.e., in which unit the variable is presented to the user in graphical user interfaces, both for input (e.g., parameter menu), as well as for output (e.g., in the plot window). Preferably, the Modelica tool should provide a list of units from which the user can select, e.g., "m", "cm", "km", "inch" for quantity "Length". When storing the value in the model as a Modelica modifier, it has to be converted to the unit defined in the declaration. Additionally, the unit used in the graphical user interface has to be stored. In order to have a standardized way of doing this, Modelica provides the following three attributs for a variable of type Real:

- **quantity** to define the physical quantity (e.g. "Length", or "Energy").
- **unit** to define the unit that has to be used in order that the equations are correct (e.g. "N.m").
- **displayUnit** to define the unit used in the graphical user interface as default display unit for input and/or output.

Note, a unit, such as "N.m", is not sufficient to define uniquely the physical quantity, since, e.g., "N.m" might be either "torque" or "energy". The "quantity" attribute might therefore be used by a tool to select the corresponding menu from which the user can select a unit for the corresponding variable.

For example, after providing a value for "m" and "f" in a parameter menu of an instance of MassOnGround, a tool might generate the following code:

```
MassOnGround myObject(m(displayUnit="g")=2, f=3);
```

The meaning is that in the equations a value of "2" is used and that in the graphical user interface a value of "2000" should be used, together with the unit "g" from the unit set "Mass" (= the quantity name). Note, according to the Modelica specification a tool might ignore the "displayUnit" attribute.

In order to help the Modelica model developer, the Modelica.SIunits library provides about 450 predefined type names, together with values for the attributes **quantity**, **unit** and sometimes **displayUnit** and **min**. The unit is always selected as SI-unit according to the ISO standard. The type and the quantity names are the quantity names used in the ISO standard. "quantity" and "unit" are defined as "**final**" in order that they cannot be modified. Attributes "displayUnit" and "min" can, however, be changed in a model via a modification. The example above, might therefore be alternatively also defined as:

```
model MassOnGround
  parameter Modelica.SIunits.Mass  m "Mass";
  parameter Modelica.SIunits.Force f "Driving force";
  ...
end MassOnGround;
```

or in a short hand notation as

```
model MassOnGround
  import SI = Modelica.SIunits;
  parameter SI.Mass  m "Mass";
  parameter SI.Force f "Driving force";
  ...
end MassOnGround;
```

For some often used Non SI-units (like hour), some additional type definitions are present as Modelica.SIunits.Conversions.NonSIunits. If this is not sufficient, the user has to define its own types or use

the attributes directly in the declaration as in the example at the beginning.

Modelica.SIunits.UsersGuide.Conventions

Conventions

The following conventions are used in package SIunits:

- Modelica quantity names are defined according to the recommendations of ISO 31. Some of these name are rather long, such as "ThermodynamicTemperature". Shorter alias names are defined, e.g., "type Temp_K = ThermodynamicTemperature;".
- Modelica units are defined according to the SI base units without multiples (only exception "kg").
- For some quantities, more convenient units for an engineer are defined as "displayUnit", i.e., the default unit for display purposes (e.g., displayUnit="deg" for quantity="Angle").
- The type name is identical to the quantity name, following the convention of type names.
- All quantity and unit attributes are defined as final in order that they cannot be redefined to another value.
- Similiar quantities, such as "Length, Breadth, Height, Thickness, Radius" are defined as the same quantity (here: "Length").
- The ordering of the type declarations in this package follows ISO 31:

```
Chapter  1: Space and Time
Chapter  2: Periodic and Related Phenomena
Chapter  3: Mechanics
Chapter  4: Heat
Chapter  5: Electricity and Magnetism
Chapter  6: Light and Related Electro-Magnetic Radiations
Chapter  7: Acoustics
Chapter  8: Physical Chemistry
Chapter  9: Atomic and Nuclear Physics
Chapter 10: Nuclear Reactions and Ionizing Radiations
Chapter 11: (not defined in ISO 31-1992)
Chapter 12: Characteristic Numbers
Chapter 13: Solid State Physics
```

- Conversion functions between SI and non-SI units are available in subpackage **Conversions**.

Modelica.SIunits.UsersGuide.Literature

Literature

This package is based on the following references

ISO 31-1992:
General principles concerning quantities, units and symbols.

ISO 1000-1992:
SI units and recommendations for the use of their multiples and of certain other units.

Cardarelli F.:
Scientific Unit Conversion - A Practical Guide to Metrication. Springer 1997.

Modelica.SIunits.UsersGuide.Contact

Contact

Main Author:
Martin Otter
Deutsches Zentrum für Luft und Raumfahrt e.V. (DLR)
Institut für Robotik und Mechatronik
Abteilung für Entwurfsorientierte Regelungstechnik
Postfach 1116
D-82230 Wessling
Germany
email: Martin.Otter@dlr.de

Acknowledgements:

- Astrid Jaschinski, Hubertus Tummescheit and Christian Schweiger contributed to the implementation of this package.

Modelica.SIunits.Conversions

Conversion functions to/from non SI units and type definitions of non SI units

Information

This package provides conversion functions from the non SI Units defined in package Modelica.SIunits.Conversions.NonSIunits to the corresponding SI Units defined in package Modelica.SIunits and vice versa. It is recommended to use these functions in the following way (note, that all functions have one Real input and one Real output argument):

```
  import SI = Modelica.SIunits;
  import Modelica.SIunits.Conversions.*;
     ...
  parameter SI.Temperature     T   = from_degC(25);   // convert 25 degree
Celsius to Kelvin
  parameter SI.Angle           phi = from_deg(180);   // convert 180 degree to
radian
  parameter SI.AngularVelocity w   = from_rpm(3600);  // convert 3600
revolutions per minutes
                                                     // to radian per seconds
```

Package Content

Name	Description
NonSIunits	Type definitions of non SI units
to_degC	Convert from Kelvin to °Celsius
from_degC	Convert from °Celsius to Kelvin
to_degF	Convert from Kelvin to °Fahrenheit
from_degF	Convert from °Fahrenheit to Kelvin
to_degRk	Convert from Kelvin to °Rankine
from_degRk	Convert from °Rankine to Kelvin
to_deg	Convert from radian to degree
from_deg	Convert from degree to radian
to_rpm	Convert from radian per second to revolutions per minute

from_rpm	Convert from revolutions per minute to radian per second
to_kmh	Convert from metre per second to kilometre per hour
from_kmh	Convert from kilometre per hour to metre per second
to_day	Convert from second to day
from_day	Convert from day to second
to_hour	Convert from second to hour
from_hour	Convert from hour to second
to_minute	Convert from second to minute
from_minute	Convert from minute to second
to_litre	Convert from cubic metre to litre
from_litre	Convert from litre to cubic metre
to_kWh	Convert from Joule to kilo Watt hour
from_kWh	Convert from kilo Watt hour to Joule
to_bar	Convert from Pascal to bar
from_bar	Convert from bar to Pascal
to_gps	Convert from kilogram per second to gram per second
from_gps	Convert from gram per second to kilogram per second
ConversionIcon	Base icon for conversion functions

Modelica.SIunits.Conversions.NonSIunits

Type definitions of non SI units

Information

This package provides predefined types, such as **Angle_deg** (angle in degree), **AngularVelocity_rpm** (angular velocity in revolutions per minute) or **Temperature_degF** (temperature in degree Fahrenheit), which are in common use but are not part of the international standard on units according to ISO 31-1992 "General principles concerning quantities, units and symbols" and ISO 1000-1992 "SI units and recommendations for the use of their multiples and of certain other units".

If possible, the types in this package should not be used. Use instead types of package Modelica.SIunits. For more information on units, see also the book of Francois Cardarelli **Scientific Unit Conversion - A Practical Guide to Metrication** (Springer 1997).

Some units, such as **Temperature_degC/Temp_C** are both defined in Modelica.SIunits and in Modelica.Conversions.NonSIunits. The reason is that these definitions have been placed erroneously in Modelica.SIunits although they are not SIunits. For backward compatibility, these type definitions are still kept in Modelica.SIunits.

Package Content

Name	Description
Temperature_degC	
Temperature_degF	
Temperature_degRk	

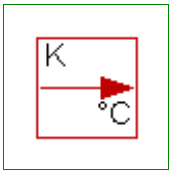
Angle_deg	
AngularVelocity_rpm	
Velocity_kmh	
Time_day	
Time_hour	
Time_minute	
Volume_litre	
Energy_kWh	
Pressure_bar	
MassFlowRate_gps	

Types and constants

```
type Temperature_degC = Real (final quantity="ThermodynamicTemperature",
      final unit="degC");

type Temperature_degF = Real (final quantity="ThermodynamicTemperature",
      final unit="degF");

type Temperature_degRk = Real (final quantity="ThermodynamicTemperature",
      final unit="degRk");

type Angle_deg = Real (final quantity="Angle", final unit="deg");

type AngularVelocity_rpm = Real (final quantity="AngularVelocity", final unit
    =      "rev/min");

type Velocity_kmh = Real (final quantity="Velocity", final unit="km/h");

type Time_day = Real (final quantity="Time", final unit="d");

type Time_hour = Real (final quantity="Time", final unit="h");

type Time_minute = Real (final quantity="Time", final unit="min");

type Volume_litre = Real (final quantity="Volume", final unit="l");

type Energy_kWh = Real (final quantity="Energy", final unit="kW.h");

type Pressure_bar = Real (final quantity="Pressure", final unit="bar");

type MassFlowRate_gps = Real (final quantity="MassFlowRate", final unit=
       "g/s");
```

Modelica.Slunits.Conversions.to_degC

Convert from Kelvin to °Celsius



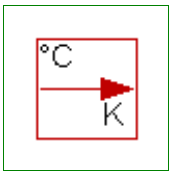
Inputs

Type	Name	Default	Description
Temperature	Kelvin		Kelvin value [K]

Outputs

Type	Name	Description
Temperature_degC	Celsius	Celsius value [degC]

Modelica.SIunits.Conversions.from_degC



Convert from °Celsius to Kelvin

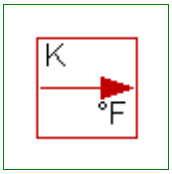
Inputs

Type	Name	Default	Description
Temperature_degC	Celsius		Celsius value [degC]

Outputs

Type	Name	Description
Temperature	Kelvin	Kelvin value [K]

Modelica.SIunits.Conversions.to_degF



Convert from Kelvin to °Fahrenheit

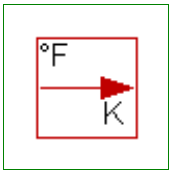
Inputs

Type	Name	Default	Description
Temperature	Kelvin		Kelvin value [K]

Outputs

Type	Name	Description
Temperature_degF	Fahrenheit	Fahrenheit value [degF]

Modelica.SIunits.Conversions.from_degF



Convert from °Fahrenheit to Kelvin

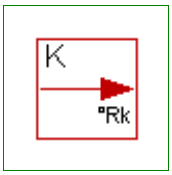
Inputs

Type	Name	Default	Description
Temperature_degF	Fahrenheit		Fahrenheit value [degF]

Outputs

Type	Name	Description
Temperature	Kelvin	Kelvin value [K]

Modelica.SIunits.Conversions.to_degRk



Convert from Kelvin to °Rankine

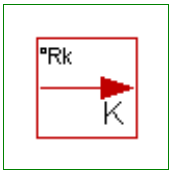
Inputs

Type	Name	Default	Description
Temperature	Kelvin		Kelvin value [K]

Outputs

Type	Name	Description
Temperature_degRk	Rankine	Rankine value [degRk]

Modelica.SIunits.Conversions.from_degRk



Convert from °Rankine to Kelvin

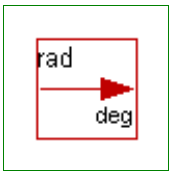
Inputs

Type	Name	Default	Description
Temperature_degRk	Rankine		Rankine value [degRk]

Outputs

Type	Name	Description
Temperature	Kelvin	Kelvin value [K]

Modelica.SIunits.Conversions.to_deg



Convert from radian to degree

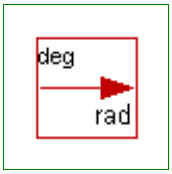
Inputs

Type	Name	Default	Description
Angle	radian		radian value [rad]

Outputs

Type	Name	Description
Angle_deg	degree	degree value [deg]

Modelica.SIunits.Conversions.from_deg



Convert from degree to radian

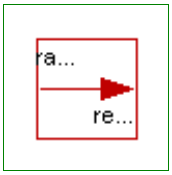
Inputs

Type	Name	Default	Description
Angle_deg	degree		degree value [deg]

Outputs

Type	Name	Description
Angle	radian	radian value [rad]

Modelica.Slunits.Conversions.to_rpm



Convert from radian per second to revolutions per minute

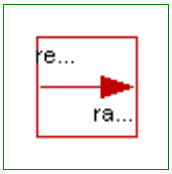
Inputs

Type	Name	Default	Description
AngularVelocity	rs		radian per second value [rad/s]

Outputs

Type	Name	Description
AngularVelocity_rpm	rpm	revolutions per minute value [rev/min]

Modelica.Slunits.Conversions.from_rpm



Convert from revolutions per minute to radian per second

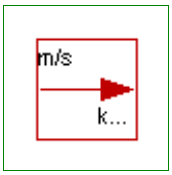
Inputs

Type	Name	Default	Description
AngularVelocity_rpm	rpm		revolutions per minute value [rev/min]

Outputs

Type	Name	Description
AngularVelocity	rs	radian per second value [rad/s]

Modelica.Slunits.Conversions.to_kmh



Convert from metre per second to kilometre per hour

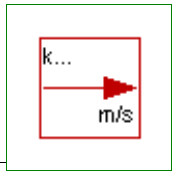
Inputs

Type	Name	Default	Description
Velocity	ms		metre per second value [m/s]

Outputs

Type	Name	Description
Velocity_kmh	kmh	kilometre per hour value [km/h]

Modelica.Slunits.Conversions.from_kmh



Convert from kilometre per hour to metre per second

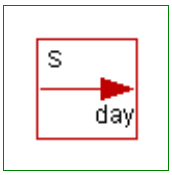
Inputs

Type	Name	Default	Description
Velocity_kmh	kmh		kilometre per hour value [km/h]

Outputs

Type	Name	Description
Velocity	ms	metre per second value [m/s]

Modelica.Slunits.Conversions.to_day



Convert from second to day

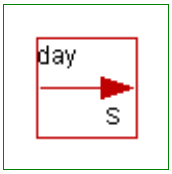
Inputs

Type	Name	Default	Description
Time	s		second value [s]

Outputs

Type	Name	Description
Time_day	day	day value [d]

Modelica.Slunits.Conversions.from_day



Convert from day to second

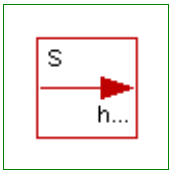
Inputs

Type	Name	Default	Description
Time_day	day		day value [d]

Outputs

Type	Name	Description
Time	s	second value [s]

Modelica.Slunits.Conversions.to_hour



Convert from second to hour

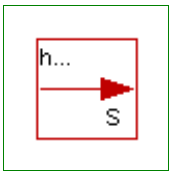
Inputs

Type	Name	Default	Description
Time	s		second value [s]

Outputs

Type	Name	Description
Time_hour	hour	hour value [h]

Modelica.Slunits.Conversions.from_hour



Convert from hour to second

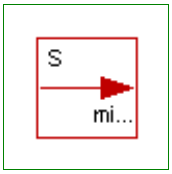
Inputs

Type	Name	Default	Description
Time_hour	hour		hour value [h]

Outputs

Type	Name	Description
Time	s	second value [s]

Modelica.Slunits.Conversions.to_minute



Convert from second to minute

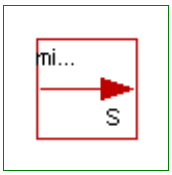
Inputs

Type	Name	Default	Description
Time	s		second value [s]

Outputs

Type	Name	Description
Time_minute	minute	minute value [min]

Modelica.Slunits.Conversions.from_minute



Convert from minute to second

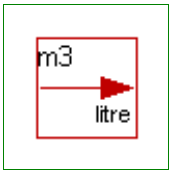
Inputs

Type	Name	Default	Description
Time_minute	minute		minute value [min]

Outputs

Type	Name	Description
Time	s	second value [s]

Modelica.Slunits.Conversions.to_litre



Convert from cubic metre to litre

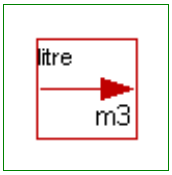
Inputs

Type	Name	Default	Description
Volume	m3		cubic metre value [m3]

Outputs

Type	Name	Description
Volume_litre	litre	litre value [l]

Modelica.SIunits.Conversions.from_litre



Convert from litre to cubic metre

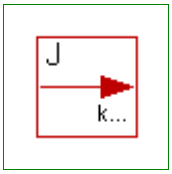
Inputs

Type	Name	Default	Description
Volume_litre	litre		litre value [l]

Outputs

Type	Name	Description
Volume	m3	cubic metre value [m3]

Modelica.SIunits.Conversions.to_kWh



Convert from Joule to kilo Watt hour

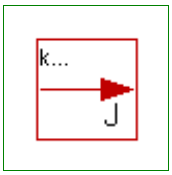
Inputs

Type	Name	Default	Description
Energy	J		Joule value [J]

Outputs

Type	Name	Description
Energy_kWh	kWh	kWh value [kW.h]

Modelica.SIunits.Conversions.from_kWh



Convert from kilo Watt hour to Joule

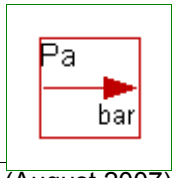
Inputs

Type	Name	Default	Description
Energy_kWh	kWh		kWh value [kW.h]

Outputs

Type	Name	Description
Energy	J	Joule value [J]

Modelica.SIunits.Conversions.to_bar



Convert from Pascal to bar

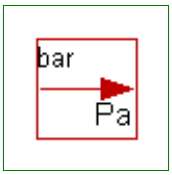
Inputs

Type	Name	Default	Description
Pressure	Pa		Pascal value [Pa]

Outputs

Type	Name	Description
Pressure_bar	bar	bar value [bar]

Modelica.Slunits.Conversions.from_bar



Convert from bar to Pascal

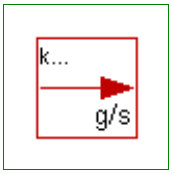
Inputs

Type	Name	Default	Description
Pressure_bar	bar		bar value [bar]

Outputs

Type	Name	Description
Pressure	Pa	Pascal value [Pa]

Modelica.Slunits.Conversions.to_gps



Convert from kilogram per second to gram per second

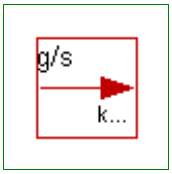
Inputs

Type	Name	Default	Description
MassFlowRate	kgps		kg/s value [kg/s]

Outputs

Type	Name	Description
MassFlowRate_gps	gps	g/s value [g/s]

Modelica.Slunits.Conversions.from_gps



Convert from gram per second to kilogram per second

Inputs

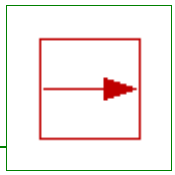
Type	Name	Default	Description
MassFlowRate_gps	gps		g/s value [g/s]

Outputs

Type	Name	Description
MassFlowRate	kgps	kg/s value [kg/s]

Modelica.Slunits.Conversions.ConversionIcon

Base icon for conversion functions



Modelica.StateGraph

Library of hierarchical state machine components to model discrete event and reactive systems

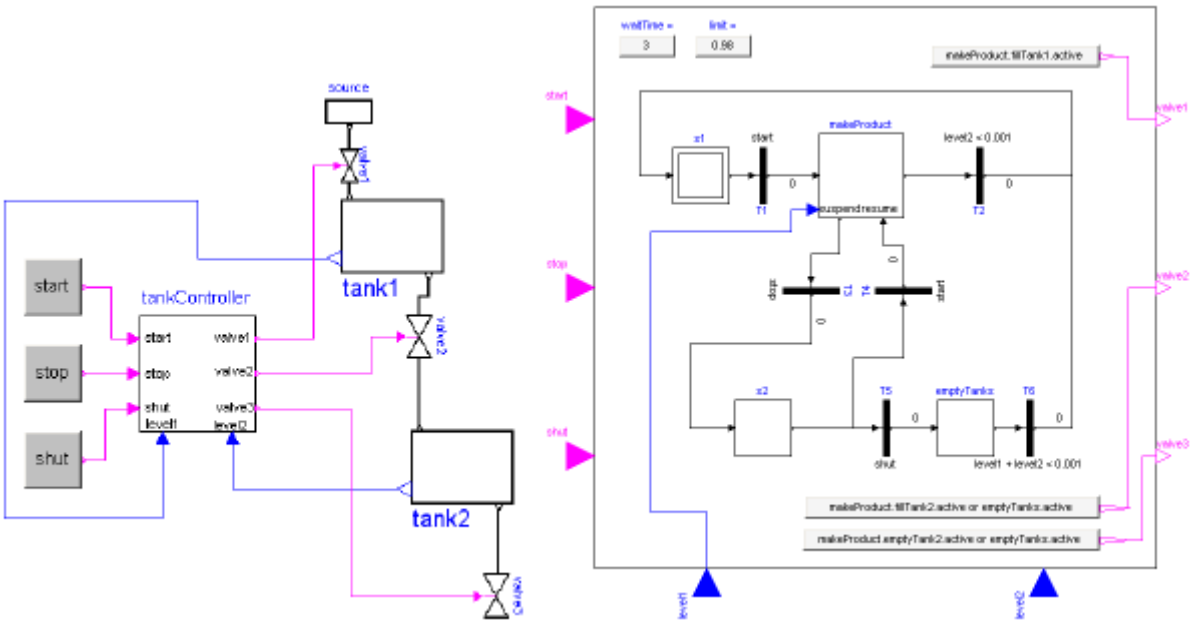
Information

Library **StateGraph** is a **free** Modelica package providing components to model **discrete event** and **reactive** systems in a convenient way. It is based on the JGraphChart method and takes advantage of Modelica features for the "action" language. JGraphChart is a further development of Grafcet to include elements of StateCharts that are not present in Grafcet/Sequential Function Charts. Therefore, the StateGraph library has a similar modeling power as StateCharts but avoids some deficiences of StateCharts.

For an introduction, have especially a look at:

- StateGraph.UsersGuide discusses the most important aspects how to use this library.
- StateGraph.Examples contains examples that demonstrate the usage of this library.

A typical model generated with this library is shown in the next figure where on the left hand side a two-tank system with a tank controller and on the right hand side the top-level part of the tank controller as a StateGraph is shown:



The unique feature of the StateGraph library with respect to JGraphCharts, Grafcet, Sequential Function Charts, and StateCharts, is Modelica's "single assignment rule" that requires that every variable is defined by exactly one equation. This leads to a different "action" definition as in these formalisms. The advantage is that the translator can either determine a useful evaluation sequence by equation sorting or reports an error if this is not possible, e.g., because a model would lead to a non-determinism or to a dead-lock. As a side effect, this leads also to simpler and more easier to understand models and global variables are no longer needed (whereas in JGraphCharts, Grafcet, Sequential Function Charts and StateCharts global variables are nearly always needed).

The StateGraph library is currently available in a beta release. The available components will most likely not be changed for the release version. It is planned to improve the convenience of building models with the

StateGraph library for the release version (this may require to introduce some additional annotations). It is planned to include the StateGraph library in the Modelica standard library. It is most useful to combine this libray with the Modelica libraries

- **Modelica.Blocks.Logical** that provides components available in PLCs (programmable logic controllers).
- **UserInteraction** that provides components to interactively communicate with models in a running simulation.

Copyright © 1998-2007, Modelica Association and DLR

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
UsersGuide	User's Guide of StateGraph Library
Examples	Examples to demonstrate the usage of the components of the StateGraph library
Interfaces	Connectors and partial models
InitialStep	Initial step (= step that is active when simulation starts)
InitialStepWithSignal	Initial step (= step that is active when simulation starts). Connector 'active' is true when the step is active
Step	Ordinary step (= step that is not active when simulation starts)
StepWithSignal	Ordinary step (= step that is not active when simulation starts). Connector 'active' is true when the step is active
Transition	Transition where the fire condition is set by a modification of variable condition
TransitionWithSignal	Transition where the fire condition is set by a Boolean input signal
Alternative	Alternative splitting of execution path (use component between two steps)
Parallel	Parallel splitting of execution path (use component between two transitions)
PartialCompositeStep	Superclass of a subgraph, i.e., a composite step that has internally a StateGraph
StateGraphRoot	Root of a StateGraph (has to be present on the highest level of a StateGraph)
Temporary	Components that will be provided by other libraries in the future

Modelica.StateGraph.UsersGuide

User's Guide of StateGraph Library

Library **StateGraph** is a **free** Modelica package providing components to model **discrete event** and **reactive** systems in a convenient way. This package contains the **User's Guide** for the library and has the following content:

1. Overview of library gives an overview of the library.
2. A first example demonstrates at hand of a first example how to use this library.
3. An application example demonstrates varies features at hand of the control of a two tank system.
4. Release Notes summarizes the differences between different versions of this library.
5. Literature provides references that have been used to design and implement this library.
6. Contact provides information about the authors of the library as well as acknowledgments.

Package Content

Name	Description
OverView	Overview of library
FirstExample	A first example
ApplicationExample	An application example
ReleaseNotes	Release notes
Literature	Literature
Contact	Contact

Modelica.StateGraph.UsersGuide.OverView

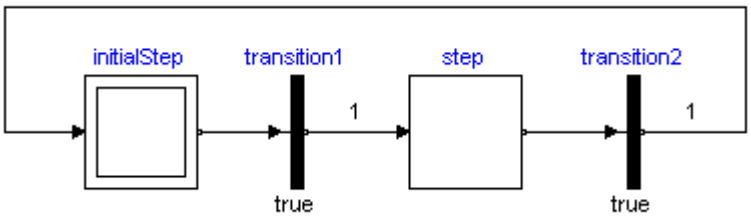
Overview of library

In this section, an overview of the most important features of this library is given.

Steps and Transitions

A **StateGraph** is an enhanced finite state machine. It is based on the JGraphChart method and takes advantage of Modelica features for the "action" language. JGraphChart is a further development of Grafcet to include elements of StateCharts that are not present in Grafcet/Sequential Function Charts. Therefore, the StateGraph library has a similar modeling power as StateCharts but avoids some deficiences of StateCharts.

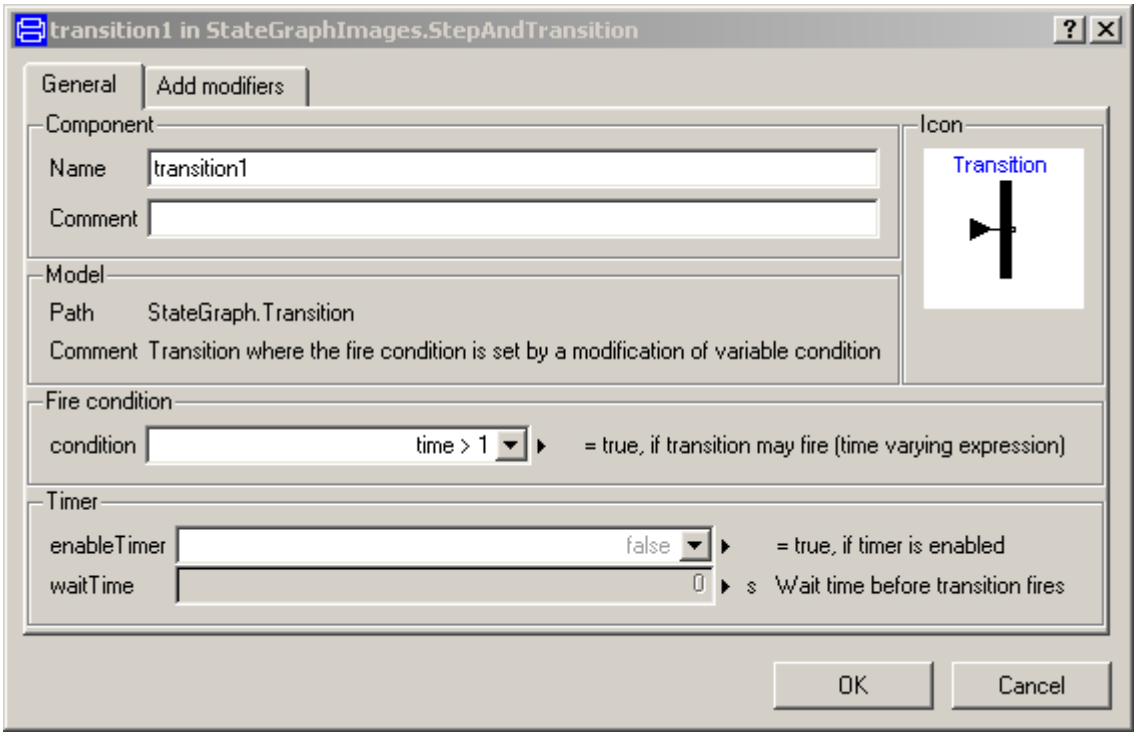
The basic elements of StateGraphs are **steps** and **transitions**:



Steps represent the possible states a StateGraph can have. If a step is active the Boolean variable **active** of the step is **true**. If it is deactivated, **active** = **false**. At the initial time, all "usual" steps are deactivated. The **InitialStep** objects are steps that are activated at the initial time. They are characterized by a double box (see figure above).

Transitions are used to change the state of a StateGraph. When the step connected to the input of a transition is active, the step connected to the output of this transition is deactivated and the transition condition becomes true, then the transition fires. This means that the step connected to the input to the transition is deactivated and the step connected to the output of the transition is activated.

The transition **condition** is defined via the parameter menu of the transition object. Clicking on object "transition1" in the above figure, results in the following menu:



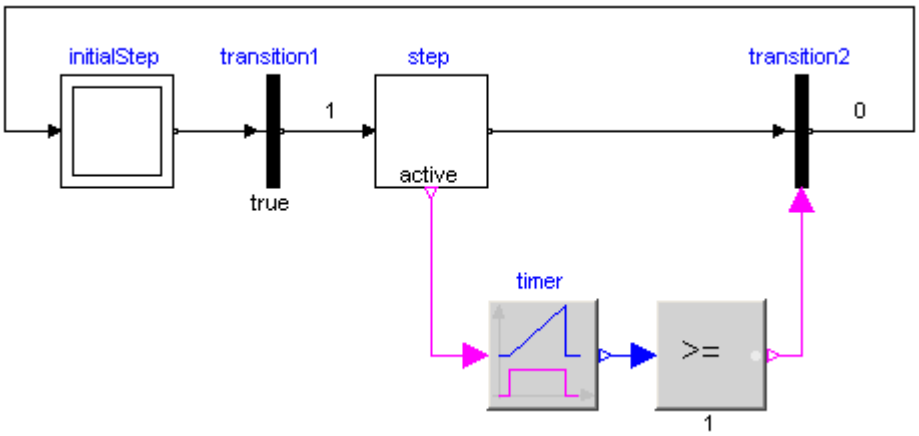
In the input field "**condition**", any type of time varying Boolean expression can be given (in Modelica notation, this is a modification of the time varying variable **condition**). Whenever this condition is true, the transition can fire. Additionally, it is possible to activate a timer, via **enableTimer** (see menu above) and provide a **waitTime**. In this case the firing of the transition is delayed according to the provided waitTime. The transition condition and the waitTime are displayed in the transition icon.

In the above example, the simulation starts at **initialStep**. After 1 second, **transition1** fires and **step1** becomes active. After another second **transition2** fires and **initialStep** becomes again active. After a further second **step1** becomes again active, and so on.

In JGrafcharts, Grafcet and Sequential Function Charts, the network of steps and transitions is drawn from top to bottom. In StateGraphs, no particular direction is defined, since steps and transitions are blocks with input and output connectors that can be arbitrarily placed and connected. Usually, it is most practical to define the network from left to right, as in the example above, since then it is easy to read the labels on the icons.

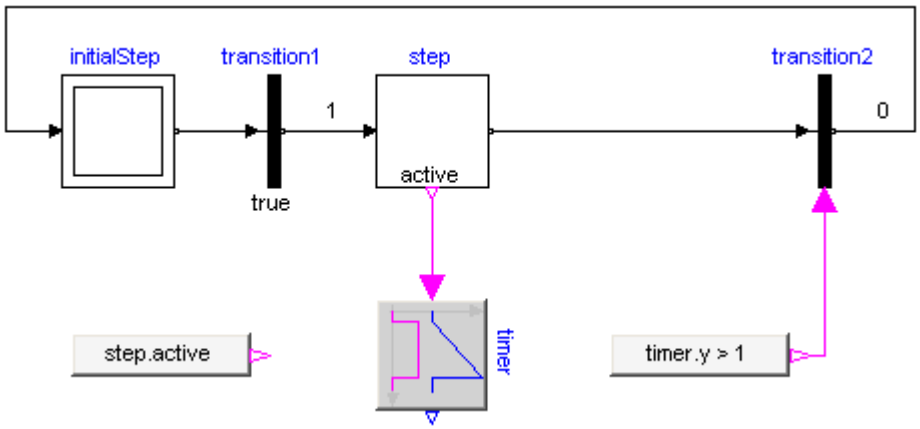
Conditions and Actions

With the block **TransitionWithSignal**, the firing condition can be provided as Boolean input signal, instead as entry in the menu of the transition. An example is given in the next figure:



Component "step" is an instance of "StepWithSignal" that is a usual step where the active flag is available as Boolean output signal. To this output, component "Timer" from library "Modelica.Blocks.Logical" is connected. It measures the time from the time instant where the Boolean input (i.e., the active flag of the step) became true upto the current time instant. The timer is connected to a comparison component. The output is true, once the timer reaches 1 second. This signal is used as condition input of the transition. As a result, "transition2" fires, once step "step" has been active for 1 second. Of course, any other Modelica block with a Boolean output signal can be connected to the condition input of such a transition block as well.

Conditions of a transition can either be computed by a network of logical blocks from the Logical library as in the figure above, or via the "SetBoolean" component any type of logical expression can be defined in textual form, as shown in the next figure:



With the block "**SetBoolean**", a time varying expression can be provided as modification to the output signal **y** (see block with icon text "timer.y > 1" in the figure above). The output signal can be in turn connected to the condition input of a TransitionWithSignal block.

The "**SetBoolean**" block can also be used to compute a Boolean signal depending on the active step. In the figure above, the output of the block with the icon text "step.active" is true, when "step" is active, otherwise it is false (note, the icon text of "SetBoolean" displays the modification of the output signal "y"). This signal can then be used to compute desired **actions** in the physical systems model. For example, if a **valve** shall be open, when the StateGraph is in "step1" or in "step4", a "SetBoolean" block may be connected to the valve model using the following condition:

```
valve = step1.active or step2.active
```

Via the Modelica operators **edge**(..) and **change**(..), conditions depending on rising and falling edges of Boolean expressions can be used when needed.

In JGrafcharts, Grafcet, Sequential Function Charts and StateCharts, **actions** are formulated **within a step**.

Such actions are distinguished as **entry**, **normal**, **exit** and **abort** actions. For example, a valve might be opened by an entry action of a step and might be closed by an exit action of the same step. In StateGraphs, this is (fortunately) **not possible** due to Modelicas "single assignment rule" that requires that every variable is defined by exactly one equation. Instead, the approach explained above is used. For example, via the "SetBoolean" component, the valve variable is set to true when the StateGraph is in particular steps.

This feature of a StateGraph is **very useful**, since it allows a Modelica translator to **guarantee** that a given StateGraph has always **deterministic** behaviour without conflicts. In the other methodologies this is much more cumbersome. For example, if two steps are executed in parallel and both step actions modify the same variable, the result is either non-deterministic or non-obvious rules have to be defined which action takes priority. In a StateGraph, such a situation is detected by the translator resulting in an error, since there are two equations to compute one variable. Additional benefits of the StateGraph approach are:

- A JGrafchart or a StateChart need to potentially access variables in a step that are defined in higher hierarchical levels of a model. Therefore, mostly **global variables** are used in the whole network, even if the network is structured hierarchically. In StateGraphs this is not necessary, since the physical systems outside of a StateGraph might access the step or transition state via a hierarchical name. This means that **no global variables** are needed, because the local variables in the StateGraph are accessed from local variables outside of the StateGraph.
- It is simpler for a user to understand the information that is provided in the non-graphical definition, since every variable is defined at exactly one place. In the other methodologies, the setting and re-setting of the same variable is cluttered within the whole network.

To emphasize this important difference between these methodologies, consider the case that a state machine has the following hierarchy:

```
stateMachine.superstate1.superstate2.step1
```

Within "step1" a StateChart would, e.g., access variable "stateMachine.openValve", say as "entry action: openValve = true". This typical usage has the severe drawback that it is not possible to use the hierarchical state "superstate1" as component in another context, because "step1" references a particular name outside of this component.

In a StateGraph, there would be typically a "SetBoolean" component in the "stateMachine" component stating:

```
openValve = superstate1.superstate2.step1.active;
```

As a result, the "superstate1" component can be used in another context, because it does not depend on the environment where it is used.

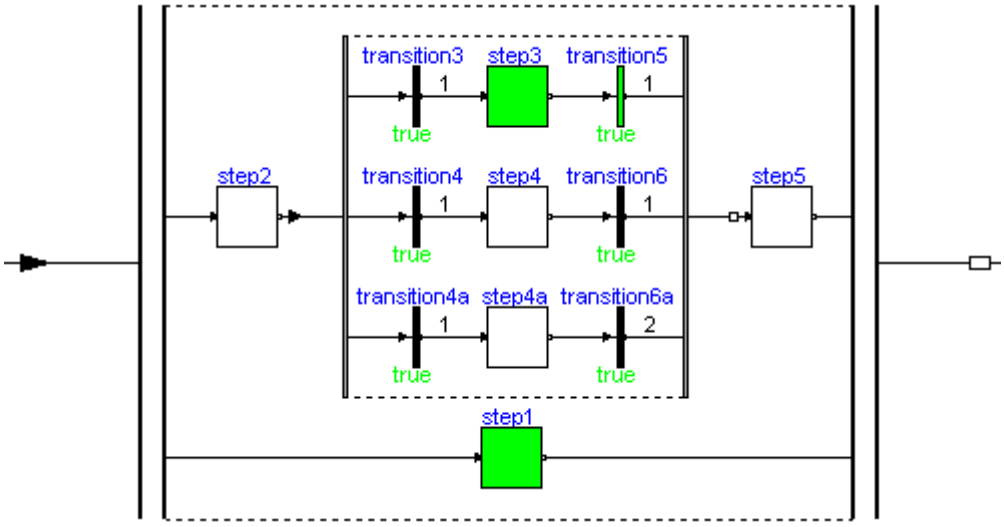
Execution Model

The execution model of a StateGraph follows from its Modelica implementation: Given the states of all steps, i.e., whether a step is active or not active, the equations of all steps, transitions, transition conditions, actions etc. are sorted resulting in an execution sequence to compute essentially the new values of the steps. If conflicts occur, e.g., if there are more equations as variables, of if there are algebraic loops between Boolean variables, an exception is raised. Once all equations have been processed, the **active** variable of all steps are updated to the newly calculated values. Afterwards, the equations are again evaluated. The iteration stops, once no step changes its state anymore, i.e., once no transition fires anymore. Then, simulation continuous until a new event is triggered, (when a relation changes its value).

With the Modelica "sampled(..)" operator, a StateGraph might also be executed within a discrete controller that is called at regular time instants. In a future version of the StateGraph library, this might be more directly supported.

Parallel and Alternative Execution

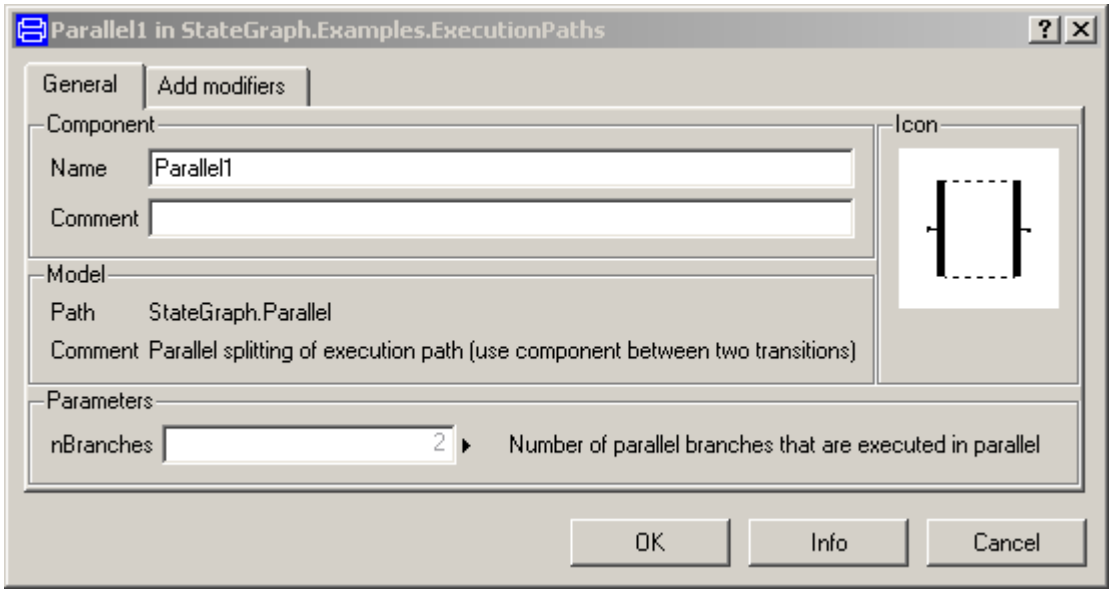
Parallel activities can be defined by component **Parallel** and alternative activities can be defined by component **Alternative**. An example for both components is given in the next figure.



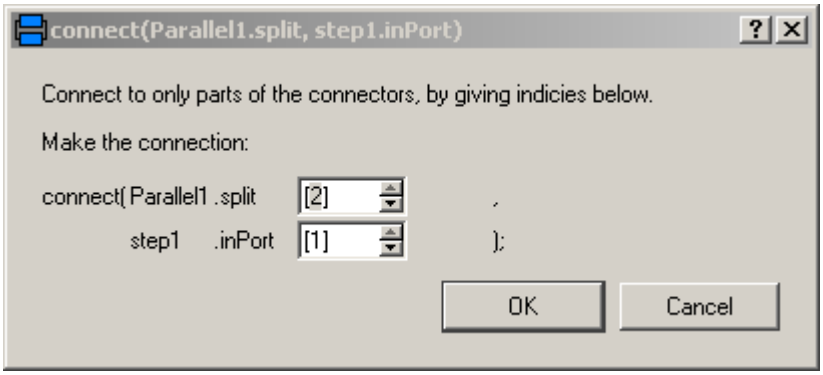
Here, the branch from "step2" to "step5" is executed in parallel to "step1". A transition connected to the output of a parallel branch component can only fire if the final steps in all parallel branches are active simultaneously. The figure above is a screen-shot from the animation of the StateGraph: Whenever a step is active or a transition can fire, the corresponding component is marked in **green** color.

The three branches within "step2" to "step5" are executed alternatively, depending which transition condition of "transition3", "transition4", "transition4a" fires first. Since all three transitions fire after 1 second, they are all candidates for the active branch. If two or more transitions would fire at the same time instant, a priority selection is made: The alternative and parallel components have a vector of connectors. Every branch has to be connected to exactly one entry of the connector vector. The entry with the lowest number has the highest priority.

Parallel, Alternative and Step components have vectors of connectors. The dimensions of these vectors are set in the corresponding parameter menu. E.g. in a "Parallel" component:

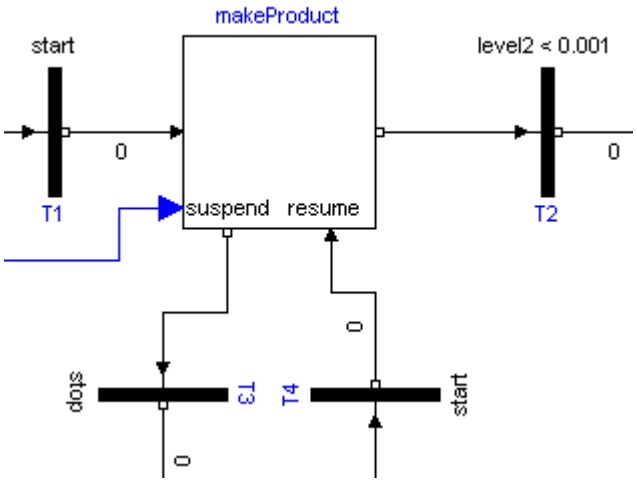


Currently in Dymola the following menu pops up, when a branch is connected to a vector of components in order to define the vector index to which the component shall be connected:



CompositeSteps, Suspend and Resume Port

A StateGraph can be hierarchically structured by using a **CompositeStep**. This is a component that inherits from **PartialCompositeStep**. An example is given in the next figure (from Examples.ControlledTanks):



The CompositeStep component contains a local StateGraph that is entered by one or more input transitions and that is left by one or more output transitions. Also, other needed signals may enter a CompositeStep. The CompositeStep has similiar properties as a "usual" step: The CompositeStep is **active** once at least one step within the CompositeStep is active. Variable **active** defines the state of the CompositeStep.

Additionally, a CompositeStep has a **suspend** port. Whenever the transition connected to this port fires, the CompositeStep is left at once. When leaving the CompositeStep via the suspend port, the internal state of the CompositeStep is saved, i.e., the active flags of all steps within the CompositeStep. The CompositeStep might be entered via its **resume** port. In this case the internal state from the suspend transition is reconstructed and the CompositeStep continues the execution that it had before the suspend transition fired (this corresponds to the history ports of StateCharts or JGrafCharts).

A CompositeStep may contain other CompositeSteps. At every level, a CompositeStep and all of its content can be left via its suspend ports (actually, there is a vector of suspend connectors, i.e., a CompositeStep might be aborted due to different transitions).

Modelica.StateGraph.UsersGuide.FirstExample

A first example

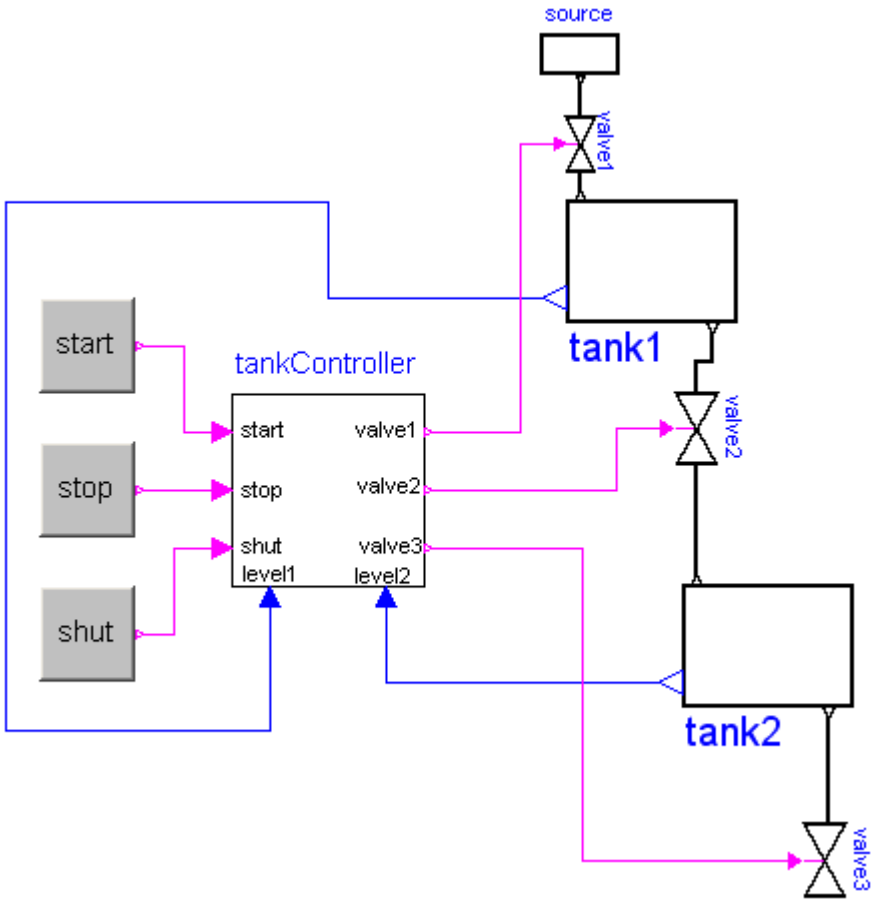
A first example will be given here (not yet done).

Modelica.StateGraph.UsersGuide.ApplicationExample

An application example

In this section a more realistic, still simple, application example is given, to demonstrate various features of the StateGraph library. This example shows the control of a two tank system from the master thesis of Isolde Dressler (see literature).

In the following figure the top level of the model is shown. This model is available as StateGraph.Examples.ControlledTanks.



In the right part of the figure, two tanks are shown. At the top part, a large fluid source is present from which fluid can be filled in **tank1** when **valve1** is open. Tank1 can be emptied via **valve2** that is located in the bottom of tank2 and fills a second **tank2** which in turn is emptied via **valve3**. The actual levels of the tanks are measured and are provided as signals **level1** and **level2** to the **tankController**.

The **tankController** is controlled by three buttons, **start**, **stop** and **shut** (for shutdown) that are mutually exclusive. This means that whenever one button is pressed (i.e., its state is **true**) then the other two buttons are not pressed (i.e., their states are **false**). When button **start** is pressed, the "normal" operation to fill and to empty the two tanks is processed:

1. Valve 1 is opened and tank 1 is filled.
2. When tank 1 reaches its fill level limit, valve 1 is closed.
3. After a waiting time, valve 2 is opened and the fluid flows from tank 1 into tank 2.
4. When tank 1 is empty, valve 2 is closed.
5. After a waiting time, valve 3 is opened and the fluid flows out of tank 2
6. When tank 2 is empty, valve 3 is closed

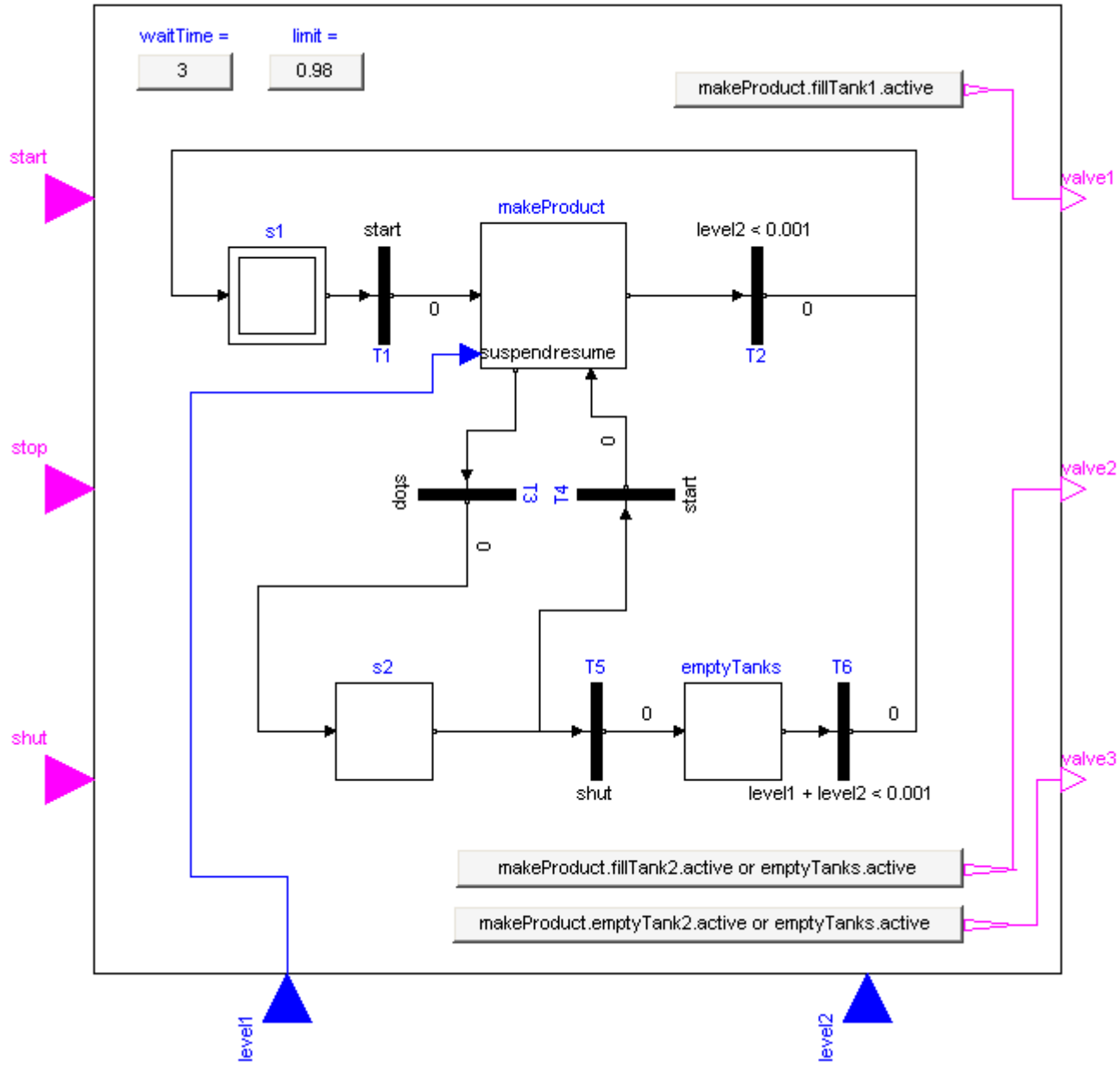
The above "normal" process can be influenced by the following buttons:

- Button **start** starts the above process. When this button is pressed after a "stop" or "shut" operation,

the process operation continues. .

- Button **stop** stops the above process by closing all valves. Then, the controller waits for further input (either "start" or "shut" operation).
- Button **shut** is used to shutdown the process, by emptying at once both tanks. When this is achieved, the process goes back to its start configuration. Clicking on "start", restarts the process.

The implementation of the **tankController** is shown in the next figure:



When the "**start**" button is pressed, the stateGraph is within the CompositeStep "**makeProduct**". During normal operation this CompositeStep is only left, once tank2 is empty. Afterwards, the CompositeStep is at once re-entered.

When the "**stop**" button is pressed, the "makeProduct" CompositeStep is at once terminated via the "**suspend**" port and the stateGraph waits in step "**s2**" for further commands. When the "**start**" button is pressed, the CompositeStep is re-entered via its **resume** port and the "normal" operation continues at the state where it was aborted by the suspend transition. If the "**shut**" button is pressed, the stateGraph waits in the "**emptyTanks**" step, until both tanks are empty and then waits at the initial step "**s1**" for further input.

The opening and closing of valves is **not** directly defined in the stateGraph. Instead via the "**setValveX**" components, the Boolean state of the valves are computed. For example, the output y of "setValve2" is computed as:

```
y = makeProduct.fillTank2.active or emptyTanks.active
```

i.e., valve2 is open, when step "makeProduct.fillTank2 or when step "emptyTanks" is active. Otherwise, valve2 is closed.

Modelica.StateGraph.UsersGuide.ReleaseNotes

Release notes

Version 0.87, 2004-06-23

- Included in Modelica standard library 2.0 Beta 1 with the new block connectors. Changed all the references to the block connectors and the Logical library correspondingly.

Version 0.86, 2004-06-20

- New components "Alternative" and "Parallel" for alternative and parallel execution paths.
- A step has now a vector of input and output connectors in order that multiple connections to and from a step are possible
- Removed components "AlternativeSplit", "AlternativeJoin", "ParallelSplit" and "ParallelJoin" since the newly introduced components ("Alternative", "Parallel", vector connectors of steps) have the same modeling power but are safer and more convenient.
- Removed the timer in a step (attach instead Logical.Timer to the "active" port of a "StepWithSignal" component). Note, that in most cases it is more convenient and more efficient to use the built-in timer of a transition.
- Component "StepInitial" renamed to "InitialStep".
- New component "Timer" within sublibrary Logical.
- Updated and improved documentation of the library.

Version 0.85, 2004-06-17

- Renamed "MacroStep" to "CompositeStep" and the ports of the MacroStep from "abort" to "suspend" and "histoy" to "resume".
- Nested "CompositeStep" components are supported, based on the experimental feature of nested inner/outer components introduced by Dymola. This means that CompositeSteps can be suspended and resumed at every level.
- New example "Examples.ShowExceptions" to demonstrate the new feature of nested CompositeSteps.
- New package "Logical". It contains all components of ModelicaAdditions.Blocks.Logical, but with new block connectors and nicer icons. Additionally, logical blocks are also added.
- Improved icons for several components of the StateGraph library.

Version 0.83, 2004-05-21

- The "abort" and "history" connectors are no longer visible in the diagram layer of a CompositeStep since it is not allowed to connect to them in a CompositeStep.
- Made the diagram/icon size of a CompositeStep smaller (from 200/-200 to 150/-150).
- Improved icons for "SetBoolean/SetInteger/SetReal" components.
- Renamed "ParameterReal" to "SetRealParameter".

Version 0.82, 2004-05-18

Implemented a first version that is provided to other people.

Modelica.StateGraph.UsersGuide.Literature

Literature

The StateGraph library is based on the following references:

Arzen K.-E. (2004):
JGrafchart User Manual. Version 1.5. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, Feb. 13

Dressler I. (2004):
Code Generation From JGrafchart to Modelica. Master thesis, supervisor: Karl-Erik Arzen, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, March 30

Elmqvist H., Mattsson S.E., Otter M. (2001):
Object-Oriented and Hybrid Modeling in Modelica. Journal Europeen des systemes automatises (JESA), Volume 35 - n. 1.

Mosterman P., Otter M., Elmqvist H. (1998):
Modeling Petri Nets as Local Constraint Equations for Hybrid Systems using Modelica. SCSC'98, Reno, Nevada, USA, Society for Computer Simulation International, pp. 314-319.

Modelica.StateGraph.UsersGuide.Contact

Contact

Main Author:
Martin Otter
Deutsches Zentrum für Luft und Raumfahrt e.V. (DLR)
Institut für Robotik und Mechatronik
Abteilung für Entwurfsorientierte Regelungstechnik
Postfach 1116
D-82230 Wessling
Germany
email: Martin.Otter@dlr.de

Acknowledgements:

- The development of this library was strongly motivated by the master thesis of Isolde Dressler (see literature), in which a compiler from JGrafChart to Modelica was designed and implemented. This project was supervised by Karl-Erik Arzen from Departement of Automatic Control, Lund Institut of Technology, Lund, Sweden.
- This library profits also from the experience gained in the focused research program (Schwerpunktprogramm) "Continuous-Discrete Dynamic Systems" (KONDISK), sponsored by the Deutsche Forschungsgemeinschaft under grants OT174/1-2 and EN152/22-2. This support is most gratefully acknowledged.
- The implementation of the basic components of this library by describing finite state machines with equations is based on (Elmqvist, Mattsson and Otter, 2001), which in turn uses ideas from (Mosterman, Otter and Elmqvist, 1998), see literature

---

Modelica.StateGraph.Examples

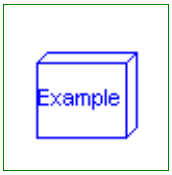
Examples to demonstrate the usage of the components of the StateGraph library

Information

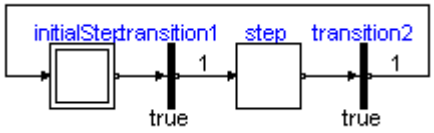
Package Content

Name	Description
FirstExample	A first simple StateGraph example
FirstExample_Variant2	A variant of the first simple StateGraph example
FirstExample_Variant3	A variant of the first simple StateGraph example
ExecutionPaths	Example to demonstrate parallel and alternative execution paths
ShowCompositeStep	Example to demonstrate parallel activities described by a StateGraph
ShowExceptions	Example to demonstrate how a hierarchically structured StateGraph can suspend and resume actions on different levels
ControlledTanks	Demonstrating the controller of a tank filling/emptying system
Utilities	Utility components for the examples

Modelica.StateGraph.Examples.FirstExample

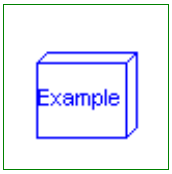


A first simple StateGraph example

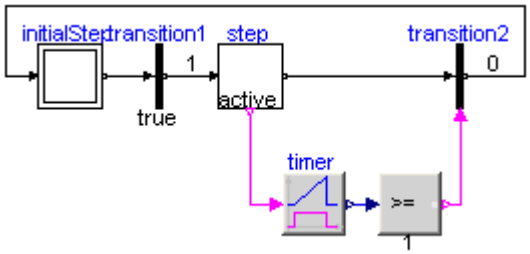


Information

Modelica.StateGraph.Examples.FirstExample_Variant2

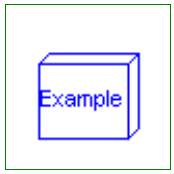


A variant of the first simple StateGraph example

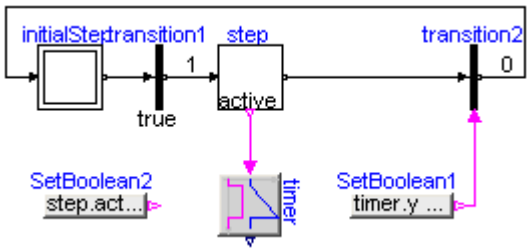


Information

Modelica.StateGraph.Examples.FirstExample_Variant3

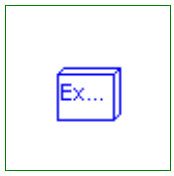


A variant of the first simple StateGraph example

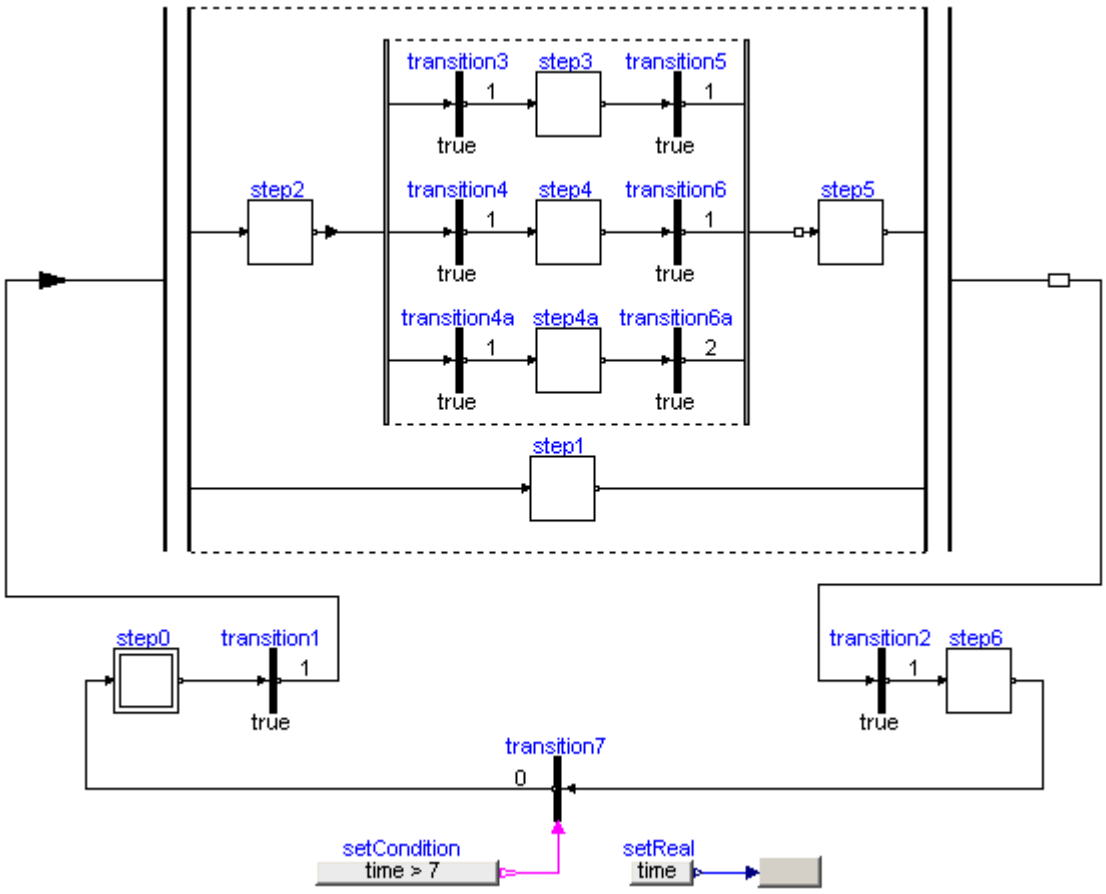


Information

Modelica.StateGraph.Examples.ExecutionPaths



Example to demonstrate parallel and alternative execution paths



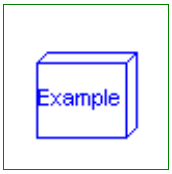
Information

This is an example to demonstrate in which way **parallel** activities can be modelled by a StateGraph. When transition1 fires (after 1 second), two branches are executed in parallel. After 6 seconds the two branches

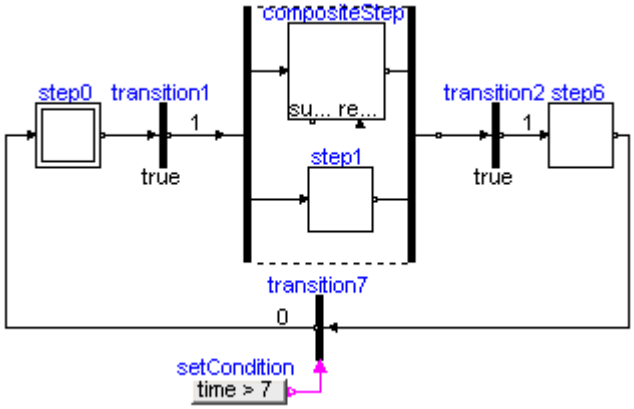
are synchronized in order to arrive at step6.

Before simulating the model, try to figure out whether which branch of the alternative sequence is executed. Note, that alternatives have priorities according to the port index (alternative.split[1] has a higher priority to fire as alternative.split[2]).

Modelica.StateGraph.Examples.ShowCompositeStep



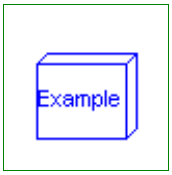
Example to demonstrate parallel activities described by a StateGraph



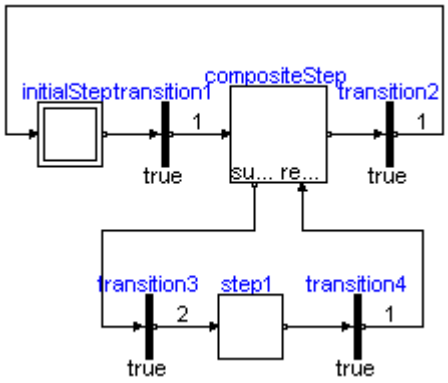
Information

This is the same example as "ExecutionPaths". The only difference is that the alternative paths are included in a "CompositeStep".

Modelica.StateGraph.Examples.ShowExceptions



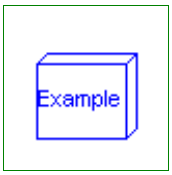
Example to demonstrate how a hierarchically structured StateGraph can suspend and resume actions on different levels



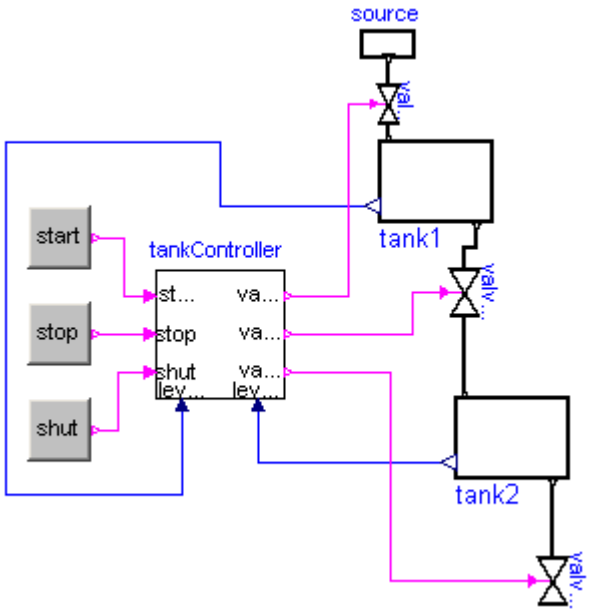
Information

CompositeStep "compositeStep" is a hierarchical StateGraph consisting of two other subgraphs. Whenever component "compositeStep" is suspended, all steps with in "compositeStep" are deactivated. By entering "compositeStep" via its "resume" port, all steps within "compositeStep" are activated according to their setting before leaving the "compositeStep" via its "suspend" port.

Modelica.StateGraph.Examples.ControlledTanks



Demonstrating the controller of a tank filling/emptying system



Information

With this example the controller of a tank filling/emptying system is demonstrated. This example is from Dressler (2004), see Literature. The basic operation is to fill and empty the two tanks:

1. Valve 1 is opened and tank 1 is filled.
2. When tank 1 reaches its fill level limit, valve 1 is closed.
3. After a waiting time, valve 2 is opened and the fluid flows from tank 1 into tank 2.
4. When tank 1 is empty, valve 2 is closed.
5. After a waiting time, valve 3 is opened and the fluid flows out of tank 2
6. When tank 3 is empty, valve 3 is closed

The above "normal" process can be influenced by three buttons:

- Button **start** starts the above process. When this button is pressed after a "stop" or "shut" operation, the process operation continues. .
- Button **stop** stops the above process by closing all valves. Then, the controller waits for further input (either "start" or "shut" operation).
- Button **shut** is used to shutdown the process, by emptying at once both tanks. When this is achieved, the process goes back to its start configuration. Clicking on "start", restarts the process.

Modelica.StateGraph.Examples.Utilities

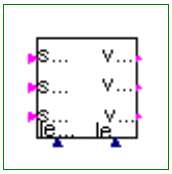
Utility components for the examples

Package Content

Name	Description
TankController	Controller for tank system
MakeProduct	

inflow	Inflow connector (this is a copy from Isolde Dressler's master thesis project)
outflow	Outflow connector (this is a copy from Isolde Dressler's master thesis project)
valve	Simple valve model (this is a copy from Isolde Dressler's master thesis project)
Tank	Simple tank model (this is a copy from Isolde Dressler's master thesis project)
Source	Simple source model (this is a copy from Isolde Dressler's master thesis project)
CompositeStep	
CompositeStep1	
CompositeStep2	

Modelica.StateGraph.Examples.Utilities.TankController



Controller for tank system

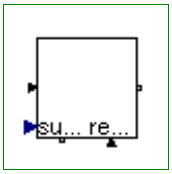
Parameters

Type	Name	Default	Description
SetRealParameter	limit	0.98	Limit level of tank 1
SetRealParameter	waitTime	3	Wait time

Connectors

Type	Name	Description
input BooleanInput	start	
input BooleanInput	stop	
input BooleanInput	shut	
input RealInput	level1	
input RealInput	level2	
output BooleanOutput	valve1	
output BooleanOutput	valve2	
output BooleanOutput	valve3	

Modelica.StateGraph.Examples.Utilities.MakeProduct



Parameters

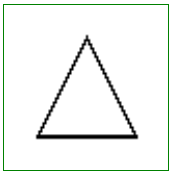
Type	Name	Default	Description
SetRealParameter	limit	0.98	Limit level of tank 1
SetRealParameter	waitTime	3	Wait time
Exception connections			
Integer	nSuspend	1	Number of suspend ports
Integer	nResume	1	Number of resume ports

Connectors

Type	Name	Description
Step_in	inPort	

Step_out	outPort	
CompositeStep_suspend	suspend[nSuspend]	
CompositeStep_resume	resume[nResume]	
input RealInput	level1	

Modelica.StateGraph.Examples.Utilities.inflow

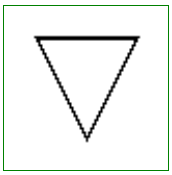


Inflow connector (this is a copy from Isolde Dressler's master thesis project)

Contents

Type	Name	Description
VolumeFlowRate	Fi	inflow [m3/s]

Modelica.StateGraph.Examples.Utilities.outflow

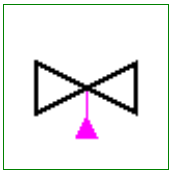


Outflow connector (this is a copy from Isolde Dressler's master thesis project)

Contents

Type	Name	Description
VolumeFlowRate	Fo	outflow [m3/s]
Boolean	open	valve open

Modelica.StateGraph.Examples.Utilities.valve

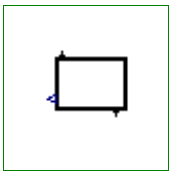


Simple valve model (this is a copy from Isolde Dressler's master thesis project)

Connectors

Type	Name	Description
input BooleanInput	valveControl	
inflow	inflow1	
outflow	outflow1	

Modelica.StateGraph.Examples.Utilities.Tank



Simple tank model (this is a copy from Isolde Dressler's master thesis project)

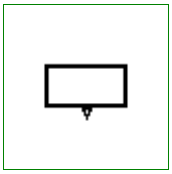
Parameters

Type	Name	Default	Description
Real	A	1	ground area of tank in mÂ²
Real	a	0.2	area of drain hole in mÂ²
Real	hmax	1	max height of tank in m

Connectors

Type	Name	Description
output RealOutput	levelSensor	
inflow	inflow1	
outflow	outflow1	

Modelica.StateGraph.Examples.Utilities.Source



Simple source model (this is a copy from Isolde Dressler's master thesis project)

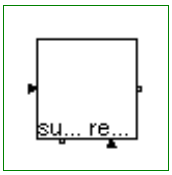
Parameters

Type	Name	Default	Description
Real	maxflow	1	maximal flow out of source

Connectors

Type	Name	Description
outflow	outflow1	

Modelica.StateGraph.Examples.Utilities.CompositeStep



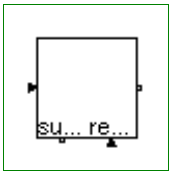
Parameters

Type	Name	Default	Description
Exception connections			
Integer	nSuspend	1	Number of suspend ports
Integer	nResume	1	Number of resume ports

Connectors

Type	Name	Description
Step_in	inPort	
Step_out	outPort	
CompositeStep_suspend	suspend[nSuspend]	
CompositeStep_resume	resume[nResume]	

Modelica.StateGraph.Examples.Utilities.CompositeStep1



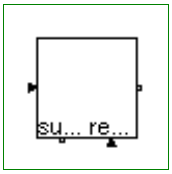
Parameters

Type	Name	Default	Description
Exception connections			
Integer	nSuspend	1	Number of suspend ports
Integer	nResume	1	Number of resume ports

Connectors

Type	Name	Description
Step_in	inPort	
Step_out	outPort	
CompositeStep_suspend	suspend[nSuspend]	
CompositeStep_resume	resume[nResume]	

Modelica.StateGraph.Examples.Utilities.CompositeStep2



Parameters

Type	Name	Default	Description
SetRealParameter	waitTime	2	waiting time in this composite step
Exception connections			
Integer	nSuspend	1	Number of suspend ports
Integer	nResume	1	Number of resume ports

Connectors

Type	Name	Description
Step_in	inPort	
Step_out	outPort	
CompositeStep_suspend	suspend[nSuspend]	
CompositeStep_resume	resume[nResume]	

Modelica.StateGraph.Interfaces

Connectors and partial models

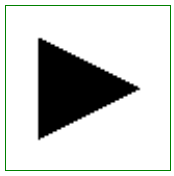
Information

Package Content

Name	Description
Step_in	Input port of a step
Step_out	Output port of a step
Transition_in	Input port of a transition
Transition_out	Output port of a transition
CompositeStep_resume	Input port of a step (used for resume connector of a CompositeStep)
CompositeStep_suspend	Output port of a step (used for suspend connector of a CompositeStep)
CompositeStepStatePort_in	Communication port between a CompositeStep and the ordinary steps within the CompositeStep (suspend/resume are inputs)
CompositeStepStatePort_out	Communication port between a CompositeStep and the ordinary steps within the CompositeStep (suspend/resume are outputs)
PartialStep	Partial step with one input and one output transition port

PartialTransition	Partial transition with input and output connections
PartialStateGraphIcon	Icon for a StateGraph object
CompositeStepState	Communication channel between CompositeSteps and steps in the CompositeStep

Modelica.StateGraph.Interfaces.Step_in



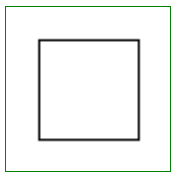
Input port of a step

Information

Contents

Type	Name	Description
Boolean	occupied	true, if step is active
Boolean	set	true, if transition fires and step is activated

Modelica.StateGraph.Interfaces.Step_out



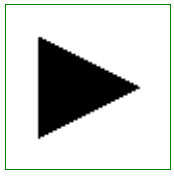
Output port of a step

Information

Contents

Type	Name	Description
Boolean	available	true, if step is active
Boolean	reset	true, if transition fires and step is deactivated

Modelica.StateGraph.Interfaces.Transition_in



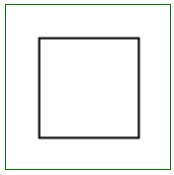
Input port of a transition

Information

Contents

Type	Name	Description
Boolean	available	true, if step connected to the transition input is active
Boolean	reset	true, if transition fires and the step connected to the transition input is deactivated

Modelica.StateGraph.Interfaces.Transition_out



Output port of a transition

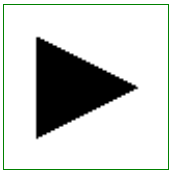
Information

Contents

Type	Name	Description
Boolean	occupied	true, if step connected to the transition output is active
Boolean	set	true, if transition fires and step connected to the transition output becomes active

---

Modelica.StateGraph.Interfaces.CompositeStep_resume



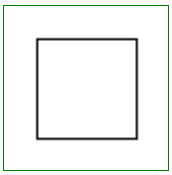
Input port of a step (used for resume connector of a CompositeStep)

Information

Contents

Type	Name	Description
Boolean	occupied	true, if step is active
Boolean	set	true, if transition fires and step is activated

Modelica.StateGraph.Interfaces.CompositeStep_suspend



Output port of a step (used for suspend connector of a CompositeStep)

Information

Contents

Type	Name	Description
Boolean	available	true, if step is active
Boolean	reset	true, if transition fires and step is deactived

---

Modelica.StateGraph.Interfaces.CompositeStepStatePort_in

Communication port between a CompositeStep and the ordinary steps within the CompositeStep (suspend/resume are inputs)

Information

Contents

Type	Name	Description
Boolean	suspend	= true, if suspend transition of CompositeStep fires
Boolean	resume	= true, if resume transition of CompositeStep fires
flow Real	activeSteps	Number of active steps in the CompositeStep

---

Modelica.StateGraph.Interfaces.CompositeStepStatePort_out

Communication port between a CompositeStep and the ordinary steps within the CompositeStep (suspend/resume are outputs)

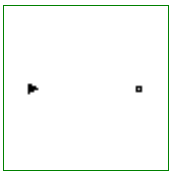
Information

Contents

Type	Name	Description
Boolean	suspend	= true, if suspend transition of CompositeStep fires
Boolean	resume	= true, if resume transition of CompositeStep fires
flow Real	activeSteps	Number of active steps in the CompositeStep

---

Modelica.StateGraph.Interfaces.PartialStep



Partial step with one input and one output transition port

Information

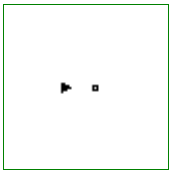
Parameters

Type	Name	Default	Description
Integer	nIn	1	Number of input connections
Integer	nOut	1	Number of output connections

Connectors

Type	Name	Description
Step_in	inPort[nIn]	Vector of step input connectors
Step_out	outPort[nOut]	Vector of step output connectors

Modelica.StateGraph.Interfaces.PartialTransition



Partial transition with input and output connections

Information

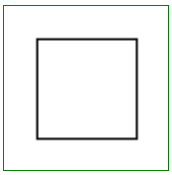
Parameters

Type	Name	Default	Description
Timer			
Boolean	enableTimer	false	= true, if timer is enabled
Time	waitTime	0	Wait time before transition fires [s]

Connectors

Type	Name	Description
Transition_in	inPort	Vector of transition input connectors
Transition_out	outPort	Vector of transition output connectors

Modelica.StateGraph.Interfaces.PartialStateGraphIcon



Icon for a StateGraph object

Information

Modelica.StateGraph.Interfaces.CompositeStepState

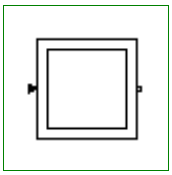
Communication channel between CompositeSteps and steps in the CompositeStep

Information

Connectors

Type	Name	Description
CompositeStepStatePort_out	subgraphStatePort	

Modelica.StateGraph.InitialStep



Initial step (= step that is active when simulation starts)

Information

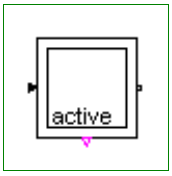
Parameters

Type	Name	Default	Description
Integer	nIn	1	Number of input connections
Integer	nOut	1	Number of output connections
Boolean	localActive	active	= true if step is active, otherwise the step is not active

Connectors

Type	Name	Description
Step_in	inPort[nIn]	Vector of step input connectors
Step_out	outPort[nOut]	Vector of step output connectors

Modelica.StateGraph.InitialStepWithSignal



Initial step (= step that is active when simulation starts). Connector 'active' is true when the step is active

Information

Parameters

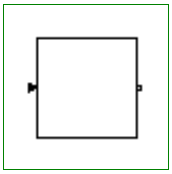
Type	Name	Default	Description
Integer	nIn	1	Number of input connections

Integer	nOut	1	Number of output connections
Boolean	localActive	active	= true if step is active, otherwise the step is not active

Connectors

Type	Name	Description
Step_in	inPort[nIn]	Vector of step input connectors
Step_out	outPort[nOut]	Vector of step output connectors
output BooleanOutput	active	

Modelica.StateGraph.Step



Ordinary step (= step that is not active when simulation starts)

Information

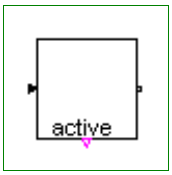
Parameters

Type	Name	Default	Description
Integer	nIn	1	Number of input connections
Integer	nOut	1	Number of output connections
Boolean	localActive	active	= true if step is active, otherwise the step is not active

Connectors

Type	Name	Description
Step_in	inPort[nIn]	Vector of step input connectors
Step_out	outPort[nOut]	Vector of step output connectors

Modelica.StateGraph.StepWithSignal



Ordinary step (= step that is not active when simulation starts). Connector 'active' is true when the step is active

Information

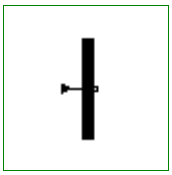
Parameters

Type	Name	Default	Description
Integer	nIn	1	Number of input connections
Integer	nOut	1	Number of output connections
Boolean	localActive	active	= true if step is active, otherwise the step is not active

Connectors

Type	Name	Description
Step_in	inPort[nIn]	Vector of step input connectors
Step_out	outPort[nOut]	Vector of step output connectors
output BooleanOutput	active	

Modelica.StateGraph.Transition



Transition where the fire condition is set by a modification of variable condition

Information

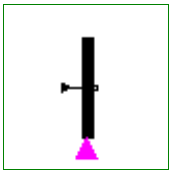
Parameters

Type	Name	Default	Description
Fire condition			
Boolean	condition	true	= true, if transition may fire (time varying expression)
Timer			
Boolean	enableTimer	false	= true, if timer is enabled
Time	waitTime	0	Wait time before transition fires [s]

Connectors

Type	Name	Description
Transition_in	inPort	Vector of transition input connectors
Transition_out	outPort	Vector of transition output connectors

Modelica.StateGraph.TransitionWithSignal



Transition where the fire condition is set by a Boolean input signal

Information

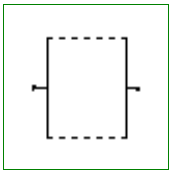
Parameters

Type	Name	Default	Description
Timer			
Boolean	enableTimer	false	= true, if timer is enabled
Time	waitTime	0	Wait time before transition fires [s]

Connectors

Type	Name	Description
input BooleanInput	condition	
Transition_in	inPort	Vector of transition input connectors
Transition_out	outPort	Vector of transition output connectors

Modelica.StateGraph.Alternative



Alternative splitting of execution path (use component between two steps)

Information

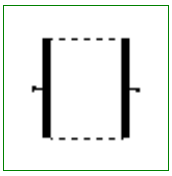
Parameters

Type	Name	Default	Description
Integer	nBranches	2	Number of alternative branches

Connectors

Type	Name	Description
Transition_in	inPort	
Transition_out	outPort	
Step_in_forAlternative	join[nBranches]	
Step_out_forAlternative	split[nBranches]	

Modelica.StateGraph.Parallel



Parallel splitting of execution path (use component between two transitions)

Information

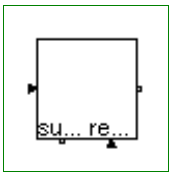
Parameters

Type	Name	Default	Description
Integer	nBranches	2	Number of parallel branches that are executed in parallel

Connectors

Type	Name	Description
Step_in	inPort	
Step_out	outPort	
Transition_in_forParallel	join[nBranches]	
Transition_out_forParallel	split[nBranches]	

Modelica.StateGraph.PartialCompositeStep



Superclass of a subgraph, i.e., a composite step that has internally a StateGraph

Information

Parameters

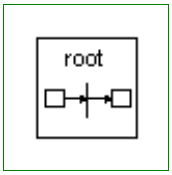
Type	Name	Default	Description
Exception connections			
Integer	nSuspend	1	Number of suspend ports
Integer	nResume	1	Number of resume ports

Connectors

Type	Name	Description

Step_in	inPort	
Step_out	outPort	
CompositeStep_suspend	suspend[nSuspend]	
CompositeStep_resume	resume[nResume]	

Modelica.StateGraph.StateGraphRoot



Root of a StateGraph (has to be present on the highest level of a StateGraph)

Information

On the highest level of a StateGraph, an instance of StateGraphRoot has to be present. If it is not within in a model, it is automatically included by a Modelica translator due to an appropriate annotation. Practically, this means that it need not be present in a StateGraph model.

The StateGraphRoot object is needed, since all Step objects have an "outer" reference to communicate with the "nearest" CompositeStep (which inherits from PartialCompositeStep), especially to abort a CompositeStep via the "suspend" port. Even if no "CompositeStep" is present, on highest level a corresponding "inner" definition is needed and is provided by the StateGraphRoot object.

Connectors

Type	Name	Description
CompositeStepStatePort_out	subgraphStatePort	

Modelica.StateGraph.Temporary

Components that will be provided by other libraries in the future

Information

This library is just temporarily present. The components of this library will be present in the future in the Modelica standard library (with the new block connectors) and in the UserInteraction library that is currently under development.

Package Content

Name	Description
SetRealParameter	Define Real parameter (GUI not yet satisfactory)
anyTrue	Returns true, if at least on element of the Boolean input vector is true
allTrue	Returns true, if all elements of the Boolean input vector are true
RadioButton	Button that sets its output to true when pressed and is reset when an element of 'reset' becomes true
NumericValue	Show value of Real input signal dynamically
IndicatorLamp	Dynamically show Boolean input signal (false/true = white/green color)

Types and constants

```
type SetRealParameter = Real "Define Real parameter (GUI not yet satisfactory)";
```

Modelica.StateGraph.Temporary.anyTrue

Returns true, if at least on element of the Boolean input vector is true

Information

Inputs

Type	Name	Default	Description
Boolean	b[:]		

Outputs

Type	Name	Description
Boolean	result	

Modelica.StateGraph.Temporary.allTrue

Returns true, if all elements of the Boolean input vector are true

Information

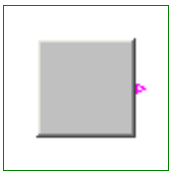
Inputs

Type	Name	Default	Description
Boolean	b[:]		

Outputs

Type	Name	Description
Boolean	result	

Modelica.StateGraph.Temporary.RadioButton



Button that sets its output to true when pressed and is reset when an element of 'reset' becomes true

Information

Parameters

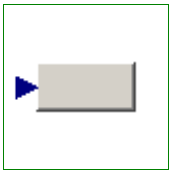
Type	Name	Default	Description
Time	buttonTimeTable[:]		Time instants where button is pressend and released [s]
Time varying expressions			
Boolean	reset[:]	{false}	Reset button to false, if an element of reset becomes true

Connectors

Type	Name	Description

output BooleanOutput	on	

Modelica.StateGraph.Temporary.NumericValue



Show value of Real input signal dynamically

Information

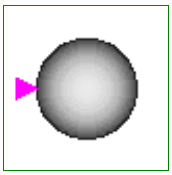
Parameters

Type	Name	Default	Description
Integer	precision	3	Number of significant digits to be shown
Boolean	hideConnector	false	= true, if connector is not shown in the dynamic object diagram
RealInput	Value		Real value to be shown in icon

Connectors

Type	Name	Description
input RealInput	Value	Real value to be shown in icon

Modelica.StateGraph.Temporary.IndicatorLamp



Dynamically show Boolean input signal (false/true = white/green color)

Information

Connectors

Type	Name	Description
input BooleanInput	u	

Modelica.Thermal

Library of thermal system components to model heat transfer and simple thermo-fluid pipe flow

Information

This package contains libraries to model heat transfer and fluid heat flow.

Package Content

Name	Description
FluidHeatFlow	Simple components for 1-dimensional incompressible thermo-fluid flow models
HeatTransfer	Library of 1-dimensional heat transfer with lumped elements

Modelica.Thermal.FluidHeatFlow

Simple components for 1-dimensional incompressible thermo-fluid flow models

Information

This package contains very simple-to-use components to model coolant flows as needed to simulate cooling e.g. of electric machines:

- Components: components like different types of pipe models
- Examples: some test examples
- Interfaces: definition of connectors and partial models (containing the core thermodynamic equations)
- Media: definition of media properties
- Sensors: various sensors for pressure, temperature, volume and enthalpy flow
- Sources: various flow sources

Variables used in connectors:

- Pressure p
- flow MassFlowRate m_flow
- SpecificEnthalpy h
- flow EnthalpyFlowRate H_flow

EnthalpyFlowRate means the Enthalpy = $cp_{constant}$ * m * T that is carried by the medium's flow.

Limitations and assumptions:

- Splitting and mixing of coolant flows (media with the same cp) is possible.
- Reversing the direction of flow is possible.
- The medium is considered to be incompressible.
- No mixtures of media is taken into consideration.
- The medium may not change its phase.
- Medium properties are kept constant.
- Pressure changes are only due to pressure drop and geodetic height differnence rho*g*h (if h > 0).
- A user-defined part (0..1) of the friction losses (V_flow*dp) are fed to the medium.
- **Note:** Connected flowPorts have the same temperature (mixing temperature)!
  Since mixing may occur, the outlet temperature may be different from the connector's temperature.
  Outlet temperature is defined by variable T of the corresponding component.

Further development:

- Additional components like tanks (if needed)

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description

Examples	Examples that demonstrate the usage of the FluidHeatFlow components
Components	Basic components (pipes, valves)
Interfaces	Connectors and partial models
Media	Medium properties
Sensors	Ideal sensors to measure port properties
Sources	Ideal fluid sources, e.g., ambient, volume flow

Modelica.Thermal.FluidHeatFlow.Examples

Examples that demonstrate the usage of the FluidHeatFlow components

Information

This package contains test examples:

- 1.SimpleCooling: heat is dissipated through a media flow
- 2.ParallelCooling: two heat sources dissipate through merged media flows
- 3.IndirectCooling: heat is disspated through two cooling cycles
- 4.PumpAndValve: demonstrates usage of an IdealPump and a Valve
- 5.PumpDropOut: demonstrates shutdown and restart of a pump
- 6.ParallelPumpDropOut: demonstrates shutdown and restart of a pump in a parallel circuit
- 7.OneMass: cooling of a mass (thermal capacity) by a coolant flow
- 8.TwoMass: cooling of two masses (thermal capacities) by two parallel coolant flows

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

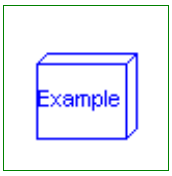
*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

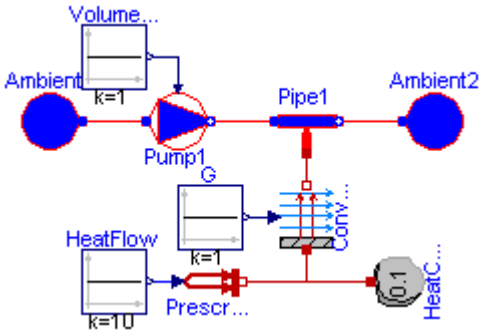
Name	Description
SimpleCooling	Example: simple cooling circuit
ParallelCooling	Example: coolig circuit with parallel branches
IndirectCooling	Example: indirect cooling circuit
PumpAndValve	Example: cooling circuit with pump and valve
PumpDropOut	Example: cooling circuit with drop out of pump

ParallelPumpDropOut	Example: cooling circuit with parallel branches and drop out of pump
OneMass	Example: ccoling of one hot mass
TwoMass	Example: cooling of two hot masses
Utilities	Utility models for examples

Modelica.Thermal.FluidHeatFlow.Examples.SimpleCooling



Example: simple cooling circuit



Information

1st test example: SimpleCooling

A prescribed heat source dissipates its heat through a thermal conductor to a coolant flow. The coolant flow is taken from an ambient and driven by a pump with prescribed mass flow.

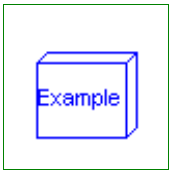
Results:

output	explanation	formula	actual steady-state value
dTSource	Source over Ambient	dtCoolant + dtToPipe	20 K
dTtoPipe	Source over Coolant	Losses / ThermalConductor.G	10 K
dTCoolant	Coolant's temperature increase	Losses * cp * massFlow	10 K

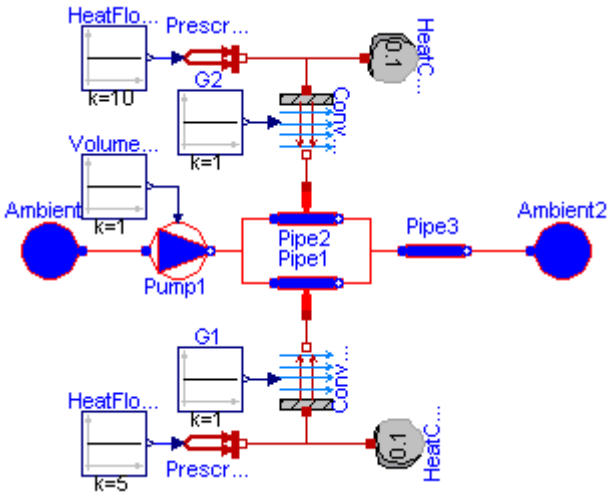
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Cooling medium
CelsiusTemperature	TAmb	20	Ambient temperature [degC]

Modelica.Thermal.FluidHeatFlow.Examples.ParallelCooling



Example: coolig circuit with parallel branches



Information

2nd test example: ParallelCooling

Two prescribed heat sources dissipate their heat through thermal conductors to coolant flows. The coolant flow is taken from an ambient and driven by a pump with prescribed mass flow, then splitted into two coolant flows connected to the two heat sources, and afterwards merged. Splitting of coolant flows is determined by pressure drop characteristic of the two pipes.

Results:

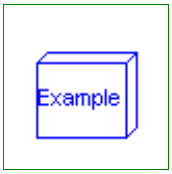
output	explanation	formula	actual steady-state value
dTSource1	Source1 over Ambient	dTCoolant1 + dTtoPipe1	15 K
dTtoPipe1	Source1 over Coolant1	Losses1 / ThermalConductor1.G	5 K
dTCoolant1	Coolant's temperature increase	Losses * cp * totalMassFlow/2	10 K
dTSource2	Source2 over Ambient	dTCoolant2 + dTtoPipe2	30 K
dTtoPipe2	Source2 over Coolant2	Losses2 / ThermalConductor2.G	10 K
dTCoolant2	Coolant's temperature increase	Losses * cp * totalMassFlow/2	20 K
dTmixedCoolant	mixed Coolant's temperature increase	(dTCoolant1+dTCoolant2)/2	15 K

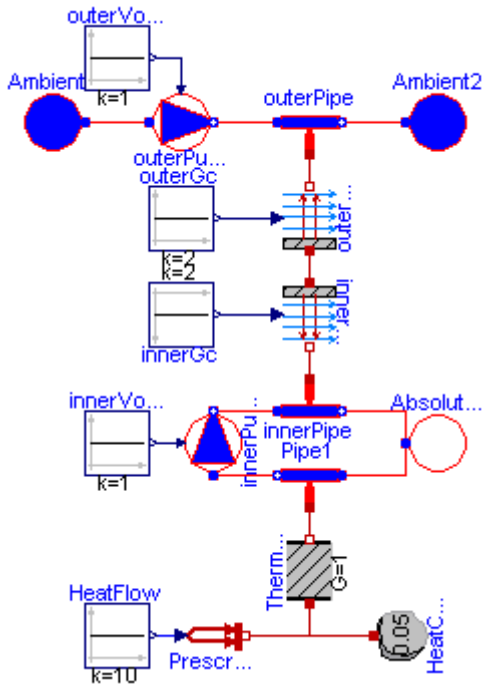
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Cooling medium
CelsiusTemperature	TAmb	20	Ambient temperature [degC]

Modelica.Thermal.FluidHeatFlow.Examples.IndirectCooling

Example: indirect cooling circuit





Information

3rd test example: IndirectCooling

A prescribed heat sources dissipates its heat through a thermal conductor to the inner coolant cycle. It is necessary to define the pressure level of the inner coolant cycle. The inner coolant cycle is coupled to the outer coolant flow through a thermal conductor.
Inner coolant's temperature rise near the source is the same as temperature drop near the cooler.
Results:

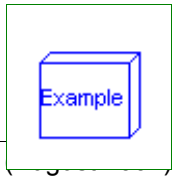
output	explanation	formula	actual steady-state value
dTSource	Source over Ambient	dtouterCoolant + dtCooler + dTinnerCoolant + dtToPipe	40 K
dTtoPipe	Source over inner Coolant	Losses / ThermalConductor.G	10 K
dTinnerColant	inner Coolant's temperature increase	Losses * cp * innerMassFlow	10 K
dTCooler	Cooler's temperature rise between inner and outer pipes	Losses * (innerGc + outerGc)	10 K
dTouterColant	outer Coolant's temperature increase	Losses * cp * outerMassFlow	10 K

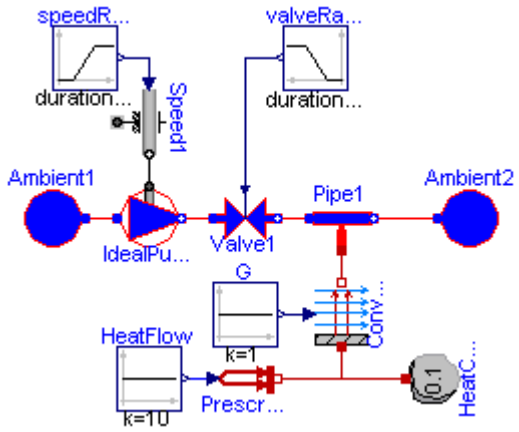
Parameters

Type	Name	Default	Description
Medium	outerMedium	FluidHeatFlow.Media.Medium()	Outer medium
Medium	innerMedium	FluidHeatFlow.Media.Medium()	Inner medium
CelsiusTemperature	TAmb	20	Ambient temperature [degC]

Modelica.Thermal.FluidHeatFlow.Examples.PumpAndValve

Example: cooling circuit with pump and valve





Information

4th test example: PumpAndValve

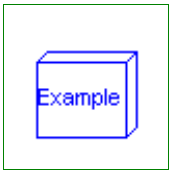
The pump is running with half speed for 0.4 s, afterwards with full speed (using a ramp of 0.1 s).
The valve is half open for 0.9 s, afterwards full open (using a ramp of 0.1 s).
You may try to:

- drive the pump with variable speed and let the valve full open to regulate the volume flow rate of coolant
- drive the pump with constant speed and throttle the valve to regulate the volume flow rate of coolant

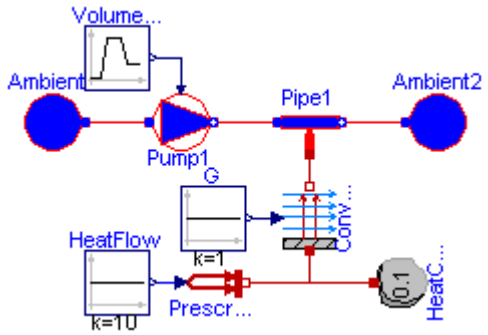
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Cooling medium
CelsiusTemperature	TAmb	20	Ambient temperature [degC]

Modelica.Thermal.FluidHeatFlow.Examples.PumpDropOut



Example: cooling circuit with drop out of pump



Information

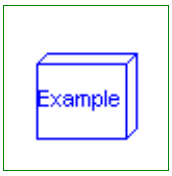
5th test example: PumpDropOut

Same as 1st test example, but with a drop out of the pump:
The pump is running for 0.2 s, then shut down (using a ramp of 0.2 s) for 0.2 s, then started again (using a ramp of 0.2 s).

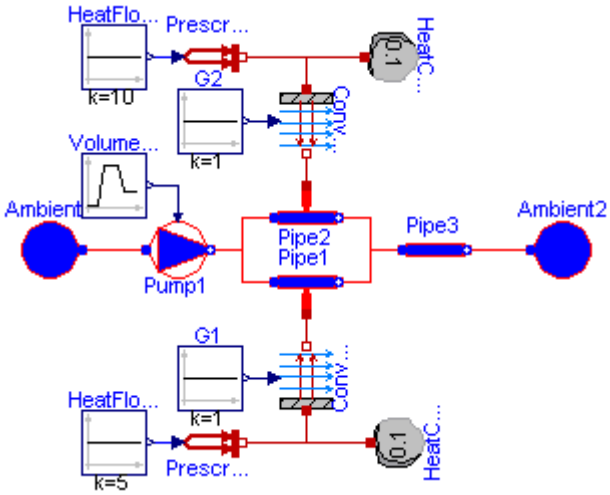
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Cooling medium
CelsiusTemperature	TAmb	20	Ambient temperature [degC]

Modelica.Thermal.FluidHeatFlow.Examples.ParallelPumpDropOut



Example: cooling circuit with parallel branches and drop out of pump



Information

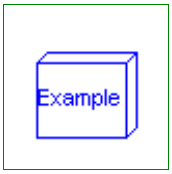
6th test example: ParallelPumpDropOut

Same as 2nd test example, but with a drop out of the pump:
The pump is running for 0.2 s, then shut down (using a ramp of 0.2 s) for 0.2 s, then started again (using a ramp of 0.2 s).

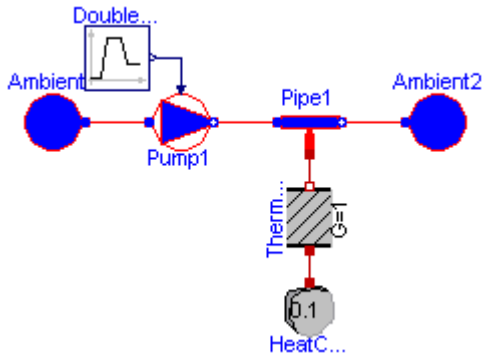
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Cooling medium
CelsiusTemperature	TAmb	20	Ambient temperature [degC]

Modelica.Thermal.FluidHeatFlow.Examples.OneMass



Example: ccoling of one hot mass



Information

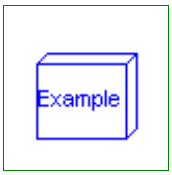
7th test example: OneMass

A thermal capacity is coupled with a coolant flow. Different inital temperatures of thermal capacity and pipe's coolant get ambient's temperature, the time behaviour depending on coolant flow.

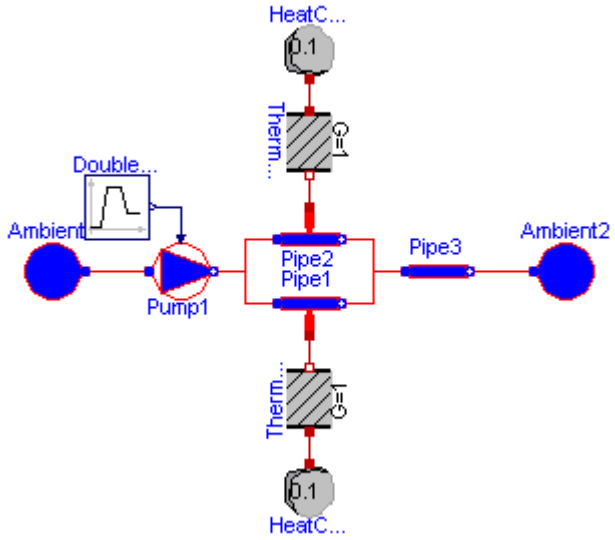
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Cooling medium
CelsiusTemperature	TAmb	20	Ambient temperature [degC]
CelsiusTemperature	TMass	40	Inital temperature of mass [degC]

Modelica.Thermal.FluidHeatFlow.Examples.TwoMass



Example: cooling of two hot masses



Information

8th test example: TwoMass

Two thermal capacities are coupled with two parallel coolant flow. Different inital temperatures of thermal capacities and pipe's coolants get ambient's temperature, the time behaviour depending on coolant flow.

Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Cooling medium
CelsiusTemperature	TAmb	20	Ambient temperature [degC]
CelsiusTemperature	TMass1	40	Inital temperature of mass1 [degC]
CelsiusTemperature	TMass2	60	Inital temperature of mass2 [degC]

Modelica.Thermal.FluidHeatFlow.Examples.Utilities

Utility models for examples

Information

This package contains utility components used for the test examples.

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

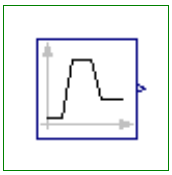
Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
DoubleRamp	Ramp going up and down

Modelica.Thermal.FluidHeatFlow.Examples.Utilities.DoubleRamp



Ramp going up and down

Information

Block generating the sum of two ramps.

Parameters

Type	Name	Default	Description
Real	offset	1	Offset of ramps
Time	startTime	0.2	StartTime of 1st ramp [s]

Time	interval	0.2	Interval between end of 1st and beginning of 2nd ramp [s]
Ramp 1			
Real	height_1	-1	Height of ramp
Time	duration_1	0.2	Duration of ramp [s]
Ramp 2			
Real	height_2	1	Height of ramp
Time	duration_2	0.2	Duration of ramp [s]

Connectors

Type	Name	Description
output RealOutput	y	Connector of Real output signal

Modelica.Thermal.FluidHeatFlow.Components

Basic components (pipes, valves)

Information

This package contains components:

- pipe without heat exchange
- pipe with heat exchange
- valve (simple controlled valve)

Pressure drop is taken from partial model SimpleFriction.
Thermodynamic equations are defined in partial models (package Partials).

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

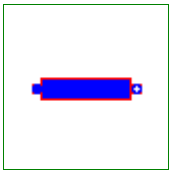
Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
IsolatedPipe	Pipe without heat exchange
HeatedPipe	Pipe with heat exchange
Valve	Simple valve

Modelica.Thermal.FluidHeatFlow.Components.IsolatedPipe



Pipe without heat exchange

Information

Pipe without heat exchange.
Thermodynamic equations are defined by Partials.TwoPortMass(Q_flow = 0).
Note: Setting parameter m (mass of medium within pipe) to zero leads to neglection of temperature transient cv*m*der(T).

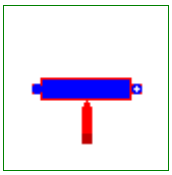
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	1	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions.. .	Initial temperature of medium [K]
Length	h_g	0	Geodetic height (heigth difference from flowPort_a to flowPort_b) [m]
Simple Friction			
VolumeFlowRate	V_flowLaminar	0.1	Laminar volume flow [m3/s]
Pressure	dpLaminar	0.1	Laminar pressure drop [Pa]
VolumeFlowRate	V_flowNominal	1	Nominal volume flow [m3/s]
Pressure	dpNominal	1	Nominal pressure drop [Pa]
Real	frictionLoss	0	Part of friction losses fed to medium

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Components.HeatedPipe



Pipe with heat exchange

Information

Pipe with heat exchange.
Thermodynamic equations are defined by Partials.TwoPort.
Q_flow is defined by heatPort.Q_flow.
Note: Setting parameter m (mass of medium within pipe) to zero leads to neglection of temperature transient cv*m*der(T).
Note: Injecting heat into a pipe with zero massflow causes temperature rise defined by storing heat in medium's mass.

Parameters

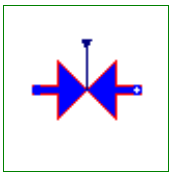
Type	Name	Default	Description

Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	1	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions...	Initial temperature of medium [K]
Real	tapT	1	Defines temperature of heatPort between inlet and outlet temperature
Length	h_g	0	Geodetic height (heigth difference from flowPort_a to flowPort_b) [m]
Simple Friction			
VolumeFlowRate	V_flowLaminar	0.1	Laminar volume flow [m3/s]
Pressure	dpLaminar	0.1	Laminar pressure drop [Pa]
VolumeFlowRate	V_flowNominal	1	Nominal volume flow [m3/s]
Pressure	dpNominal	1	Nominal pressure drop [Pa]
Real	frictionLoss	0	Part of friction losses fed to medium

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	
HeatPort_a	heatPort	

Modelica.Thermal.FluidHeatFlow.Components.Valve



Simple valve

Information

Simple controlled valve.
Standard characteristic Kv=*f* (y) is given at standard conditions (dp0, rho0),

- `either linear : Kv/Kv1 = Kv0/Kv1 + (1-Kv0/Kv1) * y/Y1`
- `or exponential: Kv/Kv1 = Kv0/Kv1 * exp[ln(Kv1/Kv0) * y/Y1]`

`where:`

- `Kv0 ... min. flow @ y = 0`
- `Y1 .... max. valve opening`
- `Kv1 ... max. flow @ y = Y1`

Flow resistance under real conditions is calculated by
`V_flow**2 * rho / dp = Kv(y)**2 * rho0 / dp0`

Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	0	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions...	Initial temperature of medium [K]
Real	frictionLoss	0	Part of friction losses fed to medium
Standard characteristic			

Boolean	LinearCharacteristic	true	Type of characteristic
Real	y1	1	Max. valve opening
VolumeFlowRate	Kv1	1	Max. flow @ y = y1 [m3/s]
Real	kv0	0.01	Leakage flow / max.flow @ y = 0
Pressure	dp0	1	Standard pressure drop [Pa]
Density	rho0	10	Standard medium's density [kg/m3]

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	
input RealInput	y	

Modelica.Thermal.FluidHeatFlow.Interfaces

Connectors and partial models

Information

This package contains connectors and partial models:

- FlowPort: basic definition of the connector.
- FlowPort_a & FlowPort_b: same as FlowPort with different icons to differentiate direction of flow
- package Partials (defining basic thermodynamic equations)

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
FlowPort	conector flow port
FlowPort_a	Filled flow port (used upstream)
FlowPort_b	Hollow flow port (used downstream)
Partials	Partial models

Modelica.Thermal.FluidHeatFlow.Interfaces.FlowPort

conector flow port

Information

Basic definition of the connector.
Variables:

- Pressure p
- flow MassFlowRate m_flow
- Specific Enthalpy h
- flow EnthaplyFlowRate H_flow

If ports with different media are connected, the simulation is asserted due to the check of parameter.

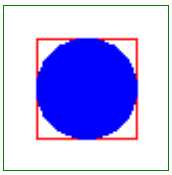
Parameters

Type	Name	Default	Description
Medium	medium		Medium in the connector

Contents

Type	Name	Description
Medium	medium	Medium in the connector
Pressure	p	[Pa]
flow MassFlowRate	m_flow	[kg/s]
SpecificEnthalpy	h	[J/kg]
flow EnthalpyFlowRate	H_flow	[W]

Modelica.Thermal.FluidHeatFlow.Interfaces.FlowPort_a



Filled flow port (used upstream)

Information

Same as FlowPort, but icon allows to differentiate direction of flow.

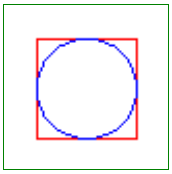
Parameters

Type	Name	Default	Description
Medium	medium		Medium in the connector

Contents

Type	Name	Description
Medium	medium	Medium in the connector
Pressure	p	[Pa]
flow MassFlowRate	m_flow	[kg/s]
SpecificEnthalpy	h	[J/kg]
flow EnthalpyFlowRate	H_flow	[W]

Modelica.Thermal.FluidHeatFlow.Interfaces.FlowPort_b



Hollow flow port (used downstream)

Information

Same as FlowPort, but icon allows to differentiate direction of flow.

Parameters

Type	Name	Default	Description
Medium	medium		Medium in the connector

Contents

Type	Name	Description
Medium	medium	Medium in the connector
Pressure	p	[Pa]
flow MassFlowRate	m_flow	[kg/s]
SpecificEnthalpy	h	[J/kg]
flow EnthalpyFlowRate	H_flow	[W]

Modelica.Thermal.FluidHeatFlow.Interfaces.Partials

Partial models

Information

This package contains partial models, defining in a very compact way the basic thermodynamic equations used by the different components.

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
SimpleFriction	Simple friction model
▪ ▫ TwoPort	Partial model of two port

Ambient	Partial model of ambient
AbsoluteSensor	Partial model of absolute sensor
RelativeSensor	Partial model of relative sensor
FlowSensor	Partial model of flow sensor

Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.SimpleFriction

Simple friction model

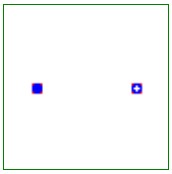
Information

Definition of relationship between pressure drop and volume flow rate:
-V_flowLaminar < VolumeFlow < +V_flowLaminar: laminar i.e. linear dependency of pressure drop on volume flow.
-V_flowLaminar > VolumeFlow or VolumeFlow < +V_flowLaminar: turbulent i.e. quadratic dependency of pressure drop on volume flow.
Linear and quadratic dependency are coupled smoothly at V_flowLaminar / dpLaminar.
Quadratic dependency is defined by nominal volume flow and pressure drop (V_flowNominal / dpNominal).
See also sketch at diagram layer.

Parameters

Type	Name	Default	Description
Simple Friction			
VolumeFlowRate	V_flowLaminar	0.1	Laminar volume flow [m3/s]
Pressure	dpLaminar	0.1	Laminar pressure drop [Pa]
VolumeFlowRate	V_flowNominal	1	Nominal volume flow [m3/s]
Pressure	dpNominal	1	Nominal pressure drop [Pa]
Real	frictionLoss	0	Part of friction losses fed to medium

Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.TwoPort



Partial model of two port

Information

Partial model with two flowPorts.
Possible heat exchange with the ambient is defined by Q_flow; setting this = 0 means no energy exchange.
Setting parameter m (mass of medium within pipe) to zero leads to neglection of temperature transient cv*m*der(T).
Mixing rule is applied.
Parameter 0 < tapT < 1 defines temperature of heatPort between medium's inlet and outlet temperature.

Parameters

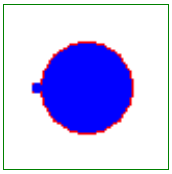
Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	1	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions...	Initial temperature of medium [K]
Real	tapT	1	Defines temperature of heatPort between inlet and

			outlet temperature

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.Ambient



Partial model of ambient

Information

Partial model of (Infinite) ambient, defines pressure and temperature.

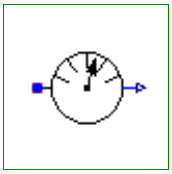
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Ambient's medium

Connectors

Type	Name	Description
FlowPort_a	flowPort	

Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.AbsoluteSensor



Partial model of absolute sensor

Information

Partial model for an absolute sensor (pressure/temperature).
Pressure, mass flow, temperature and enthalpy flow of medium are not affected.

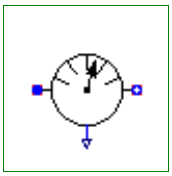
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Sensor's medium

Connectors

Type	Name	Description
FlowPort_a	flowPort	
output RealOutput	y	

Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.RelativeSensor



Partial model of relative sensor

Information

Partial model for a relative sensor (pressure drop/temperature difference).
Pressure, mass flow, temperature and enthalpy flow of medium are not affected.

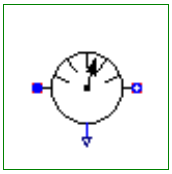
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Sensor's medium

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	
output RealOutput	y	

Modelica.Thermal.FluidHeatFlow.Interfaces.Partials.FlowSensor



Partial model of flow sensor

Information

Partial model for a flow sensor (mass flow/heat flow).
Pressure, mass flow, temperature and enthalpy flow of medium are not affected, but mixing rule is applied.

Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	
output RealOutput	y	

Modelica.Thermal.FluidHeatFlow.Media

Medium properties

Information

This package contains definitions of medium properties.

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

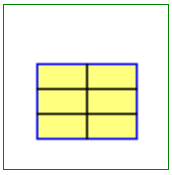
Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Medium	Record containing media properties
Air_30degC	Medium: propeties of air at 30 degC
Air_70degC	Medium: propeties of air at 70 degC
Water	Medium: propeties of water

Modelica.Thermal.FluidHeatFlow.Media.Medium



Record containing media properties

Information

Record containing (constant) medium properties.

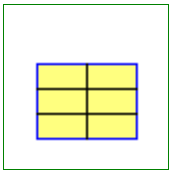
Parameters

Type	Name	Default	Description
Density	rho	1	Density [kg/m3]
SpecificHeatCapacity	cp	1	Specific heat capacity at constant pressure [J/(kg.K)]
SpecificHeatCapacity	cv	1	Specific heat capacity at constant volume [J/(kg.K)]
ThermalConductivity	lamda	1	Thermal conductivity [W/(m.K)]
KinematicViscosity	nue	1	kinematic viscosity [m2/s]

Modelica definition

```
record Medium "Record containing media properties"
  extends Modelica.Icons.Record;
  parameter Modelica.SIunits.Density rho = 1 "Density";
  parameter Modelica.SIunits.SpecificHeatCapacity cp = 1
    "Specific heat capacity at constant pressure";
  parameter Modelica.SIunits.SpecificHeatCapacity cv = 1
    "Specific heat capacity at constant volume";
  parameter Modelica.SIunits.ThermalConductivity lamda = 1
    "Thermal conductivity";
  parameter Modelica.SIunits.KinematicViscosity nue = 1 "kinematic viscosity";
end Medium;
```

Modelica.Thermal.FluidHeatFlow.Media.Air_30degC



Medium: propeties of air at 30 degC

Information

Medium: propeties of air at 30 degC

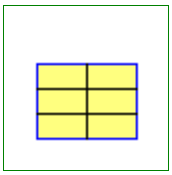
Parameters

Type	Name	Default	Description
Density	rho	1.149	Density [kg/m3]
SpecificHeatCapacity	cp	1007	Specific heat capacity at constant pressure [J/(kg.K)]
SpecificHeatCapacity	cv	720	Specific heat capacity at constant volume [J/(kg.K)]
ThermalConductivity	lamda	0.0264	Thermal conductivity [W/(m.K)]
KinematicViscosity	nue	16.3E-6	kinematic viscosity [m2/s]

Modelica definition

```
record Air_30degC "Medium: propeties of air at 30 degC"
extends Medium(
  rho=1.149,
  cp=1007,
  cv= 720,
  lamda=0.0264,
  nue=16.3E-6);
end Air_30degC;
```

Modelica.Thermal.FluidHeatFlow.Media.Air_70degC



Medium: propeties of air at 70 degC

Information

Medium: propeties of air at 70 degC

Parameters

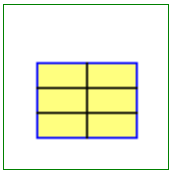
Type	Name	Default	Description
Density	rho	1.015	Density [kg/m3]
SpecificHeatCapacity	cp	1010	Specific heat capacity at constant pressure [J/(kg.K)]
SpecificHeatCapacity	cv	723	Specific heat capacity at constant volume [J/(kg.K)]
ThermalConductivity	lamda	0.0293	Thermal conductivity [W/(m.K)]
KinematicViscosity	nue	20.3E-6	kinematic viscosity [m2/s]

Modelica definition

```
record Air_70degC "Medium: propeties of air at 70 degC"
extends Medium(
  rho=1.015,
  cp=1010,
  cv= 723,
  lamda=0.0293,
```

```
  nue=20.3E-6);
end Air_70degC;
```

Modelica.Thermal.FluidHeatFlow.Media.Water



Medium: propeties of water

Information

Medium: propeties of water

Parameters

Type	Name	Default	Description
Density	rho	995.6	Density [kg/m3]
SpecificHeatCapacity	cp	4177	Specific heat capacity at constant pressure [J/(kg.K)]
SpecificHeatCapacity	cv	4177	Specific heat capacity at constant volume [J/(kg.K)]
ThermalConductivity	lamda	0.615	Thermal conductivity [W/(m.K)]
KinematicViscosity	nue	0.8E-6	kinematic viscosity [m2/s]

Modelica definition

```
record Water "Medium: propeties of water"
extends Medium(
  rho=995.6,
  cp=4177,
  cv=4177,
  lamda=0.615,
  nue=0.8E-6);
end Water;
```

Modelica.Thermal.FluidHeatFlow.Sensors

Ideal sensors to measure port properties

Information

This package contains sensors:

- pSensor: absolute pressure
- TSensor: absolute temperature (Kelvin)
- dpSensor: pressure drop between flowPort_a and flowPort_b
- dTSensor: temperature difference between flowPort_a and flowPort_b
- m_flowSensor: measures mass flow rate
- V_flowSensor: measures volume flow rate
- H_flowSensor: measures enthalpy flow rate

Some of the sensors do not need access to medium properties for measuring, but it is necessary to define the medium in the connector (check of connections).
Thermodynamic equations are defined in partial models (package Interfaces.Partials).
All sensors are considered massless, they do not change mass flow or enthalpy flow.

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

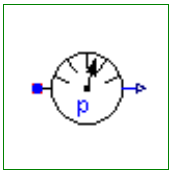
Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
pSensor	Absolute pressure sensor
TSensor	Absolute temperature sensor
dpSensor	Pressure difference sensor
dTSensor	Temperature difference sensor
m_flowSensor	Mass flow sensor
V_flowSensor	Volume flow sensor
H_flowSensor	Enthapy flow sensor

Modelica.Thermal.FluidHeatFlow.Sensors.pSensor



Absolute pressure sensor

Information

pSensor measures the absolute pressure.
Thermodynamic equations are defined by Partials.AbsoluteSensor.

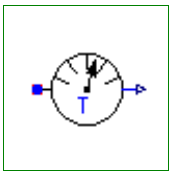
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Sensor's medium

Connectors

Type	Name	Description
FlowPort_a	flowPort	

Modelica.Thermal.FluidHeatFlow.Sensors.TSensor



Absolute temperature sensor

Information

TSensor measures the absolute temperature (Kelvin).
Thermodynamic equations are defined by Partials.AbsoluteSensor.

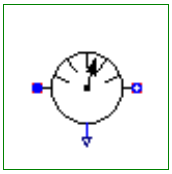
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Sensor's medium

Connectors

Type	Name	Description
FlowPort_a	flowPort	

Modelica.Thermal.FluidHeatFlow.Sensors.dpSensor



Pressure difference sensor

Information

dpSensor measures the pressure drop between flowPort_a and flowPort_b.
Thermodynamic equations are defined by Partials.RelativeSensor.

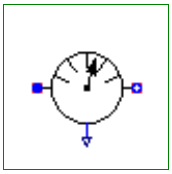
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Sensor's medium

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Sensors.dTSensor



Temperature difference sensor

Information

dTSensor measures the temperature difference between flowPort_a and flowPort_b.
Thermodynamic equations are defined by Partials.RelativeSensor.
Note: Connected flowPorts have the same temperature (mixing temperature)!
Since mixing my occur, the outlet temperature of a component may be different from the connector's temperature.
Outlet temperature is defined by variable T of the corresponding component.

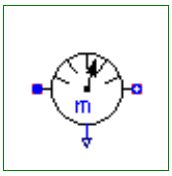
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Sensor's medium

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Sensors.m_flowSensor



Mass flow sensor

Information

m_flowSensor measures the mass flow rate.
Thermodynamic equations are defined by Partials.FlowSensor.

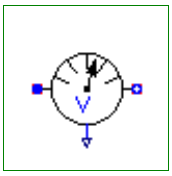
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Sensors.V_flowSensor



Volume flow sensor

Information

V_flowSensor measures the volume flow rate.
Thermodynamic equations are defined by Partials.FlowSensor.

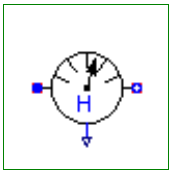
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Sensors.H_flowSensor



Enthapy flow sensor

Information

H_flowSensor measures the enthalpy flow rate.
Thermodynamic equations are defined by Partials.FlowSensor.

Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Sources

Ideal fluid sources, e.g., ambient, volume flow

Information

This package contains different types of sources:

- Ambient with constant pressure and temperature
- Ambient with prescribed pressure and temperature
- AbsolutePressure to define pressure level of a closed cooling cycle.
- Constant and prescribed volume flow
- Constant and prescribed pressure increase
- Simple pump with mechanical flange

Thermodynamic equations are defined in partial models (package Interfaces.Partials).
All fans / pumps are considered without losses, they do not change enthalpy flow.

Main Authors:

Anton Haumer
Technical Consulting & Electrical Engineering
A-3423 St.Andrae-Woerdern, Austria
email: a.haumer@haumer.at

Dr.Christian Kral & Markus Plainer
Österreichisches Forschungs- und Prüfzentrum Arsenal Ges.m.b.H.
arsenal research
Giefinggasse 2
A-1210 Vienna, Austria

Copyright © 1998-2007, Modelica Association, Anton Haumer and arsenal research.

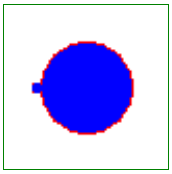
*The Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
Ambient	Ambient with constant properties

PrescribedAmbient	Ambient with prescirbed properties
AbsolutePressure	Defines absolute pressure level
ConstantVolumeFlow	Enforces constant volume flow
PrescribedVolumeFlow	Enforces prescribed volume flow
ConstantPressureIncrease	Enforces constant pressure increase
PrescribedPressureIncrease	Enforces prescribed pressure increase
IdealPump	Model of an ideal pump

Modelica.Thermal.FluidHeatFlow.Sources.Ambient



Ambient with constant properties

Information

(Infinite) ambient with constant pressure and temperature.
Thermodynamic equations are defined by Partials.Ambient.

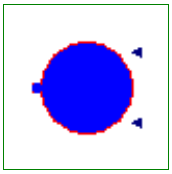
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Ambient's medium
Pressure	p_Ambient	0	Ambient's pressure [Pa]
Temperature	T_Ambient	Modelica.SIunits.Conversions...	Ambient's temperature [K]

Connectors

Type	Name	Description
FlowPort_a	flowPort	

Modelica.Thermal.FluidHeatFlow.Sources.PrescribedAmbient



Ambient with prescirbed properties

Information

(Infinite) ambient with prescribed pressure and temperature.
Thermodynamic equations are defined by Partials.Ambient.

Parameters

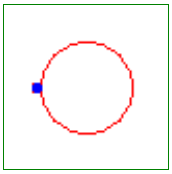
Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Ambient's medium

Connectors

Type	Name	Description
FlowPort_a	flowPort	
input RealInput	p_Ambient	

input RealInput	T_Ambient	

Modelica.Thermal.FluidHeatFlow.Sources.AbsolutePressure



Defines absolute pressure level

Information

AbsolutePressure to define pressure level of a closed cooling cycle. Coolant's mass flow, temperature and enthalpy flow are not affected.

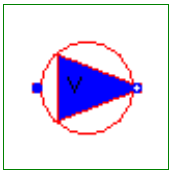
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium
Pressure	p	0	Pressure ground [Pa]

Connectors

Type	Name	Description
FlowPort_a	flowPort	

Modelica.Thermal.FluidHeatFlow.Sources.ConstantVolumeFlow



Enforces constant volume flow

Information

Fan resp. pump with constant volume flow rate. Pressure increase is the response of the whole system.
Coolant's temperature and enthalpy flow are not affected.
Setting parameter m (mass of medium within fan/pump) to zero leads to neglection of temperature transient cv*m*der(T).
Thermodynamic equations are defined by Partials.TwoPort.

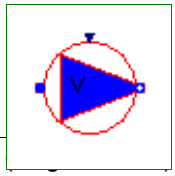
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	1	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions...	Initial temperature of medium [K]
VolumeFlowRate	VolumeFlow	1	Volume flow rate [m3/s]

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Sources.PrescribedVolumeFlow



Enforces prescribed volume flow

Information

Fan resp. pump with prescribed volume flow rate. Pressure increase is the response of the whole system. Coolant's temperature and enthalpy flow are not affected.
Setting parameter m (mass of medium within fan/pump) to zero leads to neglection of temperature transient cv*m*der(T).
Thermodynamic equations are defined by Partials.TwoPort.

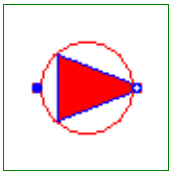
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	1	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions...	Initial temperature of medium [K]

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	
input RealInput	VolumeFlow	

Modelica.Thermal.FluidHeatFlow.Sources.ConstantPressureIncrease



Enforces constant pressure increase

Information

Fan resp. pump with constant pressure increase. Mass resp. volume flow is the response of the whole system. Coolant's temperature and enthalpy flow are not affected.
Setting parameter m (mass of medium within fan/pump) to zero leads to neglection of temperature transient cv*m*der(T).
Thermodynamic equations are defined by Partials.TwoPort.

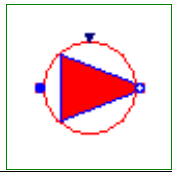
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	1	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions...	Initial temperature of medium [K]
Pressure	PressureIncrease	1	Pressure increase [Pa]

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	

Modelica.Thermal.FluidHeatFlow.Sources.PrescribedPressureIncrease



Enforces prescribed pressure increase

Information

Fan resp. pump with prescribed pressure increase. Mass resp. volume flow is the response of the whole system. Coolant's temperature and enthalpy flow are not affected.
Setting parameter m (mass of medium within fan/pump) to zero leads to neglection of temperature transient cv*m*der(T).
Thermodynamic equations are defined by Partials.TwoPort.

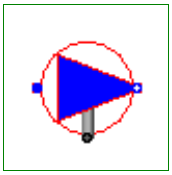
Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	1	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions...	Initial temperature of medium [K]

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	
input RealInput	PressureIncrease	

Modelica.Thermal.FluidHeatFlow.Sources.IdealPump



Model of an ideal pump

Information

Simple fan resp. pump where characteristic is dependent on shaft's speed,
torque * speed = pressure increase * volume flow (without losses)
Pressure increase versus volume flow is defined by a linear function, from dp0(V_flow=0) to V_flow0(dp=0).
The axis intersections vary with speed as follows:

- dp prop. speed^2
- V_flow prop. speed

Coolant's temperature and enthalpy flow are not affected.
Setting parameter m (mass of medium within fan/pump) to zero leads to neglection of temperature transient cv*m*der(T).
Thermodynamic equations are defined by Partials.TwoPort.

Parameters

Type	Name	Default	Description
Medium	medium	FluidHeatFlow.Media.Medium()	Medium in the component
Mass	m	1	Mass of medium [kg]
Temperature	T0	Modelica.SIunits.Conversions...	Initial temperature of medium [K]
Pump characteristic			
AngularVelocity	w_Nominal	1	Nominal speed [rad/s]
Pressure	dp0	2	Max. pressure increase @ V_flow=0 [Pa]
VolumeFlowRate	V_flow0	2	Max. volume flow rate @ dp=0 [m3/s]

Connectors

Type	Name	Description
FlowPort_a	flowPort_a	
FlowPort_b	flowPort_b	
Flange_a	flange_a	

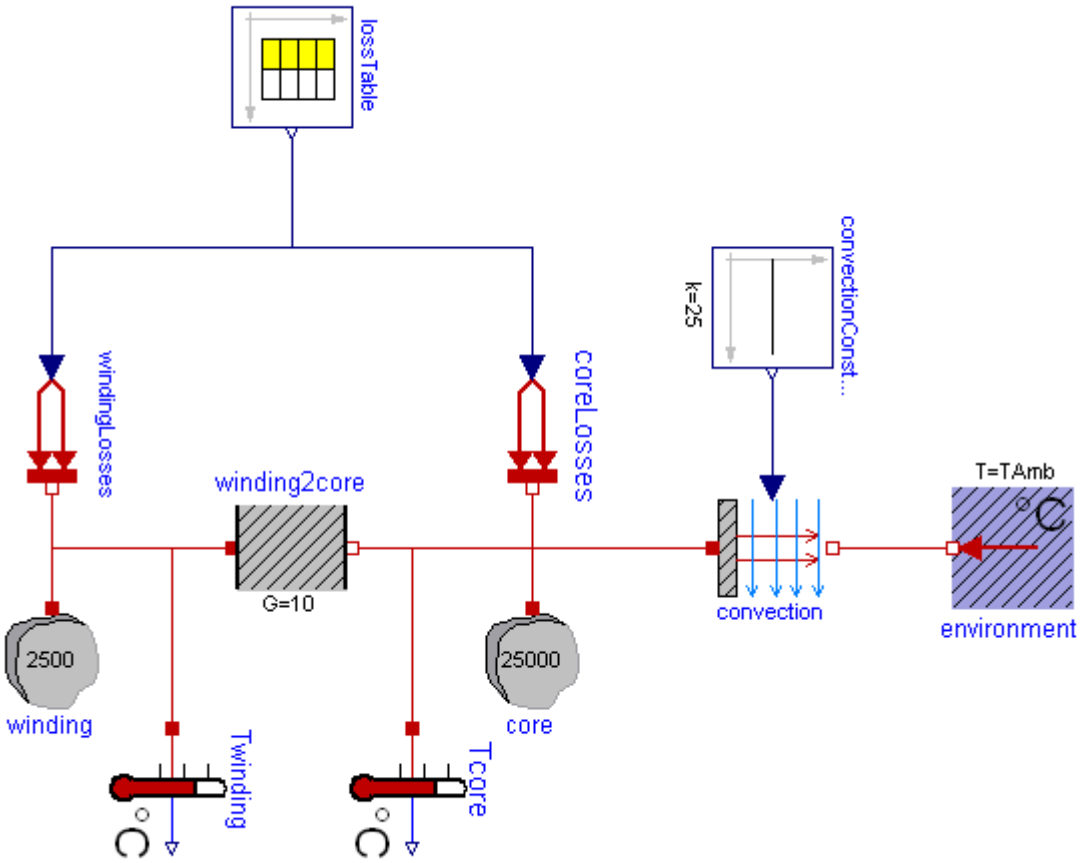
Modelica.Thermal.HeatTransfer

Library of 1-dimensional heat transfer with lumped elements

Information

This package contains components to model **1-dimensional heat transfer** with lumped elements. This allows especially to model heat transfer in machines provided the parameters of the lumped elements, such as the heat capacity of a part, can be determined by measurements (due to the complex geometries and many materials used in machines, calculating the lumped element parameters from some basic analytic formulas is usually not possible).

Example models how to use this library are given in subpackage **Examples**.
For a first simple example, see **Examples.TwoMasses** where two masses with different initial temperatures are getting in contact to each other and arriving after some time at a common temperature.
Examples.ControlledTemperature shows how to hold a temperature within desired limits by switching on and off an electric resistor.
A more realistic example is provided in **Examples.Motor** where the heating of an electrical motor is modelled, see the following screen shot of this example:



The **filled** and **non-filled red squares** at the left and right side of a component represent **thermal ports** (connector HeatPort). Drawing a line between such squares means that they are thermally connected. The variables of a HeatPort connector are the temperature **T** at the port and the heat flow rate **Q_flow** flowing into the component (if Q_flow is positive, the heat flows into the element, otherwise it flows out of the element):

```
Modelica.SIunits.Temperature  T  "absolute temperature at port in Kelvin";
Modelica.SIunits.HeatFlowRate Q_flow  "flow rate at the port in Watt";
```

Note, that all temperatures of this package, including initial conditions, are given in Kelvin. For convenience, in subpackages **HeatTransfer.Celsius**, **HeatTransfer.Fahrenheit** and **HeatTransfer.Rankine** components are provided such that source and sensor information is available in degree Celsius, degree Fahrenheit, or degree Rankine, respectively. Additionally, in package **SIunits.Conversions** conversion functions between the units Kelvin and Celsius, Fahrenheit, Rankine are provided. These functions may be used in the following way:

```
  import SI=Modelica.SIunits;
  import Modelica.SIunits.Conversions.*;
     ...
  parameter SI.Temperature T = from_degC(25);  // convert 25 degree Celsius to
Kelvin
```

There are several other components available, such as AxialConduction (discretized PDE in axial direction), which have been temporarily removed from this library. The reason is that these components reference material properties, such as thermal conductivity, and currently the Modelica design group is discussing a

general scheme to describe material properties.

For technical details in the design of this library, see the following reference:
Michael Tiller (2001): Introduction to Physical Modeling with Modelica. Kluwer Academic Publishers Boston.

Acknowledgements:
Several helpful remarks from the following persons are acknowledged: John Batteh, Ford Motors, Dearborn, U.S.A; Anton Haumer, Technical Consulting & Electrical Engineering, Austria; Ludwig Marvan, VA TECH ELIN EBG Elektronik GmbH, Wien, Austria; Hans Olsson, Dynasim AB, Sweden; Hubertus Tummescheit, Lund Institute of Technology, Lund, Sweden.

Copyright © 2001-2007, Modelica Association, Michael Tiller and DLR.

This Modelica package is free software; it can be redistributed and/or modified under the terms of the Modelica license, see the license conditions and the accompanying disclaimer in the documentation of package Modelica in file "Modelica/package.mo".

Package Content

Name	Description
Examples	Example models to demonstrate the usage of package Modelica.Thermal.HeatTransfer
Interfaces	Connectors and partial models
HeatCapacitor	Lumped thermal element storing heat
ThermalConductor	Lumped thermal element transporting heat without storing it
Convection	Lumped thermal element for heat convection
BodyRadiation	Lumped thermal element for radiation heat transfer
FixedTemperature	Fixed temperature boundary condition in Kelvin
PrescribedTemperature	Variable temperature boundary condition in Kelvin
FixedHeatFlow	Fixed heat flow boundary condition
PrescribedHeatFlow	Prescribed heat flow boundary condition
TemperatureSensor	Absolute temperature sensor in Kelvin
RelTemperatureSensor	Relative Temperature sensor
HeatFlowSensor	Heat flow rate sensor
Celsius	Components with Celsius input and/or output
Fahrenheit	Components with Fahrenheit input and/or output
Rankine	Components with Rankine input and/or output

Modelica.Thermal.HeatTransfer.Examples

Example models to demonstrate the usage of package Modelica.Thermal.HeatTransfer

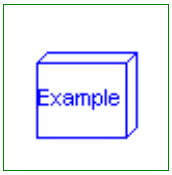
Information

Package Content

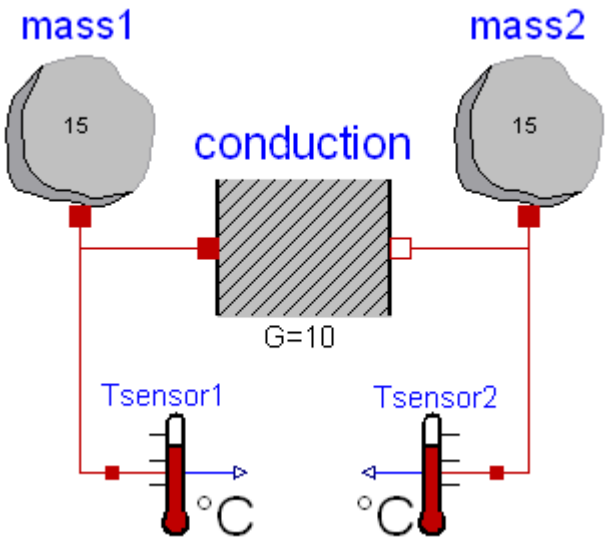
Name	Description
TwoMasses	Simple conduction demo

ControlledTemperature	Control temperature of a resistor
Motor	Second order thermal model of a motor

Modelica.Thermal.HeatTransfer.Examples.TwoMasses



Simple conduction demo



Information

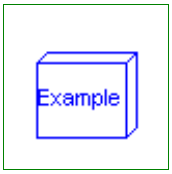
This example demonstrates the thermal response of two masses connected by a conducting element. The two masses have the same heat capacity but different initial temperatures (T1=100 [degC], T2= 0 [degC]). The mass with the higher temperature will cool off while the mass with the lower temperature heats up. They will each asymptotically approach the calculated temperature **T_final_K** (**T_final_degC**) that results from dividing the total initial energy in the system by the sum of the heat capacities of each element.

Simulate for 5 s and plot the variables
mass1.T, mass2.T, T_final_K or
Tsensor1.T, Tsensor2.T, T_final_degC

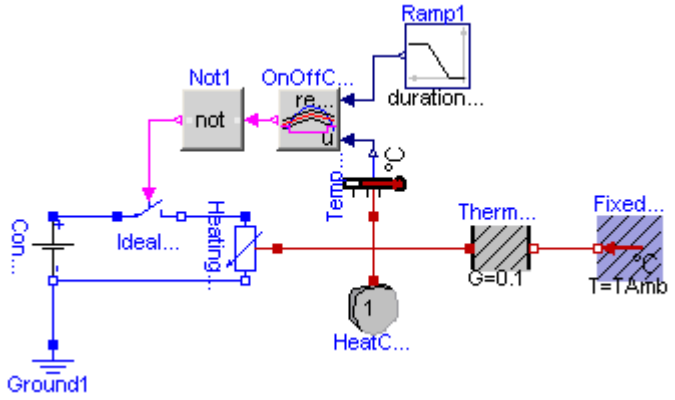
Parameters

Type	Name	Default	Description
Temperature	T_final_K		Projected final temperature [K]
Temperature_degC	T_final_degC		Projected final temperature [degC]

Modelica.Thermal.HeatTransfer.Examples.ControlledTemperature



Control temperature of a resistor



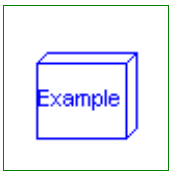
Information

A constant voltage of 10 V is applied to a temperature dependent resistor of 10*(1+(T-20C)/(235+20C)) Ohms, whose losses v**2/r are dissipated via a thermal conductance of 0.1 W/K to ambient temperature 20 degree C. The resistor is assumed to have a thermal capacity of 1 J/K, having ambient temparature at the beginning of the experiment. The temperature of this heating resistor is held by an OnOff-controller at reference temperature within a given bandwith +/- 1 K by switching on and off the voltage source. The reference temperature starts at 25 degree C and rises between t = 2 and 8 seconds linear to 50 degree C. An appropppriate simulating time would be 10 seconds.

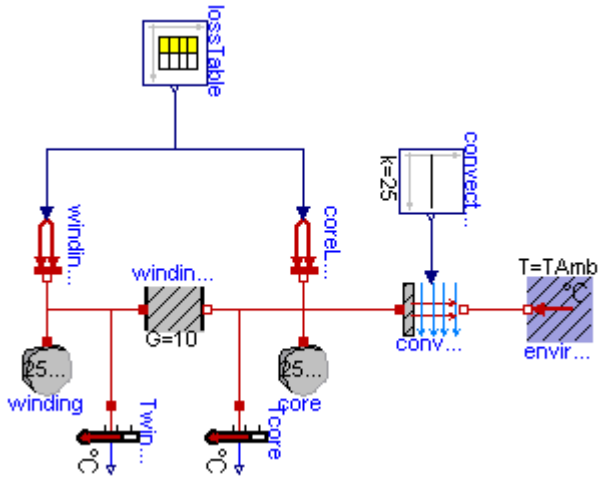
Parameters

Type	Name	Default	Description
Temperature_degC	TAmb	20	Ambient Temperature [degC]
Temperature_degC	TDif	2	Error in Temperature [degC]

Modelica.Thermal.HeatTransfer.Examples.Motor



Second order thermal model of a motor



Information

This example contains a simple second order thermal model of a motor. The periodic power losses are described by table "lossTable":

time	winding losses	core losses
0	100	500
360	100	500
360	1000	500
600	1000	500

Since constant speed is assumed, the core losses keep constant whereas the winding losses are low for 6 minutes (no-load) and high for 4 minutes (over load).
The winding losses are corrected by (1 + alpha*(T - T_ref)) because the winding's resistance is temperature dependent whereas the core losses are kept constant (alpha = 0).

The power dissipation to the environment is approximated by heat flow through a thermal conductance between winding and core, partially storage of the heat in the winding's heat capacity as well as the core's heat capacity and finally by forced convection to the environment.
Since constant speed is assumed, the cinvective conductance keeps constant.
Using Modelica.Thermal.FluidHeatFlow it would be possible to model the coolant air flow, too (instead of simple dissipation to a constant ambient's temperature).

Simulate for 7200 s; plot Twinding.T and Tcore.T.

Parameters

Type	Name	Default	Description
Temperature_degC	TAmb	20	Ambient temperature [degC]

Modelica.Thermal.HeatTransfer.Interfaces

Connectors and partial models

Information

Package Content

Name	Description
HeatPort	Thermal port for 1-dim. heat transfer
HeatPort_a	Thermal port for 1-dim. heat transfer (filled rectangular icon)
HeatPort_b	Thermal port for 1-dim. heat transfer (unfilled rectangular icon)
Element1D	Partial heat transfer element with two HeatPort connectors that does not store energy

---

Modelica.Thermal.HeatTransfer.Interfaces.HeatPort

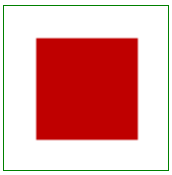
Thermal port for 1-dim. heat transfer

Information

Contents

Type	Name	Description
Temperature	T	Port temperature [K]
flow HeatFlowRate	Q_flow	Heat flow rate (positive if flowing from outside into the component) [W]

Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a



Thermal port for 1-dim. heat transfer (filled rectangular icon)

Information

This connector is used for 1-dimensional heat flow between components. The variables in the connector are:

```
T       Temperature in [Kelvin].
Q_flow  Heat flow rate in [Watt].
```

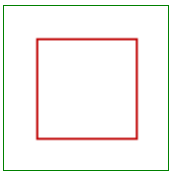
According to the Modelica sign convention, a **positive** heat flow rate **Q_flow** is considered to flow **into** a component. This convention has to be used whenever this connector is used in a model class.

Note, that the two connector classes **HeatPort_a** and **HeatPort_b** are identical with the only exception of the different **icon layout**.

Contents

Type	Name	Description
Temperature	T	Port temperature [K]
flow HeatFlowRate	Q_flow	Heat flow rate (positive if flowing from outside into the component) [W]

Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_b



Thermal port for 1-dim. heat transfer (unfilled rectangular icon)

Information

This connector is used for 1-dimensional heat flow between components. The variables in the connector are:

```
T       Temperature in [Kelvin].
Q_flow  Heat flow rate in [Watt].
```

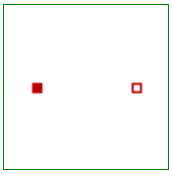
According to the Modelica sign convention, a **positive** heat flow rate **Q_flow** is considered to flow **into** a component. This convention has to be used whenever this connector is used in a model class.

Note, that the two connector classes **HeatPort_a** and **HeatPort_b** are identical with the only exception of the different **icon layout**.

Contents

Type	Name	Description
Temperature	T	Port temperature [K]
flow HeatFlowRate	Q_flow	Heat flow rate (positive if flowing from outside into the component) [W]

Modelica.Thermal.HeatTransfer.Interfaces.Element1D



Partial heat transfer element with two HeatPort connectors that does not store energy

Information

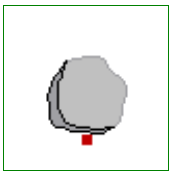
This partial model contains the basic connectors and variables to allow heat transfer models to be created that **do not store energy**, This model defines and includes equations for the temperature drop across the element, **dT**, and the heat flow rate through the element from port_a to port_b, **Q_flow**.

By extending this model, it is possible to write simple constitutive equations for many types of heat transfer components.

Connectors

Type	Name	Description
HeatPort_a	port_a	
HeatPort_b	port_b	

Modelica.Thermal.HeatTransfer.HeatCapacitor



Lumped thermal element storing heat

Information

This is a generic model for the heat capacity of a material. No specific geometry is assumed beyond a total volume with uniform temperature for the entire volume. Furthermore, it is assumed that the heat capacity is constant (indepedent of temperature).

The temperature T [Kelvin] of this component is a **state**. A default of T = 25 degree Celsius (= SIunits.Conversions.from_degC(25)) is used as start value for initialization. This usually means that at start of integration the temperature of this component is 25 degrees Celsius. You may, of course, define a different temperature as start value for initialization. Alternatively, it is possible to set parameter **steadyStateStart** to **true**. In this case the additional equation '**der**(T) = 0' is used during initialization, i.e., the temperature T is computed in such a way that the component starts in **steady state**. This is useful in cases, where one would like to start simulation in a suitable operating point without being forced to integrate for a long time to arrive at this point.

Note, that parameter **steadyStateStart** is not available in the parameter menue of the simulation window, because its value is utilized during translation to generate quite different equations depending on its setting. Therefore, the value of this parameter can only be changed before translating the model.

This component may be used for complicated geometries where the heat capacity C is determined my measurements. If the component consists mainly of one type of material, the **mass m** of the component may be measured or calculated and multiplied with the **specific heat capacity cp** of the component material to compute C:

```
C = cp*m.
Typical values for cp at 20 degC in J/(kg.K):
   aluminium   896
   concrete    840
   copper      383
   iron        452
   silver      235
   steel       420 ... 500 (V2A)
   wood       2500
```

Parameters

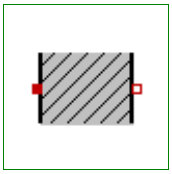
Type	Name	Default	Description
HeatCapacity	C		Heat capacity of part (= cp*m) [J/K]

Boolean	steadyStateStart	false	true, if component shall start in steady state

Connectors

Type	Name	Description
HeatPort_a	port	

Modelica.Thermal.HeatTransfer.ThermalConductor



Lumped thermal element transporting heat without storing it

Information

This is a model for transport of heat without storing it. It may be used for complicated geometries where the thermal conductance G (= inverse of thermal resistance) is determined by measurements and is assumed to be constant over the range of operations. If the component consists mainly of one type of material and a regular geometry, it may be calculated, e.g., with one of the following equations:

- Conductance for a **box** geometry under the assumption that heat flows along the box length:

```
G = k*A/L
k: Thermal conductivity (material constant)
A: Area of box
L: Length of box
```

- Conductance for a **cylindrical** geometry under the assumption that heat flows from the inside to the outside radius of the cylinder:

```
G = 2*pi*k*L/log(r_out/r_in)
pi   : Modelica.Constants.pi
k    : Thermal conductivity (material constant)
L    : Length of cylinder
log  : Modelica.Math.log;
r_out: Outer radius of cylinder
r_in : Inner radius of cylinder
```

```
Typical values for k at 20 degC in W/(m.K):
  aluminium   220
  concrete      1
  copper      384
  iron         74
  silver      407
  steel         45 .. 15 (V2A)
  wood          0.1 ... 0.2
```

Parameters

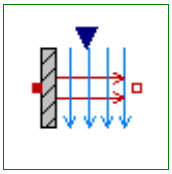
Type	Name	Default	Description
ThermalConductance	G		Constant thermal conductance of material [W/K]

Connectors

Type	Name	Description
HeatPort_a	port_a	

HeatPort_b	port_b	

Modelica.Thermal.HeatTransfer.Convection



Lumped thermal element for heat convection

Information

This is a model of linear heat convection, e.g., the heat transfer between a plate and the surrounding air. It may be used for complicated solid geometries and fluid flow over the solid by determining the convective thermal conductance Gc by measurements. The basic constitutive equation for convection is

```
Q_flow = Gc*(solid.T - fluid.T);
Q_flow: Heat flow rate from connector 'solid' (e.g. a plate)
   to connector 'fluid' (e.g. the surrounding air)
```

Gc = G.signal[1] is an input signal to the component, since Gc is nearly never constant in practice. For example, Gc may be a function of the speed of a cooling fan. For simple situations, Gc may be *calculated* according to

```
Gc = A*h
A: Convection area (e.g. perimeter*length of a box)
h: Heat transfer coefficient
```

where the heat transfer coefficient h is calculated from properties of the fluid flowing over the solid. Examples:

Machines cooled by air (empirical, very rough approximation according to R. Fischer: Elektrische Maschinen, 10th edition, Hanser-Verlag 1999, p. 378):

```
 h = 7.8*v^0.78 [W/(m2.K)] (forced convection)
   = 12         [W/(m2.K)] (free convection)
 where
   v: Air velocity in [m/s]
```

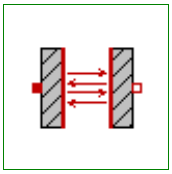
Laminar flow with constant velocity of a fluid along a **flat plate** where the heat flow rate from the plate to the fluid (= solid.Q_flow) is kept constant (according to J.P.Holman: Heat Transfer, 8th edition, McGraw-Hill, 1997, p.270):

```
h  = Nu*k/x;
Nu = 0.453*Re^(1/2)*Pr^(1/3);
where
   h  : Heat transfer coefficient
   Nu : = h*x/k       (Nusselt number)
   Re : = v*x*rho/mue (Reynolds number)
   Pr : = cp*mue/k    (Prandtl number)
   v  : Absolute velocity of fluid
   x  : distance from leading edge of flat plate
   rho: density of fluid (material constant
   mue: dynamic viscosity of fluid (material constant)
   cp : specific heat capacity of fluid (material constant)
   k  : thermal conductivity of fluid (material constant)
and the equation for h holds, provided
   Re < 5e5 and 0.6 < Pr < 50
```

Connectors

Type	Name	Description
input RealInput	Gc	Signal representing the convective thermal conductance in [W/K]
HeatPort_a	solid	
HeatPort_b	fluid	

Modelica.Thermal.HeatTransfer.BodyRadiation



Lumped thermal element for radiation heat transfer

Information

This is a model describing the thermal radiation, i.e., electromagnetic radiation emitted between two bodies as a result of their temperatures. The following constitutive equation is used:

```
Q_flow = Gr*sigma*(port_a.T^4 - port_b.T^4);
```

where Gr is the radiation conductance and sigma is the Stefan-Boltzmann constant (= Modelica.Constants.sigma). Gr may be determined by measurements and is assumed to be constant over the range of operations.

For simple cases, Gr may be analytically computed. The analytical equations use epsilon, the emission value of a body which is in the range 0..1. Epsilon=1, if the body absorbs all radiation (= black body). Epsilon=0, if the body reflects all radiation and does not absorb any.

```
Typical values for epsilon:
aluminium, polished     0.04
copper, polished        0.04
gold, polished          0.02
paper                   0.09
rubber                  0.95
silver, polished        0.02
wood                    0.85..0.9
```

Analytical Equations for Gr

Small convex object in large enclosure (e.g., a hot machine in a room):

```
Gr = e*A
where
   e: Emission value of object (0..1)
   A: Surface area of object where radiation
      heat transfer takes place
```

Two parallel plates:

```
Gr = A/(1/e1 + 1/e2 - 1)
where
   e1: Emission value of plate1 (0..1)
   e2: Emission value of plate2 (0..1)
   A : Area of plate1 (= area of plate2)
```

Two long cylinders in each other, where radiation takes place from the inner to the outer cylinder):

```
Gr = 2*pi*r1*L/(1/e1 + (1/e2 - 1)*(r1/r2))
where
   pi: = Modelica.Constants.pi
   r1: Radius of inner cylinder
```

```
r2: Radius of outer cylinder
L : Length of the two cylinders
e1: Emission value of inner cylinder (0..1)
e2: Emission value of outer cylinder (0..1)
```

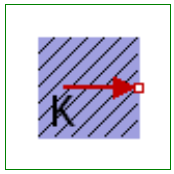
Parameters

Type	Name	Default	Description
Real	Gr		Net radiation conductance between two surfaces (see docu) [m2]

Connectors

Type	Name	Description
HeatPort_a	port_a	
HeatPort_b	port_b	

Modelica.Thermal.HeatTransfer.FixedTemperature



Fixed temperature boundary condition in Kelvin

Information

This model defines a fixed temperature T at its port in Kelvin, i.e., it defines a fixed temperature as a boundary condition.

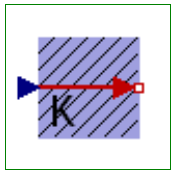
Parameters

Type	Name	Default	Description
Temperature	T		Fixed temperature at port [K]

Connectors

Type	Name	Description
HeatPort_b	port	

Modelica.Thermal.HeatTransfer.PrescribedTemperature



Variable temperature boundary condition in Kelvin

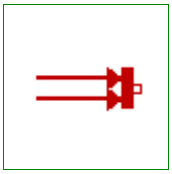
Information

This model represents a variable temperature boundary condition. The temperature in [K] is given as input signal **T** to the model. The effect is that an instance of this model acts as an infinite reservoir able to absorb or generate as much energy as required to keep the temperature at the specified value.

Connectors

Type	Name	Description
HeatPort_b	port	
input RealInput	T	

Modelica.Thermal.HeatTransfer.FixedHeatFlow



Fixed heat flow boundary condition

Information

This model allows a specified amount of heat flow rate to be "injected" into a thermal system at a given port. The constant amount of heat flow rate Q_flow is given as a parameter. The heat flows into the component to which the component FixedHeatFlow is connected, if parameter Q_flow is positive.

If parameter alpha is > 0, the heat flow is mulitplied by (1 + alpha*(port.T - T_ref)) in order to simulate temperature dependent losses (which are given an reference temperature T_ref).

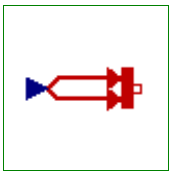
Parameters

Type	Name	Default	Description
HeatFlowRate	Q_flow		Fixed heat flow rate at port [W]
Temperature	T_ref	from_degC(20)	Reference temperature [K]
Real	alpha	0	Temperature coefficient of heat flow rate [1/K]

Connectors

Type	Name	Description
HeatPort_b	port	

Modelica.Thermal.HeatTransfer.PrescribedHeatFlow



Prescribed heat flow boundary condition

Information

This model allows a specified amount of heat flow rate to be "injected" into a thermal system at a given port. The amount of heat is given by the input signal Q_flow into the model. The heat flows into the component to which the component PrescribedHeatFlow is connected, if the input signal is positive.

If parameter alpha is > 0, the heat flow is mulitplied by (1 + alpha*(port.T - T_ref)) in order to simulate temperature dependent losses (which are given an reference temperature T_ref).

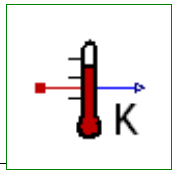
Parameters

Type	Name	Default	Description
Temperature	T_ref	from_degC(20)	Reference temperature [K]
Real	alpha	0	Temperature coefficient of heat flow rate [1/K]

Connectors

Type	Name	Description
input RealInput	Q_flow	
HeatPort_b	port	

Modelica.Thermal.HeatTransfer.TemperatureSensor



Absolute temperature sensor in Kelvin

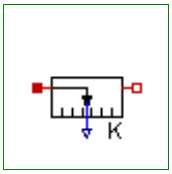
Information

This is an ideal absolute temperature sensor which returns the temperature of the connected port in Kelvin as an output signal. The sensor itself has no thermal interaction with whatever it is connected to. Furthermore, no thermocouple-like lags are associated with this sensor model.

Connectors

Type	Name	Description
output RealOutput	T	
HeatPort_a	port	

Modelica.Thermal.HeatTransfer.RelTemperatureSensor



Relative Temperature sensor

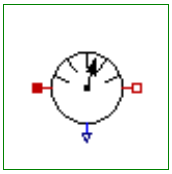
Information

The relative temperature "port_a.T - port_b.T" is determined between the two ports of this component and is provided as output signal in Kelvin.

Connectors

Type	Name	Description
HeatPort_a	port_a	
HeatPort_b	port_b	
output RealOutput	T_rel	

Modelica.Thermal.HeatTransfer.HeatFlowSensor



Heat flow rate sensor

Information

This model is capable of monitoring the heat flow rate flowing through this component. The sensed value of heat flow rate is the amount that passes through this sensor while keeping the temperature drop across the sensor zero. This is an ideal model so it does not absorb any energy and it has no direct effect on the thermal response of a system it is included in. The output signal is positive, if the heat flows from port_a to port_b.

Connectors

Type	Name	Description
output RealOutput	Q_flow	Heat flow from port_a to port_b
HeatPort_a	port_a	
HeatPort_b	port_b	

Modelica.Thermal.HeatTransfer.Celsius

Components with Celsius input and/or output

Information

The components of this package are provided for the convenience of people working mostly with Celsius units, since all models in package HeatTransfer are based on Kelvin units.

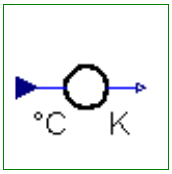
Note, that in package SIunits.Conversions, functions are provided to convert between the units Kelvin, degree Celsius, degree Fahrenheit, and degree Rankine. These functions allow, e.g., a direct conversion of units at all places where Kelvin is required as parameter. Example:

```
import SIunits.Conversions.*;
Modelica.Thermal.HeatTransfer.HeatCapacitor C(T0 = from_degC(20));
```

Package Content

Name	Description
ToKelvin	Conversion block from °Celsius to Kelvin
FromKelvin	Conversion from Kelvin to °Celsius
FixedTemperature	Fixed temperature boundary condition in degree Celsius
PrescribedTemperature	Variable temperature boundary condition in °Celsius
TemperatureSensor	Absolute temperature sensor in °Celsius

Modelica.Thermal.HeatTransfer.Celsius.ToKelvin



Conversion block from °Celsius to Kelvin

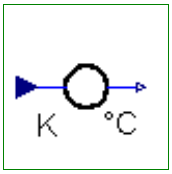
Information

This component converts an input signal from Celsius to Kelvin and provide is as output signal.

Connectors

Type	Name	Description
input RealInput	Celsius	
output RealOutput	Kelvin	

Modelica.Thermal.HeatTransfer.Celsius.FromKelvin



Conversion from Kelvin to °Celsius

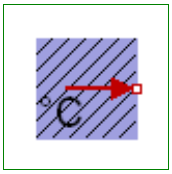
Information

This component converts an input signal from Kelvin to Celsius and provides is as output signal.

Connectors

Type	Name	Description
input RealInput	Kelvin	
output RealOutput	Celsius	

Modelica.Thermal.HeatTransfer.Celsius.FixedTemperature



Fixed temperature boundary condition in degree Celsius

Information

This model defines a fixed temperature T at its port in [degC], i.e., it defines a fixed temperature as a boundary condition.

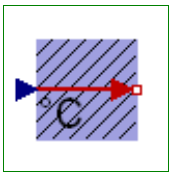
Parameters

Type	Name	Default	Description
Temperature_degC	T		Fixed Temperature at the port [degC]

Connectors

Type	Name	Description
HeatPort_b	port	

Modelica.Thermal.HeatTransfer.Celsius.PrescribedTemperature



Variable temperature boundary condition in °Celsius

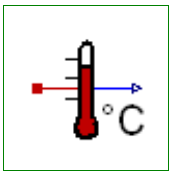
Information

This model represents a variable temperature boundary condition The temperature value in [degC] is given by the input signal to the model. The effect is that an instance of this model acts as an infinite reservoir able to absorb or generate as much energy as required to keep the temperature at the specified value.

Connectors

Type	Name	Description
HeatPort_b	port	
input RealInput	T	

Modelica.Thermal.HeatTransfer.Celsius.TemperatureSensor



Absolute temperature sensor in °Celsius

Information

This is an ideal absolute temperature sensor which returns the temperature of the connected port in Celsius as an output signal. The sensor itself has no thermal interaction with whatever it is connected to. Furthermore, no thermocouple-like lags are associated with this sensor model.

Connectors

Type	Name	Description
output RealOutput	T	
HeatPort_a	port	

Modelica.Thermal.HeatTransfer.Fahrenheit

Components with Fahrenheit input and/or output

Information

The components of this package are provided for the convenience of people working mostly with Fahrenheit units, since all models in package HeatTransfer are based on Kelvin units.

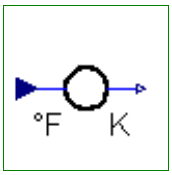
Note, that in package SIunits.Conversions, functions are provided to convert between the units Kelvin, degree Celsius, degree Fahrenheit and degree Rankine. These functions allow, e.g., a direct conversion of units at all places where Kelvin is required as parameter. Example:

```
import SIunits.Conversions.*;
Modelica.Thermal.HeatTransfer.HeatCapacitor C(T0 = from_degF(70));
```

Package Content

Name	Description
ToKelvin	Conversion block from °Fahrenheit to Kelvin
FromKelvin	Conversion from Kelvin to °Fahrenheit
FixedTemperature	Fixed temperature boundary condition in °Fahrenheit
PrescribedTemperature	Variable temperature boundary condition in °Fahrenheit
TemperatureSensor	Absolute temperature sensor in °Fahrenheit

Modelica.Thermal.HeatTransfer.Fahrenheit.ToKelvin



Conversion block from °Fahrenheit to Kelvin

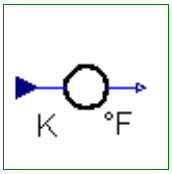
Information

This component converts a input signal from degree Fahrenheit to Kelvin and provides is as output signal.

Connectors

Type	Name	Description
input RealInput	Fahrenheit	
output RealOutput	Kelvin	

Modelica.Thermal.HeatTransfer.Fahrenheit.FromKelvin



Conversion from Kelvin to °Fahrenheit

Information

This component converts all input signals from Kelvin to Fahrenheit and provides them as output signals.

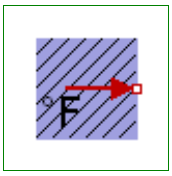
Parameters

Type	Name	Default	Description
Integer	n	1	Number of inputs (= number of outputs)

Connectors

Type	Name	Description
input RealInput	Kelvin	
output RealOutput	Fahrenheit	

Modelica.Thermal.HeatTransfer.Fahrenheit.FixedTemperature



Fixed temperature boundary condition in °Fahrenheit

Information

This model defines a fixed temperature T at its port in [degF], i.e., it defines a fixed temperature as a boundary condition.

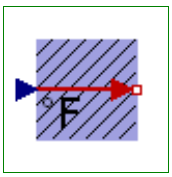
Parameters

Type	Name	Default	Description
Temperature_degF	T		Fixed Temperature at the port [degF]

Connectors

Type	Name	Description
HeatPort_b	port	

Modelica.Thermal.HeatTransfer.Fahrenheit.PrescribedTemperature



Variable temperature boundary condition in °Fahrenheit

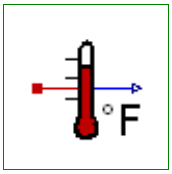
Information

This model represents a variable temperature boundary condition The temperature value in [degF] is given by the input signal to the model. The effect is that an instance of this model acts as an infinite reservoir able to absorb or generate as much energy as required to keep the temperature at the specified value.

Connectors

Type	Name	Description
HeatPort_b	port	
input RealInput	T	

Modelica.Thermal.HeatTransfer.Fahrenheit.TemperatureSensor



Absolute temperature sensor in °Fahrenheit

Information

This is an ideal absolute temperature sensor which returns the temperature of the connected port in Fahrenheit as an output signal. The sensor itself has no thermal interaction with whatever it is connected to. Furthermore, no thermocouple-like lags are associated with this sensor model.

Connectors

Type	Name	Description
output RealOutput	T	
HeatPort_a	port	

Modelica.Thermal.HeatTransfer.Rankine

Components with Rankine input and/or output

Information

The components of this package are provided for the convenience of people working mostly with Rankine units, since all models in package HeatTransfer are based on Kelvin units.

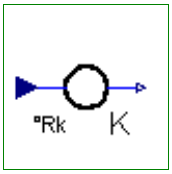
Note, that in package SIunits.Conversions, functions are provided to convert between the units Kelvin, degree Celsius, degree Fahrenheit and degree Rankine. These functions allow, e.g., a direct conversion of units at all places where Kelvin is required as parameter. Example:

```
import SIunits.Conversions.*;
Modelica.Thermal.HeatTransfer.HeatCapacitor C(T0 = from_degRk(500));
```

Package Content

Name	Description
ToKelvin	Conversion block from °Rankine to Kelvin
FromKelvin	Conversion from Kelvin to °Rankine
FixedTemperature	Fixed temperature boundary condition in °Rankine
PrescribedTemperature	Variable temperature boundary condition in °Rankine
TemperatureSensor	Absolute temperature sensor in °Rankine

Modelica.Thermal.HeatTransfer.Rankine.ToKelvin



Conversion block from °Rankine to Kelvin

Information

This component converts all input signals from degree Rankine to Kelvin and provides them as output signals.

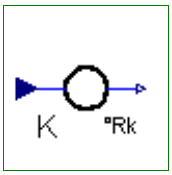
Parameters

Type	Name	Default	Description
Integer	n	1	Number of inputs (= number of outputs)

Connectors

Type	Name	Description
input RealInput	Rankine	
output RealOutput	Kelvin	

Modelica.Thermal.HeatTransfer.Rankine.FromKelvin



Conversion from Kelvin to °Rankine

Information

This component converts all input signals from Kelvin to Rankine and provides them as output signals.

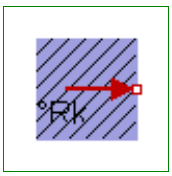
Parameters

Type	Name	Default	Description
Integer	n	1	Number of inputs (= number of outputs)

Connectors

Type	Name	Description
input RealInput	Kelvin	
output RealOutput	Rankine	

Modelica.Thermal.HeatTransfer.Rankine.FixedTemperature



Fixed temperature boundary condition in °Rankine

Information

This model defines a fixed temperature T at its port in degree Rankine, [degRk], i.e., it defines a fixed temperature as a boundary condition.

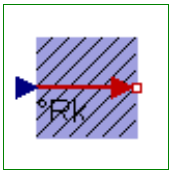
Parameters

Type	Name	Default	Description
Temperature_degRk	T		Fixed Temperature at the port [degRk]

Connectors

Type	Name	Description
HeatPort_b	port	

Modelica.Thermal.HeatTransfer.Rankine.PrescribedTemperature



Variable temperature boundary condition in °Rankine

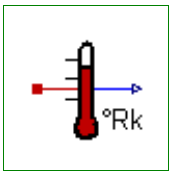
Information

This model represents a variable temperature boundary condition The temperature value in degree Rankine, [degRk] is given by the input signal to the model. The effect is that an instance of this model acts as an infinite reservoir able to absorb or generate as much energy as required to keep the temperature at the specified value.

Connectors

Type	Name	Description
HeatPort_b	port	
input RealInput	T	

Modelica.Thermal.HeatTransfer.Rankine.TemperatureSensor



Absolute temperature sensor in °Rankine

Information

This is an ideal absolute temperature sensor which returns the temperature of the connected port in Rankine as an output signal. The sensor itself has no thermal interaction with whatever it is connected to. Furthermore, no thermocouple-like lags are associated with this sensor model.

Connectors

Type	Name	Description
output RealOutput	T	
HeatPort_a	port	

Modelica.Utilities

Library of utility functions dedicated to scripting (operating on files, streams, strings, system)

Information

This package contains Modelica **functions** that are especially suited for **scripting**. The functions might be used to work with strings, read data from file, write data to file or copy, move and remove files.

For an introduction, have especially a look at:

- Modelica.Utilities.User's Guide discusses the most important aspects of this library.
- Modelica.Utilities.Examples contains examples that demonstrate the usage of this library.

The following main sublibraries are available:

- Files provides functions to operate on files and directories, e.g., to copy, move, remove files.
- Streams provides functions to read from files and write to files.
- Strings provides functions to operate on strings. E.g. substring, find, replace, sort, scanToken.
- System provides functions to interact with the environment. E.g., get or set the working directory or environment variables and to send a command to the default shell.

Copyright © 1998-2007, Modelica Association, DLR and Dynasim.

*This Modelica package is **free** software; it can be redistributed and/or modified under the terms of the **Modelica license**, see the license conditions and the accompanying **disclaimer** here.*

Package Content

Name	Description
UsersGuide	User's Guide of Utilities Library
Examples	Examples to demonstrate the usage of package Modelica.Utilities

Files	Functions to work with files and directories
Streams	Read from files and write to files
Strings	Operations on strings
System	Interaction with environment
Types	Type definitions used in package Modelica.Utilities

Modelica.Utilities.UsersGuide

User's Guide of Utilities Library

Library **Modelica.Utilities** contains Modelica **functions** that are especially suited for **scripting**. Currently, only a rudimentary User's Guide is present. This will be improved in the next releases. The User's Guide has currently the following chapters:

1. Release Notes summarizes the differences between different versions of this library.
2. ImplementationNotes describes design decisions for this library especially for Modelica tool vendors.
3. Contact provides information about the authors of the library as well as acknowledgments.

Error handling
In case of error, all functions in this library use a Modelica "assert(..)" to provide an error message and to cancel all actions. This means that functions do not return, if an error is triggered inside the function. In the near future, an exception handling mechanism will be introduced in Modelica that will allow to catch errors at a defined place.

Package Content

Name	Description
ImplementationNotes	Implementation Notes
ReleaseNotes	Release notes
Contact	Contact

Modelica.Utilities.UsersGuide.ImplementationNotes

Implementation Notes

Below the major design decisions of this library are summarized.

- **C-Function Interface**
  This library contains several interfaces to C-functions in order to operate with the environment. As will become clear, it is usally required that a Modelica tool vendor provides an implementation of these C-functions that are suited for his environment. In directory "Modelica.Utilities\C-Sources" a reference implementation is given for Microsoft Windows Systems and for POSIX environments. The files "ModelicaInternal.c" and "ModelicaStrings.c" can be used as a basis for the integration in the vendors environment.

- **Character Encoding**
  The representation of characters is different in operating systems. The more modern ones (e.g. Windows-NT) use an early variant of Unicode (16 bit per character) other (e.g. Windows-ME) use 8-bit encoding. Also 32 bit per character and multi-byte representations are in use. This is important, since e.g., Japanese Modelica users need Unicode representation. The design in this library is done in such a way that a basic set of calls to the operating system hides the actual character representation. This means, that all functions of this package can be used independent from the underlying character representation.

The C-interface of the Modelica language provides only an 8-bit character encoding passing mechanism of strings. As a result, the reference implementation in "Modelica.Utilities\C-Source" needs to be adapted to the character representation supported in the Modelica vendor environment.

- **Internal String Representation**
  The design of this package was made in order that string handling is convenient. This is in contrast to, e.g., the C-language, where string handling is inconvenient, cumbersome and error prone, but on the other hand is in some sense "efficient". The standard reference implementation in "Modelica.Utilities\C-Source" is based on the standard C definition of a string, i.e., a pointer to a sequence of characters, ended with a null terminating character. In order that the string handling in this package is convenient, some assumptions have been made, especially, that the access to a substring is efficient (O(1) access instead of O(n) as in standard C). This allows to hide string pointer arithmetic from the user. In such a case, a similiar efficiency as in C can be expected for most high level operations, such as find, sort, replace. The "efficient character access" can be reached if, e.g., the number of characters are stored in a string, and the length of a character is fixed, say 16 or 32 bit (if all Unicode characters shall be represented). A vendor should adapt the reference implementation in this respect.

- **String copy = pointer copy**
  The Modelica language has no mechanism to change a character of a string. When a string has to be modified, the only way to achieve this is to generate it newly. The advantage is that a Modelica tool can treat a string as a constant entity and can replace (expensive) string copy operations by pointer copy operations. For example, when sorting a set of strings the following type of operations occur:

```
String s[:], s_temp;
 ...
s_temp := s[i];
s[i]   := s[j];
s[j]   := s_temp;
```

  Formally, three strings are copied. Due to the feature sketched above, a Modelica tool can replace this copy operation by pointer assignments, a very "cheap" operation. The Modelica.Utilities functions will perform efficiently, if such types of optimizations are supported by the tool.

Modelica.Utilities.UsersGuide.ReleaseNotes

Release notes

Version 1.0, 2004-09-29

First version implemented.

Modelica.Utilities.UsersGuide.Contact

Contact

Responsible for Library:
Dag Brück, Dynasim AB, Sweden.
email: Dag@Dynasim.se

Acknowledgements:

- This library has been designed by:

Dag Brück, Dynasim AB, Sweden
Hilding Elmqvist, Dynasim AB, Sweden
Hans Olsson, Dynasim AB, Sweden
Martin Otter, DLR Oberpfaffenhofen, Germany.

- The library including the C reference implementation has been implemented by Martin Otter and Dag Brück.
- The Examples.calculator demonstration to implement a calculator with this library is from Hilding Elmqvist.
- Helpful comments from Kaj Nyström, PELAB, Linköping, Sweden, are appreciated, as well as discussions at the 34th, 36th, and 40th Modelica Design Meetings in Vienna, Linköping, and Dresden.

Modelica.Utilities.Examples

Examples to demonstrate the usage of package Modelica.Utilities

Information

This package contains quite involved examples that demonstrate how to use the functions of package Modelica.Utilities. In particular the following examples are present.

- Function calculator is an interpreter to evaluate expressions consisting of +,-,*,/,(),sin(), cos(), tan(), sqrt(), pi. For example: calculator("1.5*sin(pi/6)");
- Function expression is the basic function used in "calculator" to evaluate an expression. It is useful if the expression interpreter is used in a larger scan operation (such as readRealParameter below).
- Function readRealParameter reads the value of a parameter from file, given the file and the parameter name. The value on file is interpreted with the Examples.expression function and can therefore be an expression.
- Model readRealParameterModel is a test model to demonstrate the usage of "readRealParameter". The model contains 3 parameters that are read from file "Modelica.Utilities/data/Examples_readRealParameters.txt".

Package Content

Name	Description
calculator	Interpreter to evaluate simple expressions consisting of +,-,*,/,(),sin(), cos(), tan(), sqrt(), pi
expression	Expression interpreter that returns with the position after the expression (expression may consist of +,-,*,/,(),sin(), cos(), tan(), sqrt(), pi
readRealParameter	Read the value of a Real parameter from file
readRealParameterModel	Demonstrate usage of Examples.readRealParameter/.expression

---

Modelica.Utilities.Examples.calculator

Interpreter to evaluate simple expressions consisting of +,-,*,/,(),sin(), cos(), tan(), sqrt(), pi

Information

Syntax

```
result = calculator(expression);
```

Description

This function demonstrates how a simple expression calculator can be implemented in form of a recursive decent parser using basically the Strings.scanToken(..) and Strings.scanDelimiter(..) function.

The following operations are supported (pi=3.14.. is a predefined constant):

```
+, -
*, /
(expression)
sin(expression)
cos(expression)
tan(expression)
sqrt(expression)
pi
```

Example

```
calculator("2+3*(4-1)");  // returns 11
calculator("sin(pi/6)");  // returns 0.5
```

Inputs

Type	Name	Default	Description
String	string		Expression that is evaluated

Outputs

Type	Name	Description
Real	result	Value of expression

Modelica.Utilities.Examples.expression

Expression interpreter that returns with the position after the expression (expression may consist of +,-,*,/,(),sin(), cos(), tan(), sqrt(), pi

Information

Syntax

```
                result = expression(string);
(result, nextIndex) = expression(string, startIndex=1, message="");
```

Description

This function is nearly the same as Examples.**calculator**. The essential difference is that function "expression" might be used in other parsing operations: After the expression is parsed and evaluated, the

function returns with the value of the expression as well as the position of the character directly after the expression.

This function demonstrates how a simple expression calculator can be implemented in form of a recursive decent parser using basically the Strings.scanToken(..) and scanDelimiters(..) function. There are 2 local functions (term, primary) that implement the corresponding part of the grammar.

The following operations are supported (pi=3.14.. is a predefined constant):

```
+, -
*, /
(expression)
sin(expression)
cos(expression)
tan(expression)
sqrt(expression)
pi
```

The optional argument "startIndex" defines at which position scanning of the expression starts.

In case of error, the optional argument "message" is appended to the error message, in order to give additional information where the error occured.

This function parses the following grammaer

```
expression: [ add_op ] term { add_op term }
add_op    : "+" | "-"
term      : primary { mul_op primary }
mul_op    : "*" | "/"
primary   : UNSIGNED_NUMBER
            | pi
            | ( expression )
            | functionName( expression )
function  :   sin
            | cos
            | tan
            | sqrt
```

Note, in Examples.readRealParameter it is shown, how the expression function can be used as part of another scan operation.

Example

```
expression("2+3*(4-1)");  // returns 11
expression("sin(pi/6)");  // returns 0.5
```

Inputs

Type	Name	Default	Description
String	string		Expression that is evaluated
Integer	startIndex	1	Start scanning of expression at character startIndex
String	message	""	Message used in error message if scan is not successful

Outputs

Type	Name	Description
Real	result	Value of expression
Integer	nextIndex	Index after the scanned expression

Modelica.Utilities.Examples.readRealParameter

Read the value of a Real parameter from file

Information

Syntax

```
result = readRealParameter(fileName, name);
```

Description

This function demonstrates how a function can be implemented that reads the value of a parameter from file. The function performs the following actions:

1. It opens file "fileName" and reads the lines of the file.
2. In every line, Modelica line comments ("// ... end-of-line") are skipped
3. If a line consists of "name = expression" and the "name" in this line is identical to the second argument "name" of the function call, the expression calculator Examples.expression is used to evaluate the expression after the "=" character. The expression can optionally be terminated with a ";".
4. The result of the expression evaluation is returned as the value of the parameter "name".

Example

On file "test.txt" the following lines might be present:

```
// Motor data
J        = 2.3     // inertia
w_rel0   = 1.5*2;  // relative angular velocity
phi_rel0 = pi/3
```

The function returns the value "3.0" when called as:

```
readRealParameter("test.txt", "w_rel0")
```

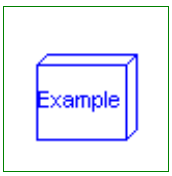
Inputs

Type	Name	Default	Description
String	fileName		Name of file
String	name		Name of parameter

Outputs

Type	Name	Description
Real	result	Actual value of parameter on file

Modelica.Utilities.Examples.readRealParameterModel



Demonstrate usage of Examples.readRealParameter/.expression

Information

Model that shows the usage of Examples.readRealParameter and Examples.expression. The model has 3 parameters and the values of these parameters are read from a file.

Parameters

Type	Name	Default	Description
String	file	classDirectory() + "data/Exa...	File on which data is present
Inertia	J	readRealParameter(file, "J")	[kg.m2]
Angle	phi_rel0	readRealParameter(file, "phi...	[rad]
AngularVelocity	w_rel0	readRealParameter(file, "w_r...	[rad/s]

Modelica.Utilities.Files

Functions to work with files and directories

Information

This package contains functions to work with files and directories. As a general convention of this package, '/' is used as directory separator both for input and output arguments of all functions. For example:

```
exist("Modelica/Mechanics/Rotational.mo");
```

The functions provide the mapping to the directory separator of the underlying operating system. Note, that on Windows system the usage of '\' as directory separator would be inconvenient, because this character is also the escape character in Modelica and C Strings.

In the table below an example call to every function is given:

Function/type	*Description*
list(name)	List content of file or of directory.
copy(oldName, newName) copy(oldName, newName, replace=false)	Generate a copy of a file or of a directory.
move(oldName, newName) move(oldName, newName, replace=false)	Move a file or a directory to another place.
remove(name)	Remove file or directory (ignore call, if it does not exist).
removeFile(name)	Remove file (ignore call, if it does not exist)
createDirectory(name)	Create directory (if directory already exists, ignore call).
result = exist(name)	Inquire whether file or directory exists.
assertNew(name,message)	Trigger an assert, if a file or directory exists.
fullName = fullPathName(name)	Get full path name of file or directory name.
(directory, name, extension) = splitPathName(name)	Split path name in directory, file name kernel, file name extension.
fileName = temporaryFileName()	Return arbitrary name of a file that does not exist and is in a directory where access rights allow to write to this file (useful for temporary output of files).

Package Content

Name	Description
list	List content of file or directory

copy	Generate a copy of a file or of a directory
move	Move a file or a directory to another place
remove	Remove file or directory (ignore call, if it does not exist)
removeFile	Remove file (ignore call, if it does not exist)
createDirectory	Create directory (if directory already exists, ignore call)
exist	Inquire whether file or directory exists
assertNew	Trigger an assert, if a file or directory exists
fullPathName	Get full path name of file or directory name
splitPathName	Split path name in directory, file name kernel, file name extension
temporaryFileName	Return arbitrary name of a file that does not exist and is in a directory where access rights allow to write to this file (useful for temporary output of files)

Modelica.Utilities.Files.list

List content of file or directory

Information

Syntax

```
Files.list(name);
```

Description

If "name" is a regular file, the content of the file is printed.

If "name" is a directory, the directory and file names in the "name" directory are printed in sorted order.

Inputs

Type	Name	Default	Description
String	name		If name is a directory, list directory content. If it is a file, list the file content

Modelica.Utilities.Files.copy

Generate a copy of a file or of a directory

Information

Syntax

```
Files.copy(oldName, newName);
Files.copy(oldName, newName, replace = true);
```

Description

Function **copy**(..) copies a file or a directory to a new location. Via the optional argument **replace** it can be

defined whether an already existing file may be replaced by the required copy.

If oldName/newName are directories, then the newName directory may exist. In such a case the content of oldName is copied into directory newName. If replace = **false** it is required that the existing files in newName are different from the existing files in oldName.

Example

```
copy("C:/test1/directory1", "C:/test2/directory2");
   -> the content of directory1 is copied into directory2
      if "C:/test2/directory2" does not exist, it is newly
      created. If "replace=true", files in directory2
      may be overwritten by their copy
copy("test1.txt", "test2.txt")
   -> make a copy of file "test1.txt" with the name "test2.txt"
      in the current directory
```

Inputs

Type	Name	Default	Description
String	oldName		Name of file or directory to be copied
String	newName		Name of copy of the file or of the directory
Boolean	replace	false	= true, if an existing file may be replaced by the required copy

---

Modelica.Utilities.Files.move

Move a file or a directory to another place

Information

Syntax

```
Files.move(oldName, newName);
Files.move(oldName, newName, replace = true);
```

Description

Function **move**(..) moves a file or a directory to a new location. Via the optional argument **replace** it can be defined whether an already existing file may be replaced.

If oldName/newName are directories, then the newName directory may exist. In such a case the content of oldName is moved into directory newName. If replace = **false** it is required that the existing files in newName are different from the existing files in oldName.

Example

```
move("C:/test1/directory1", "C:/test2/directory2");
   -> the content of directory1 is moved into directory2.
      Afterwards directory1 is deleted.
      if "C:/test2/directory2" does not exist, it is newly
      created. If "replace=true", files in directory2
      may be overwritten
 move("test1.txt", "test2.txt")
   -> rename file "test1.txt" into "test2.txt"
```

```
within the current directory
```

Inputs

Type	Name	Default	Description
String	oldName		Name of file or directory to be moved
String	newName		New name of the moved file or directory
Boolean	replace	false	= true, if an existing file or directory may be replaced

---

Modelica.Utilities.Files.remove

Remove file or directory (ignore call, if it does not exist)

Information

Syntax

```
Files.remove(name);
```

Description

Removes the file or directory "name". If "name" does not exist, the function call is ignored. If "name" is a directory, first the content of the directory is removed and afterwards the directory itself.

This function is silent, i.e., it does not print a message.

Inputs

Type	Name	Default	Description
String	name		Name of file or directory to be removed

Modelica.Utilities.Files.removeFile

Remove file (ignore call, if it does not exist)

Information

Syntax

```
Files.removeFile(fileName);
```

Description

Removes the file "fileName". If "fileName" does not exist, the function call is ignored. If "fileName" exists but is no regular file (e.g., directory, pipe, device, etc.) an error is triggered.

This function is silent, i.e., it does not print a message.

Inputs

Type	Name	Default	Description

String	fileName		Name of file that should be removed

Modelica.Utilities.Files.createDirectory

Create directory (if directory already exists, ignore call)

Information

Syntax

```
Files.createDirectory(directoryName);
```

Description

Creates directory "directorName". If this directory already exists, the function call is ignored. If several directories in "directoryName" do not exist, all of them are created. For example, assume that directory "E:/test1" exists and that directory "E:/test1/test2/test3" shall be created. In this case the directories "test2" in "test1" and "test3" in "test2" are created.

This function is silent, i.e., it does not print a message. In case of error (e.g., "directoryName" is an existing regular file), an assert is triggered.

Inputs

Type	Name	Default	Description
String	directoryName		Name of directory to be created (if present, ignore call)

Modelica.Utilities.Files.exist

Inquire whether file or directory exists

Information

Syntax

```
result = Files.exist(name);
```

Description

Returns true, if "name" is an existing file or directory. If this is not the case, the function returns false.

Inputs

Type	Name	Default	Description
String	name		Name of file or directory

Outputs

Type	Name	Description
Boolean	result	= true, if file or directory exists

Modelica.Utilities.Files.assertNew

Trigger an assert, if a file or directory exists

Information

Syntax

```
Files.assertNew(name);
Files.assertNew(name, message="This is not allowed");
```

Description

Triggers an assert, if "name" is an existing file or directory. The error message has the following structure:

```
File "<name>" already exists.
<message>
```

Inputs

Type	Name	Default	Description
String	name		Name of file or directory
String	message	"This is not allowed."	Message that should be printed after the default message in a new line

Modelica.Utilities.Files.fullPathName

Get full path name of file or directory name

Information

Syntax

```
fullName = Files.fullPathName(name);
```

Description

Returns the full path name of a file or directory "name".

Inputs

Type	Name	Default	Description
String	name		Absolute or relative file or directory name

Outputs

Type	Name	Description
String	fullName	Full path of 'name'

Modelica.Utilities.Files.splitPathName

Split path name in directory, file name kernel, file name extension

Information

Syntax

```
(directory, name, extension) = Files.splitPathName(pathName);
```

Description

Function **splitPathName**(..) splits a path name into its parts.

Example

```
(directory, name, extension) = Files.splitPathName("C:/user/test/input.txt")

-> directory = "C:/user/test/"
   name      = "input"
   extension = ".txt"
```

Inputs

Type	Name	Default	Description
String	pathName		Absolute or relative file or directory name

Outputs

Type	Name	Description
String	directory	Name of the directory including a trailing '/'
String	name	Name of the file without the extension
String	extension	Extension of the file name. Starts with '.'

Modelica.Utilities.Files.temporaryFileName

Return arbitrary name of a file that does not exist and is in a directory where access rights allow to write to this file (useful for temporary output of files)

Information

Syntax

```
fileName = Files.temporaryFileName();
```

Description

Return arbitrary name of a file that does not exist and is in a directory where access rights allow to write to this file (useful for temporary output of files).

Outputs

Type	Name	Description
String	fileName	Full path name of temporary file

Modelica.Utilities.Streams

Read from files and write to files

Information

Library content

Package **Streams** contains functions to input and output strings to a message window or on files. Note that a string is interpreted and displayed as html text (e.g., with print(..) or error(..)) if it is enclosed with the Modelica html quotation, e.g.,

string = "<html> first line <br> second line </html>".

It is a quality of implementation, whether (a) all tags of html are supported or only a subset, (b) how html tags are interpreted if the output device does not allow to display formatted text.

In the table below an example call to every function is given:

Function/type	*Description*
print(string) print(string,fileName)	Print string "string" or vector of strings to message window or on file "fileName".
stringVector = readFile(fileName)	Read complete text file and return it as a vector of strings.
(string, endOfFile) = readLine(fileName, lineNumber)	Returns from the file the content of line lineNumber.
lines = countLines(fileName)	Returns the number of lines in a file.
error(string)	Print error message "string" to message window and cancel all actions
close(fileName)	Close file if it is still open. Ignore call if file is already closed or does not exist.

Use functions **scanXXX** from package Strings to parse a string.

If Real, Integer or Boolean values shall be printed or used in an error message, they have to be first converted to strings with the builtin operator String(...). Example:

```
if x < 0 or x > 1 then
   Streams.error("x (= " + String(x) + ") has to be in the range 0 .. 1");
end if;
```

Package Content

Name	Description
print	Print string to terminal or file
readFile	Read content of a file and return it in a vector of strings
readLine	Reads a line of text from a file and returns it in a string
countLines	Returns the number of lines in a file
error	Print error message and cancel all actions

close	Close file

Modelica.Utilities.Streams.print

Print string to terminal or file

Information

Syntax

```
Streams.print(string);
Streams.print(string,fileName);
```

Description

Function **print**(..) opens automatically the given file, if it is not yet open. If the file does not exist, it is created. If the file does exist, the given string is appended to the file. If this is not desired, call "Files.remove(fileName)" before calling print ("remove(..)" is silent, if the file does not exist). The Modelica environment may close the file whenever appropriate. This can be enforced by calling **Streams.close**(fileName). After every call of "print(..)" a "new line" is printed automatically.

Example

```
Streams.print("x = " + String(x));
Streams.print("y = " + String(y));
Streams.print("x = " + String(y), "mytestfile.txt");
```

See also

Streams, Streams.error, String

Inputs

Type	Name	Default	Description
String	string	""	String to be printed
String	fileName	""	File where to print (empty string is the terminal)

Modelica.Utilities.Streams.readFile

Read content of a file and return it in a vector of strings

Information

Syntax

```
stringVector = Streams.readFile(fileName)
```

Description

Function **readFile**(..) opens the given file, reads the complete content, closes the file and returns the content

as a vector of strings. Lines are separated by LF or CR-LF; the returned strings do not contain the line separators.

Inputs

Type	Name	Default	Description
String	fileName		Name of the file that shall be read

Outputs

Type	Name	Description
String	stringVector[countLines(fileName)]	Content of file

Modelica.Utilities.Streams.readLine

Reads a line of text from a file and returns it in a string

Information

Syntax

```
(string, endOfFile) = Streams.readLine(fileName, lineNumber)
```

Description

Function **readLine**(..) opens the given file, reads enough of the content to get the requested line, and returns the line as a string. Lines are separated by LF or CR-LF; the returned string does not contain the line separator. The file might remain open after the call.

If lineNumber > countLines(fileName), an empty string is returned and endOfFile=true. Otherwise endOfFile=false.

Inputs

Type	Name	Default	Description
String	fileName		Name of the file that shall be read
Integer	lineNumber		Number of line to read

Outputs

Type	Name	Description
String	string	Line of text
Boolean	endOfFile	If true, end-of-file was reached when trying to read line

---

Modelica.Utilities.Streams.countLines

Returns the number of lines in a file

Information

Syntax

```
numberOfLines = Streams.countLines(fileName)
```

Description

Function **countLines**(..) opens the given file, reads the complete content, closes the file and returns the number of lines. Lines are separated by LF or CR-LF.

Inputs

Type	Name	Default	Description
String	fileName		Name of the file that shall be read

Outputs

Type	Name	Description
Integer	numberOfLines	Number of lines in file

Modelica.Utilities.Streams.error

Print error message and cancel all actions

Information

Syntax

```
Streams.error(string);
```

Description

Print the string "string" as error message and cancel all actions. Line breaks are characterized by "\n" in the string.

Example

```
  Streams.error("x (= " + String(x) + ")\nhas to be in the range 0 ..
1");
```

See also

Streams, Streams.print, String

Inputs

Type	Name	Default	Description
String	string		String to be printed to error message window

Modelica.Utilities.Streams.close

Close file

Information

Syntax

```
Streams.close(fileName)
```

Description

Close file if it is open. Ignore call if file is already closed or does not exist.

Inputs

Type	Name	Default	Description
String	fileName		Name of the file that shall be closed

Modelica.Utilities.Strings

Operations on strings

Information

Library content

Package **Strings** contains functions to manipulate strings.

In the table below an example call to every function is given using the **default** options.

Function	*Description*
len = length(string)	Returns length of string
string2 = substring(string1,startIndex,endIndex)	Returns a substring defined by start and end index
result = repeat(n) result = repeat(n,string)	Repeat a blank or a string n times.
result = compare(string1, string2)	Compares two substrings with regards to alphabetical order
identical = isEqual(string1,string2)	Determine whether two strings are identical
result = count(string,searchString)	Count the number of occurrences of a string
index = find(string,searchString)	Find first occurrence of a string in another string
index = findLast(string,searchString)	Find last occurrence of a string in another string
string2 = replace(string,searchString,replaceString)	Replace one or all occurrences of a string
stringVector2 = sort(stringVector1)	Sort vector of strings in alphabetic order
(token, index) = scanToken(string,startIndex)	Scan for a token (Real/Integer/Boolean/String/Identifier/Delimiter/NoToken)
(number, index) = scanReal(string,startIndex)	Scan for a Real constant
(number, index) = scanInteger(string,startIndex)	Scan for an Integer constant

(boolean, index) = scanBoolean(string,startIndex)	Scan for a Boolean constant
(string2, index) = scanString(string,startIndex)	Scan for a String constant
(identifier, index) = scanIdentifier(string,startIndex)	Scan for an identifier
(delimiter, index) = scanDelimiter(string,startIndex)	Scan for delimiters
scanNoToken(string,startIndex)	Check that remaining part of string consists solely of white space or line comments ("// ...\n").
syntaxError(string,index,message)	Print a "syntax error message" as well as a string and the index at which scanning detected an error

The functions "compare", "isEqual", "count", "find", "findLast", "replace", "sort" have the optional input argument **caseSensitive** with default **true**. If **false**, the operation is carried out without taking into account whether a character is upper or lower case.

Package Content

Name	Description
length	Returns length of string
substring	Returns a substring defined by start and end index
repeat	Repeat a string n times
compare	Compare two strings lexicographically
isEqual	Determine whether two strings are identical
count	Count the number of non-overlapping occurrences of a string
find	Find first occurrence of a string within another string
findLast	Find last occurrence of a string within another string
replace	Replace non-overlapping occurrences of a string from left to right
sort	Sort vector of strings in alphabetic order
scanToken	Scan for the next token and return it
scanReal	Scan for the next Real number and trigger an assert if not present
scanInteger	Scan for the next Integer number and trigger an assert if not present
scanBoolean	Scan for the next Boolean number and trigger an assert if not present
scanString	Scan for the next Modelica string and trigger an assert if not present
scanIdentifier	Scan for the next Identifier and trigger an assert if not present
scanDelimiter	Scan for the next delimiter and trigger an assert if not present
scanNoToken	Scan string and check that it contains no more token
syntaxError	Print an error message, a string and the index at which scanning detected an error
Advanced	Advanced scanning functions

Modelica.Utilities.Strings.length

Returns length of string

Information

Syntax

```
Strings.length(string);
```

Description

Returns the number of characters of "string".

Inputs

Type	Name	Default	Description
String	string		

Outputs

Type	Name	Description
Integer	result	Number of characters of string

Modelica.Utilities.Strings.substring

Returns a substring defined by start and end index

Information

Syntax

```
string2 = Strings.substring(string, startIndex, endIndex);
```

Description

This function returns the substring from position startIndex up to and including position endIndex of "string" .

If index, startIndex, or endIndex are not correct, e.g., if endIndex > length(string), an assert is triggered.

Example

```
string1 := "This is line 111";
string2 := Strings.substring(string1,9,12); // string2 = "line"
```

Inputs

Type	Name	Default	Description
String	string		String from which a substring is inquired
Integer	startIndex		Character position of substring begin (index=1 is first character in string)
Integer	endIndex		Character position of substring end

Outputs

Type	Name	Description
String	result	String containing substring string[startIndex:endIndex]

Modelica.Utilities.Strings.repeat

Repeat a string n times

Information

Syntax

```
string2 = Strings.repeat(n);
string2 = Strings.repeat(n, string=" ");
```

Description

The first form returns a string consisting of n blanks.

The second form returns a string consisting of n substrings defined by the optional argument "string".

Inputs

Type	Name	Default	Description
Integer	n	1	Number of occurences
String	string	" "	String that is repeated

Outputs

Type	Name	Description
String	repeatedString	String containing n concatenated strings

Modelica.Utilities.Strings.compare

Compare two strings lexicographically

Information

Syntax

```
result = Strings.compare(string1, string2);
result = Strings.compare(string1, string2, caseSensitive=true);
```

Description

Compares two strings. If the optional argument caseSensitive=false, upper case letters are treated as if they would be lower case letters. The result of the comparison is returned as:

```
result = Modelica.Utilities.Types.Compare.Less     // string1 < string2
       = Modelica.Utilities.Types.Compare.Equal    // string1 = string2
       = Modelica.Utilities.Types.Compare.Greater  // string1 > string2
```

Comparison is with regards to lexicographical order, e.g., "a" < "b";

Inputs

Type	Name	Default	Description
String	string1		
String	string2		
Boolean	caseSensitive	true	= false, if case of letters is ignored

Outputs

Type	Name	Description
Type	result	Result of comparison

Modelica.Utilities.Strings.isEqual

Determine whether two strings are identical

Information

Syntax

```
Strings.isEqual(string1, string2);
Strings.isEqual(string1, string2, caseSensitive=true);
```

Description

Compare whether two strings are identical, optionally ignoring case.

Inputs

Type	Name	Default	Description
String	string1		
String	string2		
Boolean	caseSensitive	true	= false, if lower and upper case are ignored for the comparison

Outputs

Type	Name	Description
Boolean	identical	True, if string1 is identical to string2

Modelica.Utilities.Strings.count

Count the number of non-overlapping occurrences of a string

Information

Syntax

```
Strings.count(string, searchString)
Strings.count(string, searchString, startIndex=1,
                      caseSensitive=true)
```

Description

Returns the number of non-overlapping occurrences of string "searchString" in "string". The search is started at index "startIndex" (default = 1). If the optional argument "caseSensitive" is false, for the counting it does not matter whether a letter is upper or lower case. /p>

Inputs

Type	Name	Default	Description
String	string		String that is analyzed
String	searchString		String that is searched for in string
Integer	startIndex	1	Start search at index startIndex
Boolean	caseSensitive	true	= false, if lower and upper case are ignored for count

Outputs

Type	Name	Description
Integer	result	Number of occurrences of 'searchString' in 'string'

Modelica.Utilities.Strings.find

Find first occurrence of a string within another string

Information

Syntax

```
index = Strings.find(string, searchString);
index = Strings.find(string, searchString, startIndex=1,
                     caseSensitive=true);
```

Description

Finds first occurence of "searchString" within "string" and return the corresponding index. Start search at index "startIndex" (default = 1). If the optional argument "caseSensitive" is false, lower and upper case are ignored for the search. If "searchString" is not found, a value of "0" is returned.

Inputs

Type	Name	Default	Description
String	string		String that is analyzed
String	searchString		String that is searched for in string
Integer	startIndex	1	Start search at index startIndex
Boolean	caseSensitive	true	= false, if lower and upper case are ignored for the search

Outputs

Type	Name	Description
Integer	index	Index of the beginning of the first occurrence of 'searchString' within 'string', or zero if not present

---

Modelica.Utilities.Strings.findLast

Find last occurrence of a string within another string

Information

Syntax

```
index = Strings.findLast(string, searchString);
index = Strings.findLast(string, searchString,
                         startIndex=length(string),
caseSensitive=true,
```

Description

Finds first occurence of "searchString" within "string" when searching from the last character of "string" backwards, and return the corresponding index. Start search at index "startIndex" (default = length(string)). If the optional argument "caseSensitive" is false, lower and upper case are ignored for the search. If "searchString" is not found, a value of "0" is returned.

Inputs

Type	Name	Default	Description
String	string		String that is analyzed
String	searchString		String that is searched for in string
Integer	startIndex	0	Start search at index startIndex. If startIndex = 0, start at length(string)
Boolean	caseSensitive	true	= false, if lower and upper case are ignored for the search

Outputs

Type	Name	Description
Integer	index	Index of the beginning of the last occurrence of 'searchString' within 'string', or zero if not present

---

Modelica.Utilities.Strings.replace

Replace non-overlapping occurrences of a string from left to right

Information

Syntax

```
Strings.replace(string, searchString, replaceString);
Strings.replace(string, searchString, replaceString,
                startIndex=1, replaceAll=true, caseSensitive=true);
```

Description

Search in "string" for "searchString" and replace the found substring by "replaceString".

- The search starts at the first character of "string", or at character position "startIndex", if this optional argument is provided.
- If the optional argument "replaceAll" is **true** (default), all occurrences of "searchString" are replaced.

If the argument is **false**, only the first occurrence is replaced.

- The search for "searchString" distinguishes upper and lower case letters. If the optional argument "caseSensitive" is **false**, the search ignores whether letters are upper or lower case.

The function returns the "string" with the performed replacements.

Inputs

Type	Name	Default	Description
String	string		String to be modified
String	searchString		Replace non-overlapping occurrences of 'searchString' in 'string' with 'replaceString'
String	replaceString		String that replaces 'searchString' in 'string'
Integer	startIndex	1	Start search at index startIndex
Boolean	replaceAll	true	if false, replace only the first occurrence, otherwise all occurrences
Boolean	caseSensitive	true	= false, if lower and upper case are ignored when searching for searchString

Outputs

Type	Name	Description
String	result	Resultant string of replacement operation

Modelica.Utilities.Strings.sort

Sort vector of strings in alphabetic order

Information

Syntax

```
stringVector2 = Streams.sort(stringVector1);
stringVector2 = Streams.sort(stringVector1, caseSensitive=true);
```

Description

Function **sort**(..) sorts a string vector stringVector1 in lexicographical order and returns the result in stringVector2. If the optional argument "caseSensitive" is **false**, lower and upper case letters are not distinguished.

Example

```
s1 = {"force", "angle", "pressure"};
s2 = Strings.sort(s1);
     -> s2 = {"angle", "force", "pressure"};
```

Inputs

Type	Name	Default	Description
String	stringVector1[:]		vector of strings
Boolean	caseSensitive	true	= false, if lower and upper case are ignored when comparing elements of stringVector1

Outputs

Type	Name	Description
String	stringVector2[size(stringVector1, 1)]	string1 sorted in alphabetical order

---

Modelica.Utilities.Strings.scanToken

Scan for the next token and return it

Information

Syntax

```
(token, nextIndex) = Strings.scanToken(string, startIndex,
unsigned=false);
```

Description

Function **scanToken** scans the string starting at index "startIndex" and returns the next token, as well as the index directly after the token. The returned token is a record that holds the type of the token and the value of the token:

token.tokenType	Type of the token, see below
token.real	Real value if tokenType == TokenType.RealToken
token.integer	Integer value if tokenType == TokenType.IntegerToken
token.boolean	Boolean value if tokenType == TokenType.BooleanToken
token.string	String value if tokenType == TokenType.StringToken/IdentifierToken/DelimiterToken

Variable token.tokenType is an enumeration (emulated as a package with constants) that can have the following values:

```
import T = Modelica.Utilities.Types.TokenType;
```

T.RealToken	Modelica Real literal (e.g., 1.23e-4)
T.IntegerToken	Modelica Integer literal (e.g., 123)
T.BooleanToken	Modelica Boolean literal (e.g., false)
T.StringToken	Modelica String literal (e.g., "string 123")
T.IdentifierToken	Modelica identifier (e.g., "force_a")
T.DelimiterToken	any character without white space that does not appear as first character in the tokens above (e.g., "&")
T.NoToken	White space, line comments and no other token until the end of the string

Modelica line comments ("// ... end-of-line/end-of-string") as well as white space is ignored. If "unsigned=true", a Real or Integer literal is not allowed to start with a "+" or "-" sign.

Example

```
import Modelica.Utilities.Strings.*;
import T = Modelica.Utilities.Types.TokenType;
(token, index) := scanToken(string);
if token.tokenType == T.RealToken then
   realValue := token.real;
elseif token.tokenType == T.IntegerToken then
   integerValue := token.integer;
```

```
  elseif token.tokenType == T.BooleanToken then
     booleanValue := token.boolean;
  elseif token.tokenType == T.Identifier then
     name := token.string;
  else
     syntaxError(string,index,"Expected Real, Integer, Boolean or
identifier token");
  end if;
```

Inputs

Type	Name	Default	Description
String	string		String to be scanned
Integer	startIndex	1	Start scanning of string at character startIndex
Boolean	unsigned	false	= true, if Real and Integer tokens shall not start with a sign

Outputs

Type	Name	Description
TokenValue	token	Scanned token
Integer	nextIndex	Index of character after the found token; = 0, if NoToken

---

Modelica.Utilities.Strings.scanReal

Scan for the next Real number and trigger an assert if not present

Information

Syntax

```
                number = Strings.scanReal(string);
(number, nextIndex) = Strings.scanReal(string, startIndex=1,
                                       unsigned=false,
message="");
```

Description

The first form, "scanReal(string)", scans "string" for a Real number with leading white space and returns the value.

The second form, "scanReal(string,startIndex,unsigned)", scans the string starting at index "startIndex", checks whether the next token is a Real literal and returns its value as a Real number, as well as the index directly after the Real number. If the optional argument "unsigned" is **true**, the real number shall not have a leading "+" or "-" sign.

If the required Real number with leading white space is not present in "string", an assert is triggered.

Inputs

Type	Name	Default	Description
String	string		String to be scanned
Integer	startIndex	1	Start scanning of string at character startIndex
Boolean	unsigned	false	= true, if Real token shall not start with a sign

String	message	""	Message used in error message if scan is not successful

Outputs

Type	Name	Description
Real	number	Value of real number
Integer	nextIndex	index of character after the found number

Modelica.Utilities.Strings.scanInteger

Scan for the next Integer number and trigger an assert if not present

Information

Syntax

```
                number = Strings.scanInteger(string);
(number, nextIndex) = Strings.scanInteger(string, startIndex=1,
                                          unsigned=false,
message="");
```

Description

Function **scanInteger** scans the string starting at index "startIndex", checks whether the next token is an Integer literal and returns its value as an Integer number, as well as the index directly after the Integer number. An assert is triggered, if the scanned string does not contain an Integer literal with optional leading white space.

Inputs

Type	Name	Default	Description
String	string		String to be scanned
Integer	startIndex	1	Start scanning of string at character startIndex
Boolean	unsigned	false	= true, if Integer token shall not start with a sign
String	message	""	Message used in error message if scan is not successful

Outputs

Type	Name	Description
Integer	number	Value of Integer number
Integer	nextIndex	Index of character after the found number

Modelica.Utilities.Strings.scanBoolean

Scan for the next Boolean number and trigger an assert if not present

Information

Syntax

```
number = Strings.scanBoolean(string);
```

```
(number, nextIndex) = Strings.scanBoolean(string, startIndex=1,
message="");
```

Description

Function **scanBoolean** scans the string starting at index "startIndex", checks whether the next token is a Boolean literal (i.e., is either the string "false" or "true", if converted to lower case letters) and returns its value as a Boolean number, as well as the index directly after the Boolean number. An assert is triggered, if the scanned string does not contain a Boolean literal with optional leading white space.

Inputs

Type	Name	Default	Description
String	string		String to be scanned
Integer	startIndex	1	Start scanning of string at character startIndex
String	message	""	Message used in error message if scan is not successful

Outputs

Type	Name	Description
Boolean	number	Value of Boolean
Integer	nextIndex	Index of character after the found number

Modelica.Utilities.Strings.scanString

Scan for the next Modelica string and trigger an assert if not present

Information

Syntax

```
                string2 = Strings.scanString(string);
(string2, nextIndex) = Strings.scanString(string, startIndex=1,
message="");
```

Description

Function **scanString** scans the string starting at index "startIndex", checks whether the next token is a String literal and returns its value as a String, as well as the index directly after the String. An assert is triggered, if the scanned string does not contain a String literal with optional leading white space.

Inputs

Type	Name	Default	Description
String	string		String to be scanned
Integer	startIndex	1	Start scanning of string at character startIndex
String	message	""	Message used in error message if scan is not successful

Outputs

Type	Name	Description

String	result	Value of string
Integer	nextIndex	Index of character after the found string

Modelica.Utilities.Strings.scanIdentifier

Scan for the next Identifier and trigger an assert if not present

Information

Syntax

```
                identifier = Strings.scanIdentifier(string);
(identifier, nextIndex) = Strings.scanIdentifier(string, startIndex=1,
message="");
```

Description

Function **scanIdentifier** scans the string starting at index "startIndex", checks whether the next token is an Identifier and returns its value as a string, as well as the index directly after the Identifier. An assert is triggered, if the scanned string does not contain an Identifier with optional leading white space.

Inputs

Type	Name	Default	Description
String	string		String to be scanned
Integer	startIndex	1	Start scanning of identifier at character startIndex
String	message	""	Message used in error message if scan is not successful

Outputs

Type	Name	Description
String	identifier	Value of Identifier
Integer	nextIndex	Index of character after the found identifier

Modelica.Utilities.Strings.scanDelimiter

Scan for the next delimiter and trigger an assert if not present

Information

Syntax

```
                delimiter = Strings.scanDelimiter(string);
(delimiter, nextIndex) = Strings.scanDelimiter(string, startIndex=1,
                                 requiredDelimiters={","},
message="");
```

Description

Function **scanDelimiter** scans the string starting at index "startIndex", checks whether the next token is a delimiter string and returns its value as a string, as well as the index directly after the delimiter. An assert is

triggered, if the scanned string does not contain a delimiter out of the list of requiredDelimiters. Input argument requiredDelimiters is a vector of strings. The elements may have any length, including length 0. If an element of the requiredDelimiters is zero, white space is treated as delimiter. The function returns delimiter="" and nextIndex is the index of the first non white space character.

Inputs

Type	Name	Default	Description
String	string		String to be scanned
Integer	startIndex	1	Start scanning of delimiters at character startIndex
String	requiredDelimiters[:]	{","}	Delimiters that are searched
String	message	""	Message used in error message if scan is not successful

Outputs

Type	Name	Description
String	delimiter	Found delimiter
Integer	nextIndex	Index of character after the found delimiter

---

Modelica.Utilities.Strings.scanNoToken

Scan string and check that it contains no more token

Information

Syntax

```
Strings.scanNoToken(string, startIndex=1, message="");
```

Description

Function **scanNoToken** scans the string starting at index "startIndex" and checks whether there is no more token in the string. An assert is triggered if this is not the case, using the "message" argument as additional explanation in the error text.

Inputs

Type	Name	Default	Description
String	string		String to be scanned
Integer	startIndex	1	Start scanning of string at character startIndex
String	message	""	Message used in error message if scan is not successful

---

Modelica.Utilities.Strings.syntaxError

Print an error message, a string and the index at which scanning detected an error

Information

Syntax

```
Strings.syntaxError(string, index, message);
```

Description

Function **syntaxError** prints an error message in the following form:

```
Syntax error at column <index> of
<string>
    ^       // shows character that is wrong
<message>
```

where the strings withing <..> are the actual values of the input arguments of the function.

If the given string is too long, only a relevant part of the string is printed.

Inputs

Type	Name	Default	Description
String	string		String that has an error at position index
Integer	index		Index of string at which scanning detected an error
String	message	""	String printed at end of error message

Modelica.Utilities.Strings.Advanced

Advanced scanning functions

Information

Library content

Package **Strings.Advanced** contains basic scanning functions. These functions should be **not called** directly, because it is much simpler to utilize the higher level functions "Strings.scanXXX". The functions of the "Strings.Advanced" library provide the basic interface in order to implement the higher level functions in package "Strings".

Library "Advanced" provides the following functions:

```
  (nextIndex, realNumber)     = scanReal         (string, startIndex,
unsigned=false);
  (nextIndex, integerNumber) = scanInteger      (string, startIndex,
unsigned=false);
  (nextIndex, string2)        = scanString       (string, startIndex);
  (nextIndex, identifier)     = scanIdentifier   (string, startIndex);
   nextIndex                  = skipWhiteSpace   (string, startIndex);
   nextIndex                  = skipLineComments (string, startIndex);
```

All functions perform the following actions:

1. Scanning starts at character position "startIndex" of "string" (startIndex has a default of 1).
2. First, white space is skipped, such as blanks (" "), tabs ("\t"), or newline ("\n")
3. Afterwards, the required token is scanned.

4. If successful, on return nextIndex = index of character directly after the found token and the token value is returned as second output argument.
   If not successful, on return nextIndex = startIndex.

The following additional rules apply for the scanning:

- Function scanReal:
  Scans a full number including one optional leading "+" or "-" (if unsigned=false) according to the Modelica grammar. For example, "+1.23e-5", "0.123" are Real numbers, but ".1" is not. Note, an Integer number, such as "123" is also treated as a Real number.

- Function scanInteger:
  Scans an Integer number including one optional leading "+" or "-" (if unsigned=false) according to the Modelica (and C/C++) grammar. For example, "+123", "20" are Integer numbers. Note, a Real number, such as "123.4" is not an Integer and scanInteger returns nextIndex = startIndex.

- Function scanString:
  Scans a String according to the Modelica (and C/C++) grammar, e.g., "This is a "string"" is a valid string token.

- Function scanIdentifier:
  Scans a Modelica identifier, i.e., the identifier starts either with a letter, followed by letters, digits or "_". For example, "w_rel", "T12".

- Function skipLineComments
  Skips white space and Modelica (C/C++) line comments iteratively. A line comment starts with "//" and ends either with an end-of-line ("\n") or the end of the "string".

Package Content

Name	Description
scanReal	Scans a signed real number
scanInteger	Scans signed integer number
scanString	
scanIdentifier	Scans simple identifiers
skipWhiteSpace	Scans white space
skipLineComments	Scans comments and white space

Modelica.Utilities.Strings.Advanced.scanReal

Scans a signed real number

Information

Syntax

```
(nextIndex, realNumber) = scanReal(string, startIndex=1,
unsigned=false);
```

Description

Starts scanning of "string" at position "startIndex". First skips white space and scans afterwards a number of

type Real with an optional sign according to the Modelica grammar:

```
real     ::= [sign] unsigned [fraction] [exponent]
sign     ::= '+' | '-'
unsigned ::= digit [unsigned]
fraction ::= '.' [unsigned]
exponent ::= ('e' | 'E') [sign] unsigned
digit    ::= '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
```

If successful, the function returns nextIndex = index of character directly after the found real number, as well as the value in the second output argument.

If not successful, on return nextIndex = startIndex and the second output argument is zero.

If the optional argument "unsigned" is **true**, the number shall not start with '+' or '-'. The default of "unsigned" is **false**.

See also

Strings.Advanced.

Inputs

Type	Name	Default	Description
String	string		
Integer	startIndex	1	Index where scanning starts
Boolean	unsigned	false	= true, if number shall not start with '+' or '-'

Outputs

Type	Name	Description
Integer	nextIndex	Index after the found token (success=true) or index at which scanning failed (success=false)
Real	number	Value of Real number

Modelica.Utilities.Strings.Advanced.scanInteger

Scans signed integer number

Information

Syntax

```
(nextIndex, integerNumber) = scanInteger(string, startIndex=1,
unsigned=false);
```

Description

Starts scanning of "string" at position "startIndex". First skips white space and scans afterwards a signed number of type Integer. An Integer starts with an optional '+' or '-', immediately followed by a non-empty sequence of digits.

If successful, the function returns nextIndex = index of character directly after the found Integer number, as well as the Integer value in the second output argument.

If not successful, on return nextIndex = startIndex and the second output argument is zero.

Note, a Real number, such as "123.4", is not treated as an Integer number and scanInteger will return nextIndex = startIndex in this case.

If the optional argument "unsigned" is **true**, the number shall not start with '+' or '-'. The default of "unsigned" is **false**.

See also

Strings.Advanced.

Inputs

Type	Name	Default	Description
String	string		
Integer	startIndex	1	
Boolean	unsigned	false	= true, if number shall not start with '+' or '-'

Outputs

Type	Name	Description
Integer	nextIndex	Index after the found token (success=true) or index at which scanning failed (success=false)
Integer	number	Value of Integer number

Modelica.Utilities.Strings.Advanced.scanString

Information

Syntax

```
(nextIndex, string2) = scanString(string, startIndex=1);
```

Description

Starts scanning of "string" at position "startIndex". First skips white space and scans afterwards a string according to the Modelica grammar, i.e., a string enclosed in double quotes.

If successful, the function returns nextIndex = index of character directly after the found string, as well as the string value in the second output argument.

If not successful, on return nextIndex = startIndex and the second output argument is an empty string.

See also

Strings.Advanced.

Inputs

Type	Name	Default	Description
String	string		
Integer	startIndex	1	Index where scanning starts

Outputs

Type	Name	Description
Integer	nextIndex	Index after the found token (success=true) or index at which scanning failed (success=false)
String	string2	Value of String token

Modelica.Utilities.Strings.Advanced.scanIdentifier

Scans simple identifiers

Information

Syntax

```
(nextIndex, identifier) = scanIdentifier(string, startIndex=1);
```

Description

Starts scanning of "string" at position "startIndex". First skips white space and scans afterwards a Modelica identifier, i.e., a sequence of characters starting with a letter ("a".."z" or "A".."Z") followed by letters, digits or underscores ("_").

If successful, the function returns nextIndex = index of character directly after the found identifier, as well as the identifier as string in the second output argument.

If not successful, on return nextIndex = startIndex and the second output argument is an empty string.

See also

Strings.Advanced.

Inputs

Type	Name	Default	Description
String	string		
Integer	startIndex	1	Index where scanning starts

Outputs

Type	Name	Description
Integer	nextIndex	Index after the found token (success=true) or index at which scanning failed (success=false)
String	identifier	Value of identifier token

Modelica.Utilities.Strings.Advanced.skipWhiteSpace

Scans white space

Information

Syntax

```
nextIndex = skipWhiteSpace(string, startIndex);
```

Description

Starts scanning of "string" at position "startIndex" and skips white space. The function returns nextIndex = index of character of the first non white space character.

See also

Strings.Advanced.

Inputs

Type	Name	Default	Description
String	string		
Integer	startIndex	1	

Outputs

Type	Name	Description
Integer	nextIndex	

Modelica.Utilities.Strings.Advanced.skipLineComments

Scans comments and white space

Information

Syntax

```
nextIndex = skipLineComments(string, startIndex);
```

Description

Starts scanning of "string" at position "startIndex". First skips white space and scans afterwards a Modelica (C/C++) line comment, i.e., a sequence of characters that starts with "//" and ends with an end-of-line "\n" or with the end of the string. If end-of-line is reached, the function continues to skip white space and scanning of line comments until end-of-string is reached, or another token is detected.

If successful, the function returns nextIndex = index of character directly after the found line comment.

If not successful, on return nextIndex = startIndex.

See also

Strings.Advanced.

Inputs

Type	Name	Default	Description
String	string		
Integer	startIndex	1	

Outputs

Type	Name	Description
Integer	nextIndex	

Modelica.Utilities.System

Interaction with environment

Information

This package contains functions to interact with the environment.

Package Content

Name	Description
getWorkDirectory	Get full path name of work directory
setWorkDirectory	Set work directory
getEnvironmentVariable	Get content of environment variable
setEnvironmentVariable	Set content of local environment variable
command	Execute command in default shell
exit	Terminate execution of Modelica environment

Modelica.Utilities.System.getWorkDirectory

Get full path name of work directory

Information

Outputs

Type	Name	Description
String	directory	Full path name of work directory

Modelica.Utilities.System.setWorkDirectory

Set work directory

Information

Inputs

Type	Name	Default	Description
String	directory		New work directory

Modelica.Utilities.System.getEnvironmentVariable

Get content of environment variable

Information

Inputs

Type	Name	Default	Description
String	name		Name of environment variable
Boolean	convertToSlash	false	True, if native directory separators in 'result' shall be changed to '/'

Outputs

Type	Name	Description
String	content	Content of environment variable (empty, if not existent)
Boolean	exist	= true, if environment variable exists; = false, if it does not exist

---

Modelica.Utilities.System.setEnvironmentVariable

Set content of local environment variable

Information

Inputs

Type	Name	Default	Description
String	name		Name of environment variable
String	content		Value of the environment variable
Boolean	convertFromSlash	false	True, if '/' in content shall be changed to the native directory separator

---

Modelica.Utilities.System.command

Execute command in default shell

Information

Inputs

Type	Name	Default	Description
String	string		String to be passed to shell

Outputs

Type	Name	Description
Integer	result	Return value from command (depends on environment)

Modelica.Utilities.System.exit

Terminate execution of Modelica environment

Inputs

Type	Name	Default	Description
Integer	status	0	Result to be returned by environment (0 means success)

Modelica.Utilities.Types

Type definitions used in package Modelica.Utilities

Information

This package contains type definitions used in Modelica.Utilities.

Package Content

Name	Description
Compare	Type and constants to compare two strings, as temporary solution until enumerations are available
FileType	Type and constants to describe the type of a file, as temporary solution until enumerations are available
TokenType	Type and constants for token types, as temporary solution until enumerations are available
TokenValue	Value of token

Modelica.Utilities.Types.Compare

Type and constants to compare two strings, as temporary solution until enumerations are available

Information

Syntax

```
TokenType.Temp tokenType = TokenType.RealToken;
```

Description

Package **TokenType** is an emulation of the following enumeration

```
enumeration TokenType = {RealToken, IntegerToken, BooleanToken,
                         StringToken, IdentifierToken, DelimiterToken,
                         NoToken);
```

since enumerations are not yet supported in available Modelica tools.

Package Content

Name	Description
Less=1	
Equal=2	
Greater=3	
Type	

Types and constants

```
constant Integer Less=1;

constant Integer Equal=2;

constant Integer Greater=3;

type Type = Integer;
```

Modelica.Utilities.Types.FileType

Type and constants to describe the type of a file, as temporary solution until enumerations are available

Information

Syntax

```
FileType.Type fileType = FileType.RegularFile;
```

Description

Package **FileType** is an emulation of the following enumeration

```
enumeration FileType = {NoFile, RegularFile, Directory, SpecialFile};
```

since enumerations are not yet supported in available Modelica tools.

Package Content

Name	Description
NoFile=1	no file exists
RegularFile=2	regular file
Directory=3	directory
SpecialFile=4	pipe, FIFO, device, etc.
Type	

Types and constants

```
constant Integer NoFile=1 "no file exists";
```

```
  constant Integer RegularFile=2 "regular file";

  constant Integer Directory=3 "directory";

  constant Integer SpecialFile=4 "pipe, FIFO, device, etc.";

  type Type = Integer;
```

Modelica.Utilities.Types.TokenType

Type and constants for token types, as temporary solution until enumerations are available

Information

Syntax

```
TokenType.Temp tokenType = TokenType.RealToken;
```

Description

Package **TokenType** is an emulation of the following enumeration

```
enumeration TokenType = {RealToken, IntegerToken, BooleanToken,
                         StringToken, IdentifierToken, DelimiterToken,
                         NoToken);
```

since enumerations are not yet supported in available Modelica tools.

Package Content

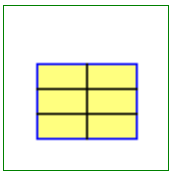
Name	Description
RealToken=1	
IntegerToken=2	
BooleanToken=3	
StringToken=4	
IdentifierToken=5	
DelimiterToken=6	
NoToken=7	
Type	

Types and constants

```
  constant Integer RealToken=1;

  constant Integer IntegerToken=2;

  constant Integer BooleanToken=3;

  constant Integer StringToken=4;

  constant Integer IdentifierToken=5;
```

```
  constant Integer DelimiterToken=6;

  constant Integer NoToken=7;

  type Type = Integer;
```

Modelica.Utilities.Types.TokenValue



Value of token

Information

Modelica definition

```
record TokenValue "Value of token"
   extends Modelica.Icons.Record;
   TokenType.Type tokenType "Type of token";
   Real real "Value if tokenType == TokenType.RealToken";
   Integer integer "Value if tokenType == TokenType.IntegerToken";
   Boolean boolean "Value if tokenType == TokenType.BooleanToken";
   String string
    "Value if tokenType ==
TokenType.StringToken/IdentifierToken/DelimiterToken";
end TokenValue;
```

HTML-documentation generated by Dymola Tue Aug 21 22:32:33 2007.