

Couenne
trunk

Generated by Doxygen 1.8.1.2

Mon Mar 16 2015 21:49:42

Contents

1 Todo List	2
2 Namespace Index	2
2.1 Namespace List	2
3 Class Index	2
3.1 Class Hierarchy	2
4 Class Index	9
4.1 Class List	9
5 File Index	15
5.1 File List	16
6 Namespace Documentation	19
6.1 Bonmin Namespace Reference	19
6.2 Coin Namespace Reference	19
6.3 Couenne Namespace Reference	19
6.3.1 Detailed Description	28
6.3.2 Typedef Documentation	28
6.3.3 Enumeration Type Documentation	28
6.3.4 Function Documentation	33
6.3.5 Variable Documentation	37
6.4 Ipopt Namespace Reference	38
6.5 Osi Namespace Reference	38
7 Class Documentation	38
7.1 Couenne::AuxRelation Class Reference	38
7.1.1 Detailed Description	38
7.1.2 Member Function Documentation	38
7.2 Couenne::BiProdDivRel Class Reference	39
7.2.1 Detailed Description	39
7.2.2 Member Function Documentation	40
7.3 Couenne::CouenneExprMatrix::compare_pair_ind Struct Reference	40
7.3.1 Detailed Description	40
7.3.2 Member Function Documentation	40
7.4 Couenne::CouenneSparseVector::compare_scalars Struct Reference	40
7.4.1 Detailed Description	41

7.4.2	Member Function Documentation	41
7.5	Couenne::compareSol Class Reference	41
7.5.1	Detailed Description	41
7.5.2	Member Function Documentation	41
7.6	Couenne::compExpr Struct Reference	41
7.6.1	Detailed Description	42
7.6.2	Member Function Documentation	42
7.7	Couenne::compNode Struct Reference	42
7.7.1	Detailed Description	42
7.7.2	Member Function Documentation	42
7.8	Couenne::CouenneAggrProbing Class Reference	42
7.8.1	Detailed Description	44
7.8.2	Constructor & Destructor Documentation	44
7.8.3	Member Function Documentation	45
7.8.4	Member Data Documentation	45
7.9	Couenne::CouenneAmplInterface Class Reference	46
7.9.1	Detailed Description	48
7.9.2	Constructor & Destructor Documentation	48
7.9.3	Member Function Documentation	48
7.10	Couenne::CouenneBab Class Reference	48
7.10.1	Detailed Description	49
7.10.2	Constructor & Destructor Documentation	49
7.10.3	Member Function Documentation	50
7.10.4	Member Data Documentation	50
7.11	Couenne::CouenneBranchingObject Class Reference	50
7.11.1	Detailed Description	52
7.11.2	Constructor & Destructor Documentation	52
7.11.3	Member Function Documentation	52
7.11.4	Member Data Documentation	53
7.12	Couenne::CouenneBTPerfIndicator Class Reference	54
7.12.1	Detailed Description	56
7.12.2	Constructor & Destructor Documentation	56
7.12.3	Member Function Documentation	56
7.12.4	Member Data Documentation	56
7.13	Couenne::CouenneChooseStrong Class Reference	58
7.13.1	Detailed Description	59
7.13.2	Constructor & Destructor Documentation	59

7.13.3 Member Function Documentation	59
7.13.4 Member Data Documentation	60
7.14 Couenne::CouenneChooseVariable Class Reference	61
7.14.1 Detailed Description	62
7.14.2 Constructor & Destructor Documentation	62
7.14.3 Member Function Documentation	62
7.14.4 Member Data Documentation	63
7.15 Couenne::CouenneComplBranchingObject Class Reference	63
7.15.1 Detailed Description	65
7.15.2 Constructor & Destructor Documentation	65
7.15.3 Member Function Documentation	65
7.15.4 Member Data Documentation	65
7.16 Couenne::CouenneComplObject Class Reference	66
7.16.1 Detailed Description	67
7.16.2 Constructor & Destructor Documentation	67
7.16.3 Member Function Documentation	68
7.17 Couenne::CouenneConstraint Class Reference	68
7.17.1 Detailed Description	70
7.17.2 Constructor & Destructor Documentation	70
7.17.3 Member Function Documentation	70
7.17.4 Member Data Documentation	71
7.18 Couenne::CouenneCrossConv Class Reference	71
7.18.1 Detailed Description	72
7.18.2 Constructor & Destructor Documentation	73
7.18.3 Member Function Documentation	73
7.18.4 Member Data Documentation	73
7.19 Couenne::CouenneCutGenerator Class Reference	74
7.19.1 Detailed Description	76
7.19.2 Constructor & Destructor Documentation	76
7.19.3 Member Function Documentation	77
7.19.4 Member Data Documentation	79
7.20 Couenne::CouenneDisjCuts Class Reference	81
7.20.1 Detailed Description	83
7.20.2 Constructor & Destructor Documentation	84
7.20.3 Member Function Documentation	84
7.20.4 Member Data Documentation	85
7.21 Couenne::CouenneExprMatrix Class Reference	87

7.21.1	Detailed Description	88
7.21.2	Constructor & Destructor Documentation	88
7.21.3	Member Function Documentation	88
7.21.4	Member Data Documentation	89
7.22	Couenne::CouenneFeasPump Class Reference	90
7.22.1	Detailed Description	91
7.22.2	Member Enumeration Documentation	91
7.22.3	Constructor & Destructor Documentation	92
7.22.4	Member Function Documentation	92
7.23	Couenne::CouenneFixPoint Class Reference	95
7.23.1	Detailed Description	96
7.23.2	Constructor & Destructor Documentation	96
7.23.3	Member Function Documentation	96
7.23.4	Member Data Documentation	97
7.24	Couenne::CouenneFpool Class Reference	98
7.24.1	Detailed Description	98
7.24.2	Constructor & Destructor Documentation	99
7.24.3	Member Function Documentation	99
7.24.4	Member Data Documentation	99
7.25	Couenne::CouenneFPsolution Class Reference	99
7.25.1	Detailed Description	101
7.25.2	Constructor & Destructor Documentation	101
7.25.3	Member Function Documentation	101
7.25.4	Member Data Documentation	101
7.26	Couenne::CouenneInfo Class Reference	102
7.26.1	Detailed Description	104
7.26.2	Constructor & Destructor Documentation	104
7.26.3	Member Function Documentation	104
7.26.4	Member Data Documentation	105
7.27	Couenne::CouenneInterface Class Reference	105
7.27.1	Detailed Description	105
7.27.2	Constructor & Destructor Documentation	105
7.27.3	Member Function Documentation	106
7.27.4	Member Data Documentation	106
7.28	Couenne::CouenneIterativeRounding Class Reference	106
7.28.1	Detailed Description	108
7.28.2	Constructor & Destructor Documentation	108

7.28.3 Member Function Documentation	108
7.29 Couenne::CouenneMINLPInterface Class Reference	110
7.29.1 Detailed Description	110
7.29.2 Member Function Documentation	110
7.30 Couenne::CouenneMultiVarProbe Class Reference	111
7.30.1 Detailed Description	112
7.30.2 Constructor & Destructor Documentation	112
7.30.3 Member Function Documentation	112
7.30.4 Member Data Documentation	112
7.31 Couenne::CouenneObject Class Reference	113
7.31.1 Detailed Description	115
7.31.2 Member Enumeration Documentation	116
7.31.3 Constructor & Destructor Documentation	116
7.31.4 Member Function Documentation	117
7.31.5 Member Data Documentation	119
7.32 Couenne::CouenneObjective Class Reference	120
7.32.1 Detailed Description	121
7.32.2 Constructor & Destructor Documentation	122
7.32.3 Member Function Documentation	122
7.32.4 Member Data Documentation	122
7.33 Couenne::CouenneOrbitBranchingObj Class Reference	123
7.33.1 Detailed Description	124
7.33.2 Constructor & Destructor Documentation	124
7.33.3 Member Function Documentation	124
7.34 Couenne::CouenneOSInterface Class Reference	125
7.34.1 Detailed Description	126
7.34.2 Constructor & Destructor Documentation	126
7.34.3 Member Function Documentation	127
7.35 Couenne::CouenneProblem Class Reference	127
7.35.1 Detailed Description	136
7.35.2 Member Enumeration Documentation	136
7.35.3 Constructor & Destructor Documentation	136
7.35.4 Member Function Documentation	137
7.35.5 Friends And Related Function Documentation	148
7.35.6 Member Data Documentation	148
7.36 Couenne::CouennePSDcon Class Reference	154
7.36.1 Detailed Description	156

7.36.2 Constructor & Destructor Documentation	156
7.36.3 Member Function Documentation	156
7.36.4 Member Data Documentation	156
7.37 Couenne::CouenneRecordBestSol Class Reference	157
7.37.1 Detailed Description	158
7.37.2 Constructor & Destructor Documentation	159
7.37.3 Member Function Documentation	159
7.37.4 Member Data Documentation	160
7.38 Couenne::CouenneScalar Class Reference	161
7.38.1 Detailed Description	162
7.38.2 Constructor & Destructor Documentation	162
7.38.3 Member Function Documentation	163
7.38.4 Friends And Related Function Documentation	163
7.38.5 Member Data Documentation	163
7.39 Couenne::CouenneSdpCuts Class Reference	164
7.39.1 Detailed Description	165
7.39.2 Constructor & Destructor Documentation	165
7.39.3 Member Function Documentation	165
7.39.4 Member Data Documentation	166
7.40 Couenne::CouenneSetup Class Reference	167
7.40.1 Detailed Description	167
7.40.2 Constructor & Destructor Documentation	167
7.40.3 Member Function Documentation	168
7.41 Couenne::CouenneSolverInterface< T > Class Template Reference	169
7.41.1 Detailed Description	171
7.41.2 Constructor & Destructor Documentation	171
7.41.3 Member Function Documentation	171
7.41.4 Member Data Documentation	173
7.42 Couenne::CouenneSOSBranchingObject Class Reference	173
7.42.1 Detailed Description	174
7.42.2 Constructor & Destructor Documentation	175
7.42.3 Member Function Documentation	175
7.42.4 Member Data Documentation	175
7.43 Couenne::CouenneSOSObject Class Reference	176
7.43.1 Detailed Description	177
7.43.2 Constructor & Destructor Documentation	177
7.43.3 Member Function Documentation	177

7.43.4 Member Data Documentation	177
7.44 Couenne::CouenneSparseBndVec< T > Class Template Reference	178
7.44.1 Detailed Description	178
7.44.2 Constructor & Destructor Documentation	179
7.44.3 Member Function Documentation	179
7.45 Couenne::CouenneSparseMatrix Class Reference	180
7.45.1 Detailed Description	180
7.45.2 Constructor & Destructor Documentation	180
7.45.3 Member Function Documentation	181
7.46 Couenne::CouenneSparseVector Class Reference	181
7.46.1 Detailed Description	183
7.46.2 Constructor & Destructor Documentation	183
7.46.3 Member Function Documentation	183
7.46.4 Member Data Documentation	184
7.47 Couenne::CouenneThreeWayBranchObj Class Reference	184
7.47.1 Detailed Description	185
7.47.2 Constructor & Destructor Documentation	185
7.47.3 Member Function Documentation	185
7.47.4 Member Data Documentation	185
7.48 Couenne::CouenneTNLP Class Reference	186
7.48.1 Detailed Description	188
7.48.2 Constructor & Destructor Documentation	188
7.48.3 Member Function Documentation	188
7.49 Couenne::CouenneTwoImplied Class Reference	191
7.49.1 Detailed Description	192
7.49.2 Constructor & Destructor Documentation	194
7.49.3 Member Function Documentation	194
7.49.4 Member Data Documentation	194
7.50 Couenne::CouenneUserInterface Class Reference	195
7.50.1 Detailed Description	196
7.50.2 Constructor & Destructor Documentation	196
7.50.3 Member Function Documentation	196
7.50.4 Member Data Documentation	197
7.51 Couenne::CouenneVarObject Class Reference	197
7.51.1 Detailed Description	199
7.51.2 Constructor & Destructor Documentation	199
7.51.3 Member Function Documentation	200

7.51.4 Member Data Documentation	200
7.52 Couenne::CouenneVTOObject Class Reference	201
7.52.1 Detailed Description	202
7.52.2 Constructor & Destructor Documentation	202
7.52.3 Member Function Documentation	203
7.53 Couenne::CouExpr Class Reference	203
7.53.1 Detailed Description	203
7.53.2 Constructor & Destructor Documentation	203
7.53.3 Member Function Documentation	204
7.54 Couenne::DepGraph Class Reference	204
7.54.1 Detailed Description	205
7.54.2 Constructor & Destructor Documentation	205
7.54.3 Member Function Documentation	206
7.54.4 Member Data Documentation	207
7.55 Couenne::DepNode Class Reference	207
7.55.1 Detailed Description	208
7.55.2 Member Enumeration Documentation	209
7.55.3 Constructor & Destructor Documentation	209
7.55.4 Member Function Documentation	209
7.55.5 Member Data Documentation	210
7.56 Couenne::Domain Class Reference	210
7.56.1 Detailed Description	212
7.56.2 Constructor & Destructor Documentation	212
7.56.3 Member Function Documentation	212
7.56.4 Member Data Documentation	213
7.57 Couenne::DomainPoint Class Reference	214
7.57.1 Detailed Description	215
7.57.2 Constructor & Destructor Documentation	215
7.57.3 Member Function Documentation	215
7.57.4 Friends And Related Function Documentation	216
7.57.5 Member Data Documentation	217
7.58 Couenne::exprAbs Class Reference	217
7.58.1 Detailed Description	219
7.58.2 Constructor & Destructor Documentation	219
7.58.3 Member Function Documentation	220
7.59 Couenne::exprAux Class Reference	221
7.59.1 Detailed Description	225

7.59.2 Member Enumeration Documentation	225
7.59.3 Constructor & Destructor Documentation	226
7.59.4 Member Function Documentation	226
7.59.5 Member Data Documentation	229
7.60 Couenne::exprBinProd Class Reference	230
7.60.1 Detailed Description	231
7.60.2 Constructor & Destructor Documentation	231
7.60.3 Member Function Documentation	231
7.61 Couenne::exprCeil Class Reference	232
7.61.1 Detailed Description	234
7.61.2 Constructor & Destructor Documentation	234
7.61.3 Member Function Documentation	234
7.62 Couenne::exprClone Class Reference	236
7.62.1 Detailed Description	238
7.62.2 Constructor & Destructor Documentation	238
7.62.3 Member Function Documentation	238
7.63 Couenne::exprConst Class Reference	239
7.63.1 Detailed Description	241
7.63.2 Constructor & Destructor Documentation	241
7.63.3 Member Function Documentation	241
7.64 Couenne::exprCopy Class Reference	243
7.64.1 Detailed Description	246
7.64.2 Constructor & Destructor Documentation	246
7.64.3 Member Function Documentation	247
7.64.4 Member Data Documentation	252
7.65 Couenne::exprCos Class Reference	252
7.65.1 Detailed Description	254
7.65.2 Constructor & Destructor Documentation	254
7.65.3 Member Function Documentation	254
7.66 Couenne::exprDiv Class Reference	256
7.66.1 Detailed Description	258
7.66.2 Constructor & Destructor Documentation	259
7.66.3 Member Function Documentation	259
7.67 Couenne::expression Class Reference	261
7.67.1 Detailed Description	265
7.67.2 Member Enumeration Documentation	265
7.67.3 Constructor & Destructor Documentation	265

7.67.4 Member Function Documentation	266
7.68 Couenne::exprExp Class Reference	273
7.68.1 Detailed Description	274
7.68.2 Constructor & Destructor Documentation	274
7.68.3 Member Function Documentation	275
7.69 Couenne::exprFloor Class Reference	276
7.69.1 Detailed Description	278
7.69.2 Constructor & Destructor Documentation	278
7.69.3 Member Function Documentation	278
7.70 Couenne::exprGroup Class Reference	280
7.70.1 Detailed Description	284
7.70.2 Member Typedef Documentation	284
7.70.3 Constructor & Destructor Documentation	284
7.70.4 Member Function Documentation	284
7.70.5 Member Data Documentation	287
7.71 Couenne::ExprHess Class Reference	287
7.71.1 Detailed Description	287
7.71.2 Constructor & Destructor Documentation	288
7.71.3 Member Function Documentation	288
7.72 Couenne::exprIf Class Reference	288
7.72.1 Detailed Description	289
7.73 Couenne::exprInv Class Reference	290
7.73.1 Detailed Description	291
7.73.2 Constructor & Destructor Documentation	292
7.73.3 Member Function Documentation	292
7.74 Couenne::exprVar Class Reference	293
7.74.1 Detailed Description	296
7.74.2 Constructor & Destructor Documentation	296
7.74.3 Member Function Documentation	296
7.75 Couenne::ExprJac Class Reference	297
7.75.1 Detailed Description	297
7.75.2 Constructor & Destructor Documentation	297
7.75.3 Member Function Documentation	297
7.76 Couenne::exprLBCos Class Reference	298
7.76.1 Detailed Description	299
7.76.2 Constructor & Destructor Documentation	299
7.76.3 Member Function Documentation	300

7.77 Couenne::exprLBDiv Class Reference	300
7.77.1 Detailed Description	302
7.77.2 Constructor & Destructor Documentation	302
7.77.3 Member Function Documentation	302
7.78 Couenne::exprLBMul Class Reference	303
7.78.1 Detailed Description	304
7.78.2 Constructor & Destructor Documentation	304
7.78.3 Member Function Documentation	304
7.79 Couenne::exprLBQuad Class Reference	305
7.79.1 Detailed Description	306
7.79.2 Constructor & Destructor Documentation	306
7.79.3 Member Function Documentation	306
7.80 Couenne::exprLBSin Class Reference	307
7.80.1 Detailed Description	308
7.80.2 Constructor & Destructor Documentation	308
7.80.3 Member Function Documentation	309
7.81 Couenne::exprLog Class Reference	309
7.81.1 Detailed Description	311
7.81.2 Constructor & Destructor Documentation	311
7.81.3 Member Function Documentation	312
7.82 Couenne::exprLowerBound Class Reference	313
7.82.1 Detailed Description	316
7.82.2 Constructor & Destructor Documentation	316
7.82.3 Member Function Documentation	316
7.83 Couenne::exprMax Class Reference	317
7.83.1 Detailed Description	319
7.83.2 Constructor & Destructor Documentation	319
7.83.3 Member Function Documentation	319
7.84 Couenne::exprMin Class Reference	321
7.84.1 Detailed Description	323
7.84.2 Constructor & Destructor Documentation	323
7.84.3 Member Function Documentation	323
7.85 Couenne::exprMultiLin Class Reference	325
7.85.1 Detailed Description	326
7.85.2 Constructor & Destructor Documentation	326
7.85.3 Member Function Documentation	326
7.86 Couenne::exprNorm Class Reference	327

7.86.1	Detailed Description	328
7.87	Couenne::exprOddPow Class Reference	329
7.87.1	Detailed Description	331
7.87.2	Constructor & Destructor Documentation	331
7.87.3	Member Function Documentation	331
7.88	Couenne::exprOp Class Reference	333
7.88.1	Detailed Description	335
7.88.2	Constructor & Destructor Documentation	335
7.88.3	Member Function Documentation	336
7.88.4	Member Data Documentation	338
7.89	Couenne::exprOpp Class Reference	339
7.89.1	Detailed Description	341
7.89.2	Constructor & Destructor Documentation	341
7.89.3	Member Function Documentation	341
7.90	Couenne::exprPow Class Reference	343
7.90.1	Detailed Description	345
7.90.2	Constructor & Destructor Documentation	345
7.90.3	Member Function Documentation	345
7.91	Couenne::exprPWLinear Class Reference	348
7.91.1	Detailed Description	349
7.92	Couenne::exprQuad Class Reference	349
7.92.1	Detailed Description	352
7.92.2	Member Typedef Documentation	353
7.92.3	Constructor & Destructor Documentation	353
7.92.4	Member Function Documentation	353
7.92.5	Member Data Documentation	357
7.93	Couenne::exprSin Class Reference	358
7.93.1	Detailed Description	360
7.93.2	Constructor & Destructor Documentation	360
7.93.3	Member Function Documentation	360
7.94	Couenne::exprStore Class Reference	362
7.94.1	Detailed Description	363
7.94.2	Constructor & Destructor Documentation	364
7.94.3	Member Function Documentation	364
7.94.4	Member Data Documentation	364
7.95	Couenne::exprSub Class Reference	365
7.95.1	Detailed Description	366

7.95.2 Constructor & Destructor Documentation	366
7.95.3 Member Function Documentation	367
7.96 Couenne::exprSum Class Reference	368
7.96.1 Detailed Description	371
7.96.2 Constructor & Destructor Documentation	371
7.96.3 Member Function Documentation	371
7.97 Couenne::exprTrilinear Class Reference	373
7.97.1 Detailed Description	374
7.97.2 Constructor & Destructor Documentation	374
7.97.3 Member Function Documentation	374
7.98 Couenne::exprUBCos Class Reference	375
7.98.1 Detailed Description	377
7.98.2 Constructor & Destructor Documentation	377
7.98.3 Member Function Documentation	377
7.99 Couenne::exprUBDiv Class Reference	378
7.99.1 Detailed Description	379
7.99.2 Constructor & Destructor Documentation	379
7.99.3 Member Function Documentation	379
7.100 Couenne::exprUBMul Class Reference	380
7.100.1 Detailed Description	381
7.100.2 Constructor & Destructor Documentation	381
7.100.3 Member Function Documentation	382
7.101 Couenne::exprUBQuad Class Reference	382
7.101.1 Detailed Description	384
7.101.2 Constructor & Destructor Documentation	384
7.101.3 Member Function Documentation	384
7.102 Couenne::exprUBSin Class Reference	385
7.102.1 Detailed Description	386
7.102.2 Constructor & Destructor Documentation	386
7.102.3 Member Function Documentation	386
7.103 Couenne::exprUnary Class Reference	387
7.103.1 Detailed Description	389
7.103.2 Constructor & Destructor Documentation	389
7.103.3 Member Function Documentation	389
7.103.4 Member Data Documentation	392
7.104 Couenne::exprUpperBound Class Reference	392
7.104.1 Detailed Description	395

7.104.2 Constructor & Destructor Documentation	395
7.104.3 Member Function Documentation	395
7.105 Couenne::exprVar Class Reference	396
7.105.1 Detailed Description	399
7.105.2 Constructor & Destructor Documentation	399
7.105.3 Member Function Documentation	400
7.105.4 Member Data Documentation	404
7.106 Couenne::funtriplet Class Reference	405
7.106.1 Detailed Description	405
7.106.2 Constructor & Destructor Documentation	405
7.106.3 Member Function Documentation	406
7.107 Couenne::GlobalCutOff Class Reference	406
7.107.1 Detailed Description	406
7.107.2 Constructor & Destructor Documentation	406
7.107.3 Member Function Documentation	406
7.108 Couenne::InitHeuristic Class Reference	407
7.108.1 Detailed Description	407
7.108.2 Constructor & Destructor Documentation	407
7.108.3 Member Function Documentation	408
7.109 Couenne::kpowertriplet Class Reference	408
7.109.1 Detailed Description	410
7.109.2 Constructor & Destructor Documentation	410
7.109.3 Member Function Documentation	410
7.109.4 Member Data Documentation	410
7.110 less_than_struct Struct Reference	411
7.110.1 Detailed Description	411
7.110.2 Member Function Documentation	411
7.111 Couenne::LinMap Class Reference	411
7.111.1 Detailed Description	411
7.111.2 Member Function Documentation	412
7.112 Couenne::MultiProdRel Class Reference	412
7.112.1 Detailed Description	413
7.112.2 Member Function Documentation	413
7.113 myclass Struct Reference	413
7.113.1 Detailed Description	414
7.113.2 Member Function Documentation	414
7.114 myclass0 Struct Reference	414

7.114.1 Detailed Description	414
7.114.2 Member Function Documentation	414
7.115Nauty Class Reference	414
7.115.1 Detailed Description	415
7.115.2 Member Enumeration Documentation	415
7.115.3 Constructor & Destructor Documentation	415
7.115.4 Member Function Documentation	415
7.116Couenne::CcouenneInfo::NlpSolution Class Reference	416
7.116.1 Detailed Description	417
7.116.2 Constructor & Destructor Documentation	417
7.116.3 Member Function Documentation	417
7.117Couenne::NlpSolveHeuristic Class Reference	417
7.117.1 Detailed Description	418
7.117.2 Constructor & Destructor Documentation	418
7.117.3 Member Function Documentation	419
7.118Node Class Reference	419
7.118.1 Detailed Description	420
7.118.2 Member Function Documentation	420
7.119Couenne::powertriplet Class Reference	421
7.119.1 Detailed Description	422
7.119.2 Constructor & Destructor Documentation	422
7.119.3 Member Function Documentation	422
7.119.4 Member Data Documentation	423
7.120Couenne::PowRel Class Reference	423
7.120.1 Detailed Description	424
7.120.2 Member Function Documentation	424
7.121Couenne::Qroot Class Reference	424
7.121.1 Detailed Description	425
7.121.2 Constructor & Destructor Documentation	426
7.121.3 Member Function Documentation	426
7.121.4 Member Data Documentation	426
7.122Couenne::quadElem Class Reference	426
7.122.1 Detailed Description	426
7.122.2 Constructor & Destructor Documentation	427
7.122.3 Member Function Documentation	427
7.123Couenne::QuadMap Class Reference	427
7.123.1 Detailed Description	427

7.123.2 Member Function Documentation	427
7.124 Couenne::simpletriplet Class Reference	428
7.124.1 Detailed Description	429
7.124.2 Constructor & Destructor Documentation	429
7.124.3 Member Function Documentation	429
7.124.4 Member Data Documentation	430
7.125 Couenne::SmartAsl Class Reference	430
7.125.1 Detailed Description	430
7.125.2 Constructor & Destructor Documentation	431
7.125.3 Member Data Documentation	431
7.126 Couenne::SumLogAuxRel Class Reference	431
7.126.1 Detailed Description	432
7.126.2 Member Function Documentation	432
7.127 Couenne::t_chg_bounds Class Reference	432
7.127.1 Detailed Description	433
7.127.2 Member Enumeration Documentation	433
7.127.3 Constructor & Destructor Documentation	433
7.127.4 Member Function Documentation	433
8 File Documentation	434
8.1 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneAggrProbing.hpp File Reference	434
8.2 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneBTPerIndicator.hpp File Reference	435
8.3 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneFixPoint.hpp File Reference	436
8.4 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneInfeasCut.hpp File Reference	436
8.4.1 Function Documentation	437
8.5 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneMultiVarProbe.hpp File Reference	437
8.6 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneSparseBndVec.hpp File Reference	438
8.7 /home/ted/COIN/trunk/Couenne/src/bound_tightening/twolImpliedBT/CouenneTwoImplied.hpp File Reference	438
8.8 /home/ted/COIN/trunk/Couenne/src/branch/CouenneBranchingObject.hpp File Reference	439
8.8.1 Macro Definition Documentation	441
8.9 /home/ted/COIN/trunk/Couenne/src/branch/CouenneChooseStrong.hpp File Reference	441
8.10 /home/ted/COIN/trunk/Couenne/src/branch/CouenneChooseVariable.hpp File Reference	442
8.11 /home/ted/COIN/trunk/Couenne/src/branch/CouenneComplBranchingObject.hpp File Reference	444
8.12 /home/ted/COIN/trunk/Couenne/src/branch/CouenneComplObject.hpp File Reference	445
8.13 /home/ted/COIN/trunk/Couenne/src/branch/CouenneObject.hpp File Reference	445
8.13.1 Macro Definition Documentation	447
8.14 /home/ted/COIN/trunk/Couenne/src/branch/CouenneOrbitBranchingObj.hpp File Reference	447

8.15 /home/ted/COIN/trunk/Couenne/src/branch/CouenneOrbitObj.hpp File Reference	448
8.16 /home/ted/COIN/trunk/Couenne/src/branch/CouenneProjections.hpp File Reference	448
8.17 /home/ted/COIN/trunk/Couenne/src/branch/CouenneSOSObject.hpp File Reference	449
8.18 /home/ted/COIN/trunk/Couenne/src/branch/CouenneThreeWayBranchObj.hpp File Reference	450
8.19 /home/ted/COIN/trunk/Couenne/src/branch/CouenneVarObject.hpp File Reference	452
8.20 /home/ted/COIN/trunk/Couenne/src/branch/CouenneVTOBJECT.hpp File Reference	453
8.21 /home/ted/COIN/trunk/Couenne/src/branch/Nauty.h File Reference	453
8.22 /home/ted/COIN/trunk/Couenne/src/config_couenne_default.h File Reference	454
8.22.1 Macro Definition Documentation	454
8.23 /home/ted/COIN/trunk/Couenne/src/config_default.h File Reference	455
8.23.1 Macro Definition Documentation	455
8.24 /home/ted/COIN/trunk/Couenne/src/convex/CouenneCutGenerator.hpp File Reference	456
8.25 /home/ted/COIN/trunk/Couenne/src/CouenneConfig.h File Reference	457
8.26 /home/ted/COIN/trunk/Couenne/src/cut/crossconv/CouenneCrossConv.hpp File Reference	457
8.27 /home/ted/COIN/trunk/Couenne/src/cut/ellipcuts/CouenneEllipCuts.hpp File Reference	458
8.28 /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneMatrix.hpp File Reference	458
8.29 /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouennePSDcon.hpp File Reference	459
8.30 /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneSdpCuts.hpp File Reference	460
8.31 /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/dsyevx_wrapper.hpp File Reference	461
8.31.1 Function Documentation	461
8.32 /home/ted/COIN/trunk/Couenne/src/disjunctive/CouenneDisjCuts.hpp File Reference	461
8.33 /home/ted/COIN/trunk/Couenne/src/expression/CouenneDomain.hpp File Reference	462
8.34 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprAux.hpp File Reference	463
8.35 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprBound.hpp File Reference	465
8.36 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprClone.hpp File Reference	466
8.37 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprConst.hpp File Reference	467
8.38 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprCopy.hpp File Reference	468
8.39 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExpression.hpp File Reference	469
8.40 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprVar.hpp File Reference	470
8.41 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprOp.hpp File Reference	470
8.41.1 Macro Definition Documentation	471
8.42 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprStore.hpp File Reference	472
8.43 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprUnary.hpp File Reference	473
8.44 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprVar.hpp File Reference	474
8.45 /home/ted/COIN/trunk/Couenne/src/expression/CouennePrecisions.hpp File Reference	475
8.45.1 Macro Definition Documentation	476
8.46 /home/ted/COIN/trunk/Couenne/src/expression/CouenneTypes.hpp File Reference	477

8.47 /home/ted/COIN/trunk/Couenne/src/expression/CouExpr.hpp File Reference	479
8.48 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBCos.hpp File Reference	480
8.48.1 Macro Definition Documentation	481
8.49 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBDiv.hpp File Reference	481
8.50 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBMul.hpp File Reference	482
8.50.1 Macro Definition Documentation	483
8.51 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBQuad.hpp File Reference	483
8.52 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBSin.hpp File Reference	484
8.52.1 Macro Definition Documentation	485
8.53 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprAbs.hpp File Reference	485
8.54 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprBinProd.hpp File Reference	485
8.55 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprCeil.hpp File Reference	487
8.56 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprCos.hpp File Reference	488
8.57 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprDiv.hpp File Reference	488
8.57.1 Macro Definition Documentation	489
8.58 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprEvenPow.hpp File Reference	490
8.59 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprExp.hpp File Reference	490
8.60 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprFloor.hpp File Reference	492
8.61 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprGroup.hpp File Reference	492
8.62 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprIf.hpp File Reference	494
8.63 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprInv.hpp File Reference	494
8.64 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprLog.hpp File Reference	496
8.65 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprMax.hpp File Reference	497
8.66 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprMin.hpp File Reference	498
8.67 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprMul.hpp File Reference	498
8.68 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprMultiLin.hpp File Reference	499
8.69 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprNorm.hpp File Reference	500
8.70 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprOddPow.hpp File Reference	501
8.71 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprOpp.hpp File Reference	502
8.72 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprPow.hpp File Reference	503
8.73 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprPWLinear.hpp File Reference	505
8.74 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprQuad.hpp File Reference	505
8.75 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprSignPow.hpp File Reference	507
8.76 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprSin.hpp File Reference	507
8.77 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprSub.hpp File Reference	509
8.78 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprSum.hpp File Reference	510

8.79 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprTrilinear.hpp File Reference	511
8.80 /home/ted/COIN/trunk/Couenne/src/expression/partial/CouenneExprHess.hpp File Reference	513
8.81 /home/ted/COIN/trunk/Couenne/src/expression/partial/CouenneExprJac.hpp File Reference	514
8.82 /home/ted/COIN/trunk/Couenne/src/heuristics/BonInitHeuristic.hpp File Reference	514
8.83 /home/ted/COIN/trunk/Couenne/src/heuristics/BonNlpHeuristic.hpp File Reference	515
8.84 /home/ted/COIN/trunk/Couenne/src/heuristics/cons_rowcuts.h File Reference	516
8.84.1 Detailed Description	516
8.85 /home/ted/COIN/trunk/Couenne/src/heuristics/CouenneFeasPump.hpp File Reference	516
8.85.1 Function Documentation	517
8.86 /home/ted/COIN/trunk/Couenne/src/heuristics/CouenneFPPool.hpp File Reference	517
8.87 /home/ted/COIN/trunk/Couenne/src/heuristics/CouenneIterativeRounding.hpp File Reference	519
8.88 /home/ted/COIN/trunk/Couenne/src/interfaces/BonCouenneInterface.hpp File Reference	520
8.88.1 Macro Definition Documentation	520
8.89 /home/ted/COIN/trunk/Couenne/src/interfaces/CouenneMINLPInterface.hpp File Reference	520
8.90 /home/ted/COIN/trunk/Couenne/src/interfaces/CouenneTNLP.hpp File Reference	521
8.91 /home/ted/COIN/trunk/Couenne/src/interfaces/CouenneUserInterface.hpp File Reference	523
8.92 /home/ted/COIN/trunk/Couenne/src/main/BonCouenneInfo.hpp File Reference	524
8.93 /home/ted/COIN/trunk/Couenne/src/main/BonCouenneSetup.hpp File Reference	524
8.94 /home/ted/COIN/trunk/Couenne/src/main/CouenneBab.hpp File Reference	525
8.95 /home/ted/COIN/trunk/Couenne/src/main/CouenneOSInterface.hpp File Reference	526
8.96 /home/ted/COIN/trunk/Couenne/src/problem/CouenneGlobalCutOff.hpp File Reference	527
8.97 /home/ted/COIN/trunk/Couenne/src/problem/CouenneJournalist.hpp File Reference	527
8.98 /home/ted/COIN/trunk/Couenne/src/problem/CouenneProblem.hpp File Reference	528
8.98.1 Macro Definition Documentation	530
8.99 /home/ted/COIN/trunk/Couenne/src/problem/CouenneProblemElem.hpp File Reference	530
8.100 /home/ted/COIN/trunk/Couenne/src/problem/CouenneRecordBestSol.hpp File Reference	532
8.101 /home/ted/COIN/trunk/Couenne/src/problem/CouenneSolverInterface.hpp File Reference	532
8.102 /home/ted/COIN/trunk/Couenne/src/problem/depGraph/CouenneDepGraph.hpp File Reference	533
8.103 /home/ted/COIN/trunk/Couenne/src/readnl/CouenneAmplInterface.hpp File Reference	534
8.104 /home/ted/COIN/trunk/Couenne/src/standardize/CouenneLQelems.hpp File Reference	535
8.105 /home/ted/COIN/trunk/Couenne/src/util/CouenneFunTriplets.hpp File Reference	535
8.106 /home/ted/COIN/trunk/Couenne/src/util/CouenneRootQ.hpp File Reference	536
8.107 /home/ted/COIN/trunk/Couenne/src/util/CouenneSparseMatrix.hpp File Reference	537

1 Todo List

Member Couenne::InitHeuristic::solution (double &objectiveValue, double *newSolution)

Find a quicker way to get to Couenne objects, store them or something

Member Couenne::NlpSolveHeuristic::solution (double &objectiveValue, double *newSolution)

Find a quicker way to get to Couenne objects, store them or something

2 Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Bonmin	19
Coin	19
Couenne General include file for different compilers	19
Ipopt	38
Osi	38

3 Class Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

_EKKfactinfo [external]
doubleton_action::action [external]
forcing_constraint_action::action [external]
remove_fixed_action::action [external]
tripleton_action::action [external]

Couenne::AuxRelation	38
Couenne::BiProdDivRel	39
Couenne::MultiProdRel	412
Couenne::PowRel	423
Couenne::SumLogAuxRel	431
std::basic_fstream< char >	
std::basic_fstream< wchar_t >	
std::basic_ifstream< char >	
std::basic_ifstream< wchar_t >	
std::basic_ios< char >	
std::basic_ios< wchar_t >	

```
std::basic_iostream< char >
std::basic_iostream< wchar_t >
std::basic_istream< char >
std::basic_istream< wchar_t >
std::basic_iostreamstream< char >
std::basic_iostreamstream< wchar_t >
std::basic_ofstream< char >
std::basic_ofstream< wchar_t >
std::basic_ostream< char >
std::basic_ostream< wchar_t >
std::basic_ostringstream< char >
std::basic_ostringstream< wchar_t >
std::basic_string< char >
std::basic_string< wchar_t >
std::basic_stringstream< char >
std::basic_stringstream< wchar_t >
BitVector128 [external]
CoinAbsFltEq [external]
CoinArrayWithLength [external]
    CoinArbitraryArrayWithLength [external]
    CoinBigIndexArrayWithLength [external]
    CoinDoubleArrayWithLength [external]
    CoinFactorizationDoubleArrayWithLength [external]
    CoinFactorizationLongDoubleArrayWithLength [external]
    CoinIntArrayWithLength [external]
    CoinUnsignedIntArrayWithLength [external]
    CoinVoidStarArrayWithLength [external]
CoinBaseModel [external]
    CoinModel [external]
    CoinStructuredModel [external]
CoinBuild [external]
CoinDenseVector< T > [external]
CoinError [external]
    CoinExternalVectorFirstGreater_2< class, class, class > [external]
    CoinExternalVectorFirstGreater_3< class, class, class, class > [external]
    CoinExternalVectorFirstLess_2< class, class, class > [external]
    CoinExternalVectorFirstLess_3< class, class, class, class > [external]
    CoinFactorization [external]
    CoinFileIOBase [external]
        CoinFileInput [external]
        CoinFileOutput [external]
    CoinFirstAbsGreater_2< class, class > [external]
    CoinFirstAbsGreater_3< class, class, class > [external]
    CoinFirstAbsLess_2< class, class > [external]
    CoinFirstAbsLess_3< class, class, class > [external]
    CoinFirstGreater_2< class, class > [external]
    CoinFirstGreater_3< class, class, class > [external]
    CoinFirstLess_2< class, class > [external]
    CoinFirstLess_3< class, class, class > [external]
    CoinLpIO::CoinHashLink [external]
    CoinMpsIO::CoinHashLink [external]
    CoinIndexedVector [external]
        CoinPartitionedVector [external]
    CoinLpIO [external]
```

```
CoinMessageHandler [external]
CoinMessages [external]
    CoinMessage [external]
CoinModelHash [external]
CoinModelHash2 [external]
CoinModelHashLink [external]
CoinModelInfo2 [external]
CoinModelLink [external]
CoinModelLinkedList [external]
CoinModelTriple [external]
CoinMpsCardReader [external]
CoinMpsIO [external]
CoinOneMessage [external]
CoinOtherFactorization [external]
    CoinDenseFactorization [external]
    CoinOslFactorization [external]
    CoinSimpFactorization [external]
CoinPackedMatrix [external]
CoinPackedVectorBase [external]
    CoinPackedVector [external]
    CoinShallowPackedVector [external]
CoinPair< S, T > [external]
CoinParam [external]
CoinPrePostsolveMatrix [external]
    CoinPostsolveMatrix [external]
    CoinPresolveMatrix [external]
CoinPresolveAction [external]
    do_tighten_action [external]
    doubleton_action [external]
    drop_empty_cols_action [external]
    drop_empty_rows_action [external]
    drop_zero_coefficients_action [external]
    dupcol_action [external]
    duprow_action [external]
    forcing_constraint_action [external]
    gubrow_action [external]
    implied_free_action [external]
    isolated_constraint_action [external]
    make_fixed_action [external]
    remove_dual_action [external]
    remove_fixed_action [external]
    slack_doubleton_action [external]
    slack_singleton_action [external]
    subst_constraint_action [external]
    tripleton_action [external]
    twoxtwo_action [external]
    useless_constraint_action [external]
CoinPresolveMonitor [external]
CoinRational [external]
CoinRelFltEq [external]
CoinSearchTreeBase [external]
    CoinSearchTree< class > [external]
CoinSearchTreeCompareBest [external]
CoinSearchTreeCompareBreadth [external]
```

CoinSearchTreeCompareDepth [external]	
CoinSearchTreeComparePreferred [external]	
CoinSearchTreeManager [external]	
CoinSet [external]	
CoinSosSet [external]	
CoinSnapshot [external]	
CoinThreadRandom [external]	
CoinTimer [external]	
CoinTreeNode [external]	
CoinTreeSiblings [external]	
CoinTriple< S, T, U > [external]	
CoinWarmStart [external]	
CoinWarmStartBasis [external]	
CoinWarmStartDual [external]	
CoinWarmStartPrimalDual [external]	
CoinWarmStartVector< T > [external]	
CoinWarmStartVectorPair< T, U > [external]	
CoinWarmStartDiff [external]	
CoinWarmStartBasisDiff [external]	
CoinWarmStartDualDiff [external]	
CoinWarmStartPrimalDualDiff [external]	
CoinWarmStartVectorDiff< T > [external]	
CoinWarmStartVectorPairDiff< T, U > [external]	
CoinYacc [external]	
Couenne::CouenneExprMatrix::compare_pair_ind	40
Couenne::CouenneSparseVector::compare_scalars	40
Couenne::compareSol	41
Couenne::compExpr	41
Couenne::compNode	42
Couenne::CouenneAggrProbing	42
Couenne::CouenneBab	48
Couenne::CouenneBranchingObject	50
Couenne::CouenneComplBranchingObject	63
Couenne::CouenneOrbitBranchingObj	123
Couenne::CouenneBTPerIndicator	54
Couenne::CouenneChooseStrong	58
Couenne::CouenneChooseVariable	61
Couenne::CouenneConstraint	68
Couenne::CouennePSDcon	154
Couenne::CouenneCrossConv	71

Couenne::CouenneCutGenerator	74
Couenne::CouenneDisjCuts	81
Couenne::CouenneExprMatrix	87
Couenne::CouenneFeasPump	90
Couenne::CouenneFixPoint	95
Couenne::CouenneFPpool	98
Couenne::CouenneFPSolution	99
Couenne::CouenneInfo	102
Couenne::CouenneInterface	105
Couenne::CouenneIterativeRounding	106
Couenne::CouenneMINLPInterface	110
Couenne::CouenneMultiVarProbe	111
Couenne::CouenneObject	113
Couenne::CouenneComplObject	66
Couenne::CouenneVarObject	197
Couenne::CouenneVTOObject	201
Couenne::CouenneObjective	120
Couenne::CouenneProblem	127
Couenne::CouenneRecordBestSol	157
Couenne::CouenneScalar	161
Couenne::CouenneSdpCuts	164
Couenne::CouenneSetup	167
Couenne::CouenneSolverInterface< T >	169
Couenne::CouenneSOSBranchingObject	173
Couenne::CouenneSOSObject	176
Couenne::CouenneSparseBndVec< T >	178
Couenne::CouenneSparseMatrix	180
Couenne::CouenneSparseVector	181
Couenne::CouenneThreeWayBranchObj	184
Couenne::CouenneTNLP	186

Couenne::CouenneTwoImplied	191
Couenne::CouenneUserInterface	195
Couenne::CouenneAmplInterface	46
Couenne::CouenneOSInterface	125
Couenne::CouExpr	203
Couenne::DepGraph	204
Couenne::DepNode	207
Couenne::Domain	210
Couenne::DomainPoint	214
dropped_zero [external]	
EKKHlink [external]	
Couenne::exprBinProd	230
Couenne::expression	261
Couenne::exprConst	239
Couenne::exprCopy	243
Couenne::exprClone	236
Couenne::exprStore	362
Couenne::exprLBQuad	305
Couenne::exprOp	333
Couenne::exprDiv	256
Couenne::exprIf	288
Couenne::exprLBCos	298
Couenne::exprLBDiv	300
Couenne::exprLBMul	303
Couenne::exprLBSin	307
Couenne::exprMax	317
Couenne::exprMin	321
Couenne::exprNorm	327
Couenne::exprPow	343
Couenne::exprOddPow	329
Couenne::exprPWLinear	348

Couenne::exprSub	365
Couenne::exprSum	368
Couenne::exprGroup	280
Couenne::exprQuad	349
Couenne::exprUBCos	375
Couenne::exprUBDiv	378
Couenne::exprUBMul	380
Couenne::exprUBSin	385
Couenne::exprUBQuad	382
Couenne::exprUnary	387
Couenne::exprAbs	217
Couenne::exprCeil	232
Couenne::exprCos	252
Couenne::exprExp	273
Couenne::exprFloor	276
Couenne::exprInv	290
Couenne::exprLog	309
Couenne::exprOpp	339
Couenne::exprSin	358
Couenne::exprVar	396
Couenne::exprAux	221
Couenne::exprVar	293
Couenne::exprLowerBound	313
Couenne::exprUpperBound	392
Couenne::ExprHess	287
Couenne::ExprJac	297
Couenne::exprMultiLin	325
Couenne::exprTrilinear	373
FactorPointers [external]	
Couenne::funtriplet	405

Couenne::powertriplet	421
Couenne::kpowertriplet	408
Couenne::simpletriplet	428
Couenne::GlobalCutOff	406
Couenne::InitHeuristic	407
less_than_str	411
Couenne::LinMap	411
myclass	413
myclass0	414
Nauty	414
Couenne::CouenneInfo::NlpSolution	416
Couenne::NlpSolveHeuristic	417
Node	419
presolvehlink [external]	
Couenne::Qroot	424
Couenne::quadElem	426
Couenne::QuadMap	427
ReferencedObject [external]	
Couenne::SmartAsl	430
SmartPtr< T > [external]	
symrec [external]	
Couenne::t_chg_bounds	432

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Couenne::AuxRelation	
Base class definition for relations between auxiliaries	38
Couenne::BiProdDivRel	
Identifies 5-tuple of the form	39
Couenne::CouenneExprMatrix::compare_pair_ind	40
Couenne::CouenneSparseVector::compare_scalars	40

Couenne::compareSol	Class for comparing solutions (used in tabu list)	41
Couenne::compExpr	Structure for comparing expressions	41
Couenne::compNode	Structure for comparing nodes in the dependence graph	42
Couenne::CouenneAggrProbing	Cut Generator for aggressive BT; i.e., an aggressive probing	42
Couenne::CouenneAmplInterface		46
Couenne::CouenneBab		48
Couenne::CouenneBranchingObject	"Spatial" branching object	50
Couenne::CouenneBTPerfIndicator		54
Couenne::CouenneChooseStrong		58
Couenne::CouenneChooseVariable	Choose a variable for branching	61
Couenne::CouenneComplBranchingObject	"Spatial" branching object for complementarity constraints	63
Couenne::CouenneComplObject	OsiObject for complementarity constraints $x_1x_2 \geq, \leq, = 0$	66
Couenne::CouenneConstraint	Class to represent nonlinear constraints	68
Couenne::CouenneCrossConv	Cut Generator that uses relationships between auxiliaries	71
Couenne::CouenneCutGenerator	Cut Generator for linear convexifications	74
Couenne::CouenneDisjCuts	Cut Generator for linear convexifications	81
Couenne::CouenneExprMatrix		87
Couenne::CouenneFeasPump	An implementation of the Feasibility pump that uses linearization and Ipopt to find the two sequences of points	90
Couenne::CouenneFixPoint	Cut Generator for FBBT fixpoint	95
Couenne::CouenneFPpool	Pool of solutions	98

Couenne::CouenneFPSolution	Class containing a solution with infeasibility evaluation	99
Couenne::CouenneInfo	Bonmin class for passing info between components of branch-and-cuts	102
Couenne::CouenneInterface		105
Couenne::CouenneIterativeRounding	An iterative rounding heuristic, tailored for nonconvex MINLPs	106
Couenne::CouenneMINLPInterface	This is class provides an Osi interface for a Mixed Integer Linear Program expressed as a TMINLP (so that we can use it for example as the continuous solver in Cbc)	110
Couenne::CouenneMultiVarProbe		111
Couenne::CouenneObject	OsiObject for auxiliary variables $w=f(x)$	113
Couenne::CouenneObjective	Objective function	120
Couenne::CouenneOrbitBranchingObj	"Spatial" branching object	123
Couenne::CouenneOSInterface		125
Couenne::CouenneProblem	Class for MINLP problems with symbolic information	127
Couenne::CouennePSDcon	Class to represent positive semidefinite constraints ///////////////	154
Couenne::CouenneRecordBestSol		157
Couenne::CouenneScalar		161
Couenne::CouenneSdpCuts	These are cuts of the form	164
Couenne::CouenneSetup		167
Couenne::CouenneSolverInterface< T >	Solver interface class with a pointer to a Couenne cut generator	169
Couenne::CouenneSOSBranchingObject		173
Couenne::CouenneSOSObject		176
Couenne::CouenneSparseBndVec< T >		178
Couenne::CouenneSparseMatrix	Class for sparse Matrixs (used in modifying distances in FP)	180
Couenne::CouenneSparseVector		181

Couenne::CouenneThreeWayBranchObj	Spatial, three-way branching object	184
Couenne::CouenneTNLP	Class for handling NLPs using CouenneProblem	186
Couenne::CouenneTwoImplied	Cut Generator for implied bounds derived from pairs of linear (in)equalities	191
Couenne::CouenneUserInterface		195
Couenne::CouenneVarObject	OsiObject for variables in a MINLP	197
Couenne::CouenneVTOBJECT	OsiObject for violation transfer on variables in a MINLP	201
Couenne::CouExpr		203
Couenne::DepGraph	Dependence graph	204
Couenne::DepNode	Vertex of a dependence graph	207
Couenne::Domain	Define a dynamic point+bounds, with a way to save and restore previous points+bounds through a LIFO structure	210
Couenne::DomainPoint	Define a point in the solution space and the bounds around it	214
Couenne::exprAbs	Class for $ f(x) $	217
Couenne::exprAux	Auxiliary variable	221
Couenne::exprBinProd	Class for $\prod_{i=1}^n f_i(x)$ with $f_i(x)$ all binary	230
Couenne::exprCeil	Class ceiling, $\lceil f(x) \rceil$	232
Couenne::exprClone	Expression clone (points to another expression)	236
Couenne::exprConst	Constant-type operator	239
Couenne::exprCopy		243
Couenne::exprCos	Class cosine, $\cos f(x)$	252
Couenne::exprDiv	Class for divisions, $\frac{f(x)}{g(x)}$	256

Couenne::expression		
Expression base class		261
Couenne::exprExp		
Class for the exponential, $e^{f(x)}$		273
Couenne::exprFloor		
Class floor, $\lfloor f(x) \rfloor$		276
Couenne::exprGroup		
Class Group, with constant, linear and nonlinear terms: $a_0 + \sum_{i=1}^n a_i x_i$		280
Couenne::ExprHess		
Expression matrices		287
Couenne::exprIf		288
Couenne::exprInv		
Class inverse: $1/f(x)$		290
Couenne::exprVar		
Variable-type operator		293
Couenne::ExprJac		
Jacobian of the problem (computed through Couenne expression classes)		297
Couenne::exprLBCos		
Class to compute lower bound of a cosine based on the bounds of its arguments		298
Couenne::exprLBDiv		
Class to compute lower bound of a fraction based on the bounds of both numerator and denominator		300
Couenne::exprLBMul		
Class to compute lower bound of a product based on the bounds of both factors		303
Couenne::exprLBQuad		
Class to compute lower bound of a fraction based on the bounds of both numerator and denominator		305
Couenne::exprLBSin		
Class to compute lower bound of a sine based on the bounds on its arguments		307
Couenne::exprLog		
Class logarithm, $\log f(x)$		309
Couenne::exprLowerBound		
These are bound expression classes		313
Couenne::exprMax		
Class for maxima		317
Couenne::exprMin		
Class for minima		321
Couenne::exprMultiLin		
Another class for multiplications, $\prod_{i=1}^n f_i(x)$		325

Couenne::exprNorm		
Class for p-norms , $\ f(x)\ _p = (\sum_{i=1}^n f_i(x)^p)^{\frac{1}{p}}$		327
Couenne::exprOddPow		
Power of an expression (binary operator), $f(x)^k$ with k constant		329
Couenne::exprOp		
General n-ary operator-type expression: requires argument list		333
Couenne::exprOpp		
Class opposite, $-f(x)$		339
Couenne::exprPow		
Power of an expression (binary operator), $f(x)^k$ with k constant		343
Couenne::exprPWLinear		348
Couenne::exprQuad		
Class exprQuad, with constant, linear and quadratic terms		349
Couenne::exprSin		
Class for $\sin f(x)$		358
Couenne::exprStore		
Storage class for previously evaluated expressions		362
Couenne::exprSub		
Class for subtraction, $f(x) - g(x)$		365
Couenne::exprSum		
Class sum, $\sum_{i=1}^n f_i(x)$		368
Couenne::exprTrilinear		
Class for multiplications		373
Couenne::exprUBCos		
Class to compute lower bound of a cosine based on the bounds of its arguments		375
Couenne::exprUBDiv		
Class to compute upper bound of a fraction based on the bounds of both numerator and denominator		378
Couenne::exprUBMul		
Class to compute upper bound of a product based on the bounds of both factors		380
Couenne::exprUBQuad		
Class to compute upper bound of a fraction based on the bounds of both numerator and denominator		382
Couenne::exprUBSin		
Class to compute lower bound of a sine based on the bounds on its arguments		385
Couenne::exprUnary		
Expression class for unary functions (sin, log, etc.)		387
Couenne::exprUpperBound		
Upper bound		392

Couenne::exprVar	
Variable-type operator	396
Couenne::funtriplet	405
Couenne::GlobalCutOff	406
Couenne::InitHeuristic	
A heuristic that stores the initial solution of the NLP	407
Couenne::kpowertriplet	408
less_than_str	411
Couenne::LinMap	411
Couenne::MultiProdRel	
Identifies 5-ples of variables of the form	412
myclass	413
myclass0	414
Nauty	414
Couenne::CouenneInfo::NlpSolution	
Class for storing an Nlp Solution	416
Couenne::NlpSolveHeuristic	417
Node	419
Couenne::powertriplet	421
Couenne::PowRel	
Identifies 5-tuple of the form	423
Couenne::Qroot	
Class that stores result of previous calls to rootQ into a map for faster access	424
Couenne::quadElem	426
Couenne::QuadMap	427
Couenne::simpletriplet	428
Couenne::SmartAsl	430
Couenne::SumLogAuxRel	
Identifies 5-ples of variables of the form	431
Couenne::t_chg_bounds	
Status of lower/upper bound of a variable, to be checked/modified in bound tightening	432

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

/home/ted/COIN/trunk/Couenne/src/config_couenne_default.h	454
/home/ted/COIN/trunk/Couenne/src/config_default.h	455
/home/ted/COIN/trunk/Couenne/src/CouenneConfig.h	457
/home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneAggrProbing.hpp	434
/home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneBTPerfIndicator.hpp	435
/home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneFixPoint.hpp	436
/home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneInfeasCut.hpp	436
/home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneMultiVarProbe.hpp	437
/home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneSparseBndVec.hpp	438
/home/ted/COIN/trunk/Couenne/src/bound_tightening/twoImpliedBT/CouenneTwoImplied.hpp	438
/home/ted/COIN/trunk/Couenne/src/branch/CouenneBranchingObject.hpp	439
/home/ted/COIN/trunk/Couenne/src/branch/CouenneChooseStrong.hpp	441
/home/ted/COIN/trunk/Couenne/src/branch/CouenneChooseVariable.hpp	442
/home/ted/COIN/trunk/Couenne/src/branch/CouenneComplBranchingObject.hpp	444
/home/ted/COIN/trunk/Couenne/src/branch/CouenneComplObject.hpp	445
/home/ted/COIN/trunk/Couenne/src/branch/CouenneObject.hpp	445
/home/ted/COIN/trunk/Couenne/src/branch/CouenneOrbitBranchingObj.hpp	447
/home/ted/COIN/trunk/Couenne/src/branch/CouenneOrbitObj.hpp	448
/home/ted/COIN/trunk/Couenne/src/branch/CouenneProjections.hpp	448
/home/ted/COIN/trunk/Couenne/src/branch/CouenneSOSObject.hpp	449
/home/ted/COIN/trunk/Couenne/src/branch/CouenneThreeWayBranchObj.hpp	450
/home/ted/COIN/trunk/Couenne/src/branch/CouenneVarObject.hpp	452
/home/ted/COIN/trunk/Couenne/src/branch/CouenneVTOBJECT.hpp	453
/home/ted/COIN/trunk/Couenne/src/branch/Nauty.h	453
/home/ted/COIN/trunk/Couenne/src/convex/CouenneCutGenerator.hpp	456
/home/ted/COIN/trunk/Couenne/src/cut/crossconv/CouenneCrossConv.hpp	457
/home/ted/COIN/trunk/Couenne/src/cut/ellipcuts/CouenneEllipCuts.hpp	458
/home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneMatrix.hpp	458

/home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/ CouennePSDcon.hpp	459
/home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/ CouenneSdpCuts.hpp	460
/home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/ dsyevx_wrapper.hpp	461
/home/ted/COIN/trunk/Couenne/src/disjunctive/ CouenneDisjCuts.hpp	461
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneDomain.hpp	462
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprAux.hpp	463
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprBound.hpp	465
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprClone.hpp	466
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprConst.hpp	467
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprCopy.hpp	468
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExpression.hpp	469
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprVar.hpp	470
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprOp.hpp	470
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprStore.hpp	472
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprUnary.hpp	473
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneExprVar.hpp	474
/home/ted/COIN/trunk/Couenne/src/expression/ CouennePrecisions.hpp	475
/home/ted/COIN/trunk/Couenne/src/expression/ CouenneTypes.hpp	477
/home/ted/COIN/trunk/Couenne/src/expression/ CouExpr.hpp	479
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprAbs.hpp	485
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprBinProd.hpp	485
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprCeil.hpp	487
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprCos.hpp	488
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprDiv.hpp	488
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprEvenPow.hpp	490
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprExp.hpp	490
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprFloor.hpp	492
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprGroup.hpp	492
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprIf.hpp	494
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprInv.hpp	494

/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprLog.hpp	496
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprMax.hpp	497
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprMin.hpp	498
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprMul.hpp	498
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprMultiLin.hpp	499
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprNorm.hpp	500
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprOddPow.hpp	501
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprOpp.hpp	502
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprPow.hpp	503
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprPWLinear.hpp	505
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprQuad.hpp	505
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprSignPow.hpp	507
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprSin.hpp	507
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprSub.hpp	509
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprSum.hpp	510
/home/ted/COIN/trunk/Couenne/src/expression/operators/ CouenneExprTrilinear.hpp	511
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/ CouenneExprBCos.hpp	480
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/ CouenneExprBDiv.hpp	481
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/ CouenneExprBMul.hpp	482
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/ CouenneExprBQuad.hpp	483
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/ CouenneExprBSin.hpp	484
/home/ted/COIN/trunk/Couenne/src/expression/partial/ CouenneExprHess.hpp	513
/home/ted/COIN/trunk/Couenne/src/expression/partial/ CouenneExprJac.hpp	514
/home/ted/COIN/trunk/Couenne/src/heuristics/ BonInitHeuristic.hpp	514
/home/ted/COIN/trunk/Couenne/src/heuristics/ BonNlpHeuristic.hpp	515
/home/ted/COIN/trunk/Couenne/src/heuristics/ cons_rowcuts.h Constraint handler for rowcuts constraints enables separation of convexification cuts during SCIP solution procedure	516
/home/ted/COIN/trunk/Couenne/src/heuristics/ CouenneFeasPump.hpp	516
/home/ted/COIN/trunk/Couenne/src/heuristics/ CouenneFPpool.hpp	517
/home/ted/COIN/trunk/Couenne/src/heuristics/ CouennelterativeRounding.hpp	519

/home/ted/COIN/trunk/Couenne/src/interfaces/ BonCouenneInterface.hpp	520
/home/ted/COIN/trunk/Couenne/src/interfaces/ CouenneMINLPInterface.hpp	520
/home/ted/COIN/trunk/Couenne/src/interfaces/ CouenneTNLP.hpp	521
/home/ted/COIN/trunk/Couenne/src/interfaces/ CouenneUserInterface.hpp	523
/home/ted/COIN/trunk/Couenne/src/main/ BonCouenneInfo.hpp	524
/home/ted/COIN/trunk/Couenne/src/main/ BonCouenneSetup.hpp	524
/home/ted/COIN/trunk/Couenne/src/main/ CouenneBab.hpp	525
/home/ted/COIN/trunk/Couenne/src/main/ CouenneOSInterface.hpp	526
/home/ted/COIN/trunk/Couenne/src/problem/ CouenneGlobalCutOff.hpp	527
/home/ted/COIN/trunk/Couenne/src/problem/ CouenneJournalist.hpp	527
/home/ted/COIN/trunk/Couenne/src/problem/ CouenneProblem.hpp	528
/home/ted/COIN/trunk/Couenne/src/problem/ CouenneProblemElem.hpp	530
/home/ted/COIN/trunk/Couenne/src/problem/ CouenneRecordBestSol.hpp	532
/home/ted/COIN/trunk/Couenne/src/problem/ CouenneSolverInterface.hpp	532
/home/ted/COIN/trunk/Couenne/src/problem/depGraph/ CouenneDepGraph.hpp	533
/home/ted/COIN/trunk/Couenne/src/readnl/ CouenneAmplInterface.hpp	534
/home/ted/COIN/trunk/Couenne/src/standardize/ CouenneLQelems.hpp	535
/home/ted/COIN/trunk/Couenne/src/util/ CouenneFunTriplets.hpp	535
/home/ted/COIN/trunk/Couenne/src/util/ CouenneRootQ.hpp	536
/home/ted/COIN/trunk/Couenne/src/util/ CouenneSparseMatrix.hpp	537

6 Namespace Documentation

6.1 Bonmin Namespace Reference

6.2 Coin Namespace Reference

6.3 Couenne Namespace Reference

general include file for different compilers

Classes

- class [CouenneAggrProbing](#)
Cut Generator for aggressive BT; i.e., an aggressive probing.

- class [CouenneBTPerIndicator](#)
- class [CouenneFixPoint](#)
Cut Generator for FBBT fixpoint.
- class [CouenneMultiVarProbe](#)
- class [CouenneSparseBndVec](#)
- class [CouenneTwoImplied](#)
Cut Generator for implied bounds derived from pairs of linear (in)equalities.
- class [CouenneBranchingObject](#)
"Spatial" branching object.
- class [CouenneChooseStrong](#)
- class [CouenneChooseVariable](#)
Choose a variable for branching.
- class [CouenneComplBranchingObject](#)
"Spatial" branching object for complementarity constraints.
- class [CouenneComplObject](#)
OsiObject for complementarity constraints $x_1x_2 \geq, \leq, = 0$.
- class [CouenneObject](#)
OsiObject for auxiliary variables $\$w=f(x)$.
- class [CouenneOrbitBranchingObj](#)
"Spatial" branching object.
- class [CouenneSOSBranchingObject](#)
- class [CouenneSOSObject](#)
- class [CouenneThreeWayBranchObj](#)
Spatial, three-way branching object.
- class [CouenneVarObject](#)
OsiObject for variables in a MINLP.
- class [CouenneVTOBJECT](#)
OsiObject for violation transfer on variables in a MINLP.
- class [CouenneCutGenerator](#)
Cut Generator for linear convexifications.
- class [AuxRelation](#)
Base class definition for relations between auxiliaries.
- class [SumLogAuxRel](#)
Identifies 5-ples of variables of the form.
- class [MultiProdRel](#)
Identifies 5-ples of variables of the form.
- class [BiProdDivRel](#)
Identifies 5-tuple of the form.
- class [PowRel](#)
Identifies 5-tuple of the form.
- class [CouenneCrossConv](#)
Cut Generator that uses relationships between auxiliaries.
- class [CouenneScalar](#)
- class [CouenneSparseVector](#)
- class [CouenneExprMatrix](#)
- class [CouennePSDcon](#)
Class to represent positive semidefinite constraints ////////////////.
- class [CouenneSdpCuts](#)

These are cuts of the form.

- class [CouenneDisjCuts](#)

Cut Generator for linear convexifications.

- class [DomainPoint](#)

Define a point in the solution space and the bounds around it.

- class [Domain](#)

Define a dynamic point+bounds, with a way to save and restore previous points+bounds through a LIFO structure.

- class [exprAux](#)

Auxiliary variable.

- struct [compExpr](#)

Structure for comparing expressions.

- class [exprLowerBound](#)

These are bound expression classes.

- class [exprUpperBound](#)

upper bound

- class [exprClone](#)

expression clone (points to another expression)

- class [exprConst](#)

constant-type operator

- class [exprCopy](#)

- class [expression](#)

Expression base class.

- class [exprVar](#)

variable-type operator.

- class [exprOp](#)

general n-ary operator-type expression: requires argument list.

- class [exprStore](#)

storage class for previously evaluated expressions

- class [exprUnary](#)

expression class for unary functions (sin, log, etc.)

- class [exprVar](#)

variable-type operator

- class [t_chg_bounds](#)

status of lower/upper bound of a variable, to be checked/modified in bound tightening

- class [CouExpr](#)

- class [exprLBCos](#)

class to compute lower bound of a cosine based on the bounds of its arguments

- class [exprUBCos](#)

class to compute lower bound of a cosine based on the bounds of its arguments

- class [exprLBDiv](#)

class to compute lower bound of a fraction based on the bounds of both numerator and denominator

- class [exprUBDiv](#)

class to compute upper bound of a fraction based on the bounds of both numerator and denominator

- class [exprLBMul](#)

class to compute lower bound of a product based on the bounds of both factors

- class [exprUBMul](#)

class to compute upper bound of a product based on the bounds of both factors

- class `exprLBQuad`
`class to compute lower bound of a fraction based on the bounds of both numerator and denominator`
- class `exprUBQuad`
`class to compute upper bound of a fraction based on the bounds of both numerator and denominator`
- class `exprLBSin`
`class to compute lower bound of a sine based on the bounds on its arguments`
- class `exprUBSin`
`class to compute lower bound of a sine based on the bounds on its arguments`
- class `exprAbs`
`class for $|f(x)|$`
- class `exprBinProd`
`class for $\prod_{i=1}^n f_i(x)$ with $f_i(x)$ all binary`
- class `exprCeil`
`class ceiling, $\lceil f(x) \rceil$`
- class `exprCos`
`class cosine, $\cos f(x)$`
- class `exprDiv`
`class for divisions, $\frac{f(x)}{g(x)}$`
- class `exprExp`
`class for the exponential, $e^{f(x)}$`
- class `exprFloor`
`class floor, $\lfloor f(x) \rfloor$`
- class `exprGroup`
`class Group, with constant, linear and nonlinear terms: $a_0 + \sum_{i=1}^n a_i x_i$`
- class `exprIf`
- class `exprInv`
`class inverse: $1/f(x)$`
- class `exprLog`
`class logarithm, $\log f(x)$`
- class `exprMax`
`class for maxima`
- class `exprMin`
`class for minima`
- class `exprMultiLin`
`another class for multiplications, $\prod_{i=1}^n f_i(x)$`
- class `exprNorm`
`Class for p -norms, $\|f(x)\|_p = \left(\sum_{i=1}^n f_i(x)^p\right)^{\frac{1}{p}}$.`
- class `exprOddPow`
`Power of an expression (binary operator), $f(x)^k$ with k constant.`
- class `exprOpp`
`class opposite, $-f(x)$`
- class `exprPow`
`Power of an expression (binary operator), $f(x)^k$ with k constant.`
- class `exprPWLinear`
- class `exprQuad`
`class exprQuad, with constant, linear and quadratic terms`
- class `exprSin`

- class `exprSub`

$$\text{class for subtraction, } f(x) - g(x)$$
- class `exprSum`

$$\text{class sum, } \sum_{i=1}^n f_i(x)$$
- class `exprTrilinear`

$$\text{class for multiplications}$$
- class `ExprHess`

$$\text{expression matrices.}$$
- class `ExprJac`

$$\text{Jacobian of the problem (computed through Couenne expression classes).}$$
- class `InitHeuristic`

$$\text{A heuristic that stores the initial solution of the NLP.}$$
- class `NlpSolveHeuristic`
- class `CouenneFeasPump`

$$\text{An implementation of the Feasibility pump that uses linearization and Ipopt to find the two sequences of points.}$$
- class `CouenneFPSolution`

$$\text{Class containing a solution with infeasibility evaluation.}$$
- class `compareSol`

$$\text{class for comparing solutions (used in tabu list)}$$
- class `CouenneFPpool`

$$\text{Pool of solutions.}$$
- class `CouenneIterativeRounding`

$$\text{An iterative rounding heuristic, tailored for nonconvex MINLPs.}$$
- class `CouenneInterface`
- class `CouenneMINLPInterface`

$$\text{This is class provides an Osi interface for a Mixed Integer Linear Program expressed as a TMINLP (so that we can use it for example as the continuous solver in Cbc).}$$
- class `CouenneTNLP`

$$\text{Class for handling NLPs using CouenneProblem.}$$
- class `CouenneUserInterface`
- class `CouenneInfo`

$$\text{Bonmin class for passing info between components of branch-and-cuts.}$$
- class `SmartAsl`
- class `CouenneSetup`
- class `CouenneBab`
- class `CouenneOSInterface`
- class `GlobalCutOff`
- class `CouenneProblem`

$$\text{Class for MINLP problems with symbolic information.}$$
- class `CouenneConstraint`

$$\text{Class to represent nonlinear constraints.}$$
- class `CouenneObjective`

$$\text{Objective function.}$$
- class `CouenneRecordBestSol`
- class `CouenneSolverInterface`

$$\text{Solver interface class with a pointer to a Couenne cut generator.}$$
- struct `compNode`

- class **DepNode**
vertex of a dependence graph.
- class **DepGraph**
Dependence graph.
- class **CouenneAmplInterface**
- class **quadElem**
- class **LinMap**
- class **QuadMap**
- class **funtriplet**
- class **simpletriplet**
- class **powertriplet**
- class **kpowertriplet**
- class **Qroot**
- class that stores result of previous calls to rootQ into a map for faster access*
- class **CouenneSparseMatrix**
Class for sparse Matrixs (used in modifying distances in FP)

Typedefs

- typedef `Ipopt::SmartPtr<Ipopt::Journalist> JnlstPtr`
- typedef `Ipopt::SmartPtr<const Ipopt::Journalist> ConstJnlstPtr`
- typedef double **CouNumber**
main number type in Couenne
- typedef `CouNumber(* unary_function)(CouNumber)`
unary function, used in all exprUnary

Enumerations

- enum {
TWO_LEFT, TWO_RIGHT, TWO_RAND, THREE_LEFT,
THREE_CENTER, THREE_RIGHT, THREE_RAND, BRANCH_NONE }
Define what kind of branching (two- or three-way) and where to start from: left, (center,) or right.
- enum { **COUENNE_INFEASIBLE, COUENNE_TIGHTENED, COUENNE_FEASIBLE** }
- enum **nodeType** {
CONST = 0, VAR, UNARY, N_ARY,
COPY, AUX, EMPTY }
type of a node in an expression tree
- enum **linearity_type** {
ZERO = 0, CONSTANT, LINEAR, QUADRATIC,
NONLINEAR }
linearity of an expression, as returned by the method Linearity()
- enum **pos** { **PRE = 0, POST, INSIDE, NONE** }
position where the operator should be printed when printing the expression
- enum **con_sign** { **COUENNE_EQ, COUENNE_LE, COUENNE_GE, COUENNE RNG** }
sign of constraint
- enum **conv_type** { **CURRENT_ONLY, UNIFORM_GRID, AROUND_CURPOINT** }

- position and number of convexification cuts added for a lower convex (upper concave) envelope*
- enum `expr_type` {
 `COU_EXPRESSION, COU_EXPRCONST, COU_EXPRVAR, COU_EXPRLBOUND,`
`COU_EXPRUBOUND, COU_EXPROP, COU_EXPRSUB, COU_EXPRSUM,`
`COU_EXPRGROUP, COU_EXPRQUAD, COU_EXPRMIN, COU_EXPRMUL,`
`COU_EXPRTRILINEAR, COU_EXPRPOW, COU_EXPRSIGNPOW, COU_EXPRMAX,`
`COU_EXPRDIV, COU_EXPRUNARY, COU_EXPRCOS, COU_EXPRABS,`
`COU_EXPREXP, COU_EXPRINV, COU_EXPRLOG, COU_EXPROPP,`
`COU_EXPRSIN, COU_EXPRFLOOR, COU_EXPRCEIL, MAX_COU_EXPR_CODE }`

code returned by the method expression::code()
 - enum `convexity` {
 `UNSET, NONCONVEX, CONVEX, CONCAVE,`
`AFFINE, CONV_LINEAR, CONV_CONSTANT, CONV_ZERO }`

convexity type of an expression
 - enum `monotonicity` {
 `MON_UNSET, NONMONOTONE, NDECREAS, NINCREAS,`
`INCLIN, DECLIN, MON_CONST, MON_ZERO }`

monotonicity type of an expression
 - enum `dig_type` { `ORIG_ONLY, STOP_AT_AUX, TAG_AND_RECURSIVE, COUNT` }

type of digging when filling the dependence list
 - enum `cou_trig` { `COU_SINE, COU_COSINE` }

specify which trigonometric function is dealt with in trigEnvelope
 - enum `what_to_compare` {
 `SUM_NINF = 0, SUM_INF, OBJVAL, ALL_VARS,`
`INTEGER_VARS` }

what term to compare: the sum of infeasibilities, the sum of numbers of infeasible terms, or the objective function
 - enum `Solver` { `Elpopt = 0, EFilterSQP, EAII` }

Solvers for solving nonlinear programs.
 - enum `TrilinDecompType` { `rAI, treeDecomp, bi_tri, tri_bi` }

Functions

- `CouNumber minMaxDelta (funtriplet *ft, CouNumber lb, CouNumber ub)`
- `CouNumber maxHeight (funtriplet *ft, CouNumber lb, CouNumber ub)`
- `CouNumber project (CouNumber a, CouNumber b, CouNumber c, CouNumber x0, CouNumber y0, CouNumber lb, CouNumber ub, int sign, CouNumber *xp=NULL, CouNumber *yp=NULL)`

Compute projection of point (x_0, y_0) on the segment defined by line $ax + by + c \leq 0$ (sign provided by parameter sign) and bounds $[lb, ub]$ on x.
- `CouNumber projectSeg (CouNumber x0, CouNumber y0, CouNumber x1, CouNumber y1, CouNumber x2, CouNumber y2, int sign, CouNumber *xp=NULL, CouNumber *yp=NULL)`

Compute projection of point (x_0, y_0) on the segment defined by two points $(x_1, y_1), (x_2, y_2)$ – sign provided by parameter sign.
- void `sparse2dense (int ncols, t_chg_bounds *chg_bds, int *&changed, int &nchanged)`

translate sparse to dense vector (should be replaced)
- bool `operator< (const CouenneScalar &first, const CouenneScalar &second)`
- void `CoinInvN (register const double *orig, register int n, register double *inverted)`

invert all contents
- void `CoinCopyDisp (register const int *src, register int num, register int *dst, register int displacement)`

a CoinCopyN with a $+=$ on each element
- void `draw_cuts (OsiCuts &, const CouenneCutGenerator *, int, expression *, expression *)`

- `allow to draw function within intervals and cuts introduced`
- `bool updateBound (register int sign, register CouNumber *dst, register CouNumber src)`
updates maximum violation.
- `int compareExpr (const void *e0, const void *e1)`
independent comparison
- `bool isInteger (CouNumber x)`
is this number integer?
- `expression * getOriginal (expression *e)`
get original expression (can't make it an expression method as I need a non-const, what "this" would return)
- `CouNumber zero_fun (CouNumber x)`
zero function (used by default by exprUnary)
- `CouExpr operator+ (CouExpr &e1, CouExpr &e2)`
- `CouExpr & operator/ (CouExpr &e1, CouExpr &e2)`
- `CouExpr & operator% (CouExpr &e1, CouExpr &e2)`
- `CouExpr & operator- (CouExpr &e1, CouExpr &e2)`
- `CouExpr & operator* (CouExpr &e1, CouExpr &e2)`
- `CouExpr & operator^ (CouExpr &e1, CouExpr &e2)`
- `CouExpr & sin (CouExpr &e)`
- `CouExpr & cos (CouExpr &e)`
- `CouExpr & log (CouExpr &e)`
- `CouExpr & exp (CouExpr &e)`
- `CouExpr & operator+ (CouNumber &e1, CouExpr &e2)`
- `CouExpr & operator/ (CouNumber &e1, CouExpr &e2)`
- `CouExpr & operator% (CouNumber &e1, CouExpr &e2)`
- `CouExpr & operator- (CouNumber &e1, CouExpr &e2)`
- `CouExpr & operator* (CouNumber &e1, CouExpr &e2)`
- `CouExpr & operator^ (CouNumber &e1, CouExpr &e2)`
- `CouExpr & sin (CouNumber &e)`
- `CouExpr & cos (CouNumber &e)`
- `CouExpr & log (CouNumber &e)`
- `CouExpr & exp (CouNumber &e)`
- `CouExpr & operator+ (CouExpr &e1, CouNumber &e2)`
- `CouExpr & operator/ (CouExpr &e1, CouNumber &e2)`
- `CouExpr & operator% (CouExpr &e1, CouNumber &e2)`
- `CouExpr & operator- (CouExpr &e1, CouNumber &e2)`
- `CouExpr & operator* (CouExpr &e1, CouNumber &e2)`
- `CouExpr & operator^ (CouExpr &e1, CouNumber &e2)`
- `static CouNumber safeDiv (register CouNumber a, register CouNumber b, int sign)`
division that avoids NaN's and considers a sign when returning infinity
- `CouNumber safeProd (register CouNumber a, register CouNumber b)`
product that avoids NaN's
- `CouNumber trigNewton (CouNumber, CouNumber, CouNumber)`
common convexification method used by both cos and sin
- `bool is_boundbox_regular (register CouNumber b1, register CouNumber b2)`
check if bounding box is suitable for a multiplication/division convexification constraint
- `CouNumber inv (register CouNumber arg)`
the operator itself
- `CouNumber opplnSqr (register CouNumber x)`
derivative of inv (x)

- **CouNumber inv_dblprime** (register **CouNumber** x)

inv_dblprime, second derivative of inv (x)
- **CouNumber opp** (register **CouNumber** arg)

operator opp: returns the opposite of a number
- **CouNumber safe_pow** (**CouNumber** base, **CouNumber** exponent, bool signpower=false)

compute power and check for integer-and-odd inverse exponent
- **void addPowEnvelope** (const **CouenneCutGenerator** *, OsiCuts &, int, int, **CouNumber**, **CouNumber**, **CouNumber**, **CouNumber**, **CouNumber**, int, bool=false)

add upper/lower envelope to power in convex/concave areas
- **CouNumber powNewton** (**CouNumber**, **CouNumber**, unary_function, unary_function, unary_function)

find proper tangent point to add deepest tangent cut
- **CouNumber powNewton** (**CouNumber**, **CouNumber**, funtriplet *)

find proper tangent point to add deepest tangent cut
- **CouNumber modulo** (register **CouNumber** a, register **CouNumber** b)

normalize angle within [0,b] (typically, pi or 2pi)
- **CouNumber trigSelBranch** (const **CouenneObject** *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way, enum **cou_trig** type)

generalized procedure for both sine and cosine
- **bool trigImpliedBound** (enum **cou_trig**, int, int, **CouNumber** *, **CouNumber** *, t_chg_bounds *)

generalized implied bound procedure for sine/cosine
- **bool operator<** (const **CouenneFPSolution** &one, const **CouenneFPSolution** &two)

compare, base version
- **const Ipopt::EJournalCategory J_BRANCHING** (Ipopt::J_USER1)
- **const Ipopt::EJournalCategory J_BOUNDTIGHTENING** (Ipopt::J_USER2)
- **const Ipopt::EJournalCategory J_CONVEXIFYING** (Ipopt::J_USER3)
- **const Ipopt::EJournalCategory J_PROBLEM** (Ipopt::J_USER4)
- **const Ipopt::EJournalCategory J_NLPHEURISTIC** (Ipopt::J_USER5)
- **const Ipopt::EJournalCategory J_DISJCUTS** (Ipopt::J_USER6)
- **const Ipopt::EJournalCategory J_REFORMULATE** (Ipopt::J_USER7)
- **const Ipopt::EJournalCategory J_COUENNE** (Ipopt::J_USER8)
- **CouNumber rootQ** (int k)

Find roots of polynomial \$Q^k(x) = \sum_{i=1}^{2k} ix^{i-1}.

Variables

- **const CouNumber default_alpha** = 0.25
- **const CouNumber default_clamp** = 0.2
- **const CouNumber max_pseudocost** = 1000.
- **const double large_bound** = 1e9

if |branching point| > this, change it
- **const CouNumber closeToBounds** = .05
- **const double Couenne_large_bound** = 9.999e12

used to declare LP unbounded
- **const double maxNlpInf_0** = 1e-5

A heuristic to call an NlpSolver if all CouenneObjects are close to be satisfied (for other integer objects, rounding is performed, if SOS's are not satisfied it does not run).
- **static enum Couenne::what_to_compare comparedTerm_**
- **const CouNumber feas_tolerance_default** = 1e-5

6.3.1 Detailed Description

general include file for different compilers

6.3.2 Typedef Documentation

6.3.2.1 `typedef Ipopt::SmartPtr<Ipopt::Journalist> Couenne::JnlstPtr`

Definition at line 34 of file CouenneExprVar.hpp.

6.3.2.2 `typedef Ipopt::SmartPtr<const Ipopt::Journalist> Couenne::ConstJnlstPtr`

Definition at line 35 of file CouenneExprVar.hpp.

6.3.2.3 `typedef double Couenne::CouNumber`

main number type in [Couenne](#)

Definition at line 100 of file CouenneTypes.hpp.

6.3.2.4 `typedef CouNumber(*) Couenne::unary_function)(CouNumber)`

unary function, used in all [exprUnary](#)

Definition at line 103 of file CouenneTypes.hpp.

6.3.3 Enumeration Type Documentation

6.3.3.1 anonymous enum

Define what kind of branching (two- or three-way) and where to start from: left, (center,) or right.

The last is to help diversify branching through randomization, which may help when the same variable is branched upon in several points of the BB tree.

Enumerator:

TWO_LEFT
TWO_RIGHT
TWO_RAND
THREE_LEFT
THREE_CENTER
THREE_RIGHT
THREE_RAND
BRANCH_NONE

Definition at line 40 of file CouenneObject.hpp.

6.3.3.2 anonymous enum

Enumerator:

COUENNE_INFEASIBLE
COUENNE_TIGHTENED

COUENNE_FEASIBLE

Definition at line 30 of file CouenneDisjCuts.hpp.

6.3.3.3 enum Couenne::nodeType

type of a node in an expression tree

Enumerator:

CONST
VAR
UNARY
N_ARY
COPY
AUX
EMPTY

Definition at line 20 of file CouenneTypes.hpp.

6.3.3.4 enum Couenne::linearity_type

linearity of an expression, as returned by the method Linearity()

Enumerator:

ZERO
CONSTANT
LINEAR
QUADRATIC
NONLINEAR

Definition at line 23 of file CouenneTypes.hpp.

6.3.3.5 enum Couenne::pos

position where the operator should be printed when printing the expression

For instance, it is INSIDE for [exprSum](#), [exprMul](#), [exprDiv](#), while it is PRE for [exprLog](#), [exprSin](#), [exprExp](#)...

Enumerator:

PRE
POST
INSIDE
NONE

Definition at line 30 of file CouenneTypes.hpp.

6.3.3.6 enum Couenne::con_sign

sign of constraint

Enumerator:

COUENNE_EQ
COUENNE_LE
COUENNE_GE
COUENNE_RNG

Definition at line 33 of file CouenneTypes.hpp.

6.3.3.7 enum Couenne::conv_type

position and number of convexification cuts added for a lower convex (upper concave) envelope

Enumerator:

CURRENT_ONLY
UNIFORM_GRID
AROUND_CURPOINT

Definition at line 37 of file CouenneTypes.hpp.

6.3.3.8 enum Couenne::expr_type

code returned by the method [expression::code\(\)](#)

Enumerator:

COU_EXPRESSION
COU_EXPRCONST
COU_EXPRVAR
COU_EXPRLBOUND
COU_EXPRUBOUND
COU_EXPROP
COU_EXPRSUB
COU_EXPRSUM
COU_EXPRGROUP
COU_EXPRQUAD
COU_EXPRMIN
COU_EXPRMUL
COU_EXPRTRILINEAR
COU_EXPRPOW
COU_EXPRSIGNPOW
COU_EXPRMAX
COU_EXPRDIV
COU_EXPRUNARY

COU_EXPRCOS
COU_EXPRABS
COU_EXPREXP
COU_EXPRINV
COU_EXPRLOG
COU_EXPROPP
COU_EXPRSIN
COU_EXPRFLOOR
COU_EXPRCEIL
MAX_COU_EXPR_CODE

Definition at line 40 of file CouenneTypes.hpp.

6.3.3.9 enum Couenne::convexity

convexity type of an expression

Enumerator:

UNSET
NONCONVEX
CONVEX
CONCAVE
AFFINE
CONV_LINEAR
CONV_CONSTANT
CONV_ZERO

Definition at line 56 of file CouenneTypes.hpp.

6.3.3.10 enum Couenne::monotonicity

monotonicity type of an expression

Enumerator:

MON_UNSET
NONMONOTONE
NDECREAS
NINCREAS
INCLIN
DECLIN
MON_CONST
MON_ZERO

Definition at line 59 of file CouenneTypes.hpp.

6.3.3.11 enum Couenne::dig_type

type of digging when filling the dependence list

Enumerator:

ORIG_ONLY
STOP_AT_AUX
TAG_AND_RECURSIVE
COUNT

Definition at line 62 of file CouenneTypes.hpp.

6.3.3.12 enum Couenne::cou_trig

specify which trigonometric function is dealt with in trigEnvelope

Enumerator:

COU_SINE
COU_COSINE

Definition at line 23 of file CouenneExprSin.hpp.

6.3.3.13 enum Couenne::what_to_compare

what term to compare: the sum of infeasibilities, the sum of numbers of infeasible terms, or the objective function

Enumerator:

SUM_NINF
SUM_INF
OBJVAL
ALL_VARS
INTEGER_VARS

Definition at line 29 of file CouenneFPpool.hpp.

6.3.3.14 enum Couenne::Solver

Solvers for solving nonlinear programs.

Enumerator:

Elpopt *Ipopt* interior point algorithm
EFilterSQP *filterSQP* Sequential Quadratic Programming algorithm
EAll Use all solvers.

Definition at line 47 of file CouenneMINLPIterface.hpp.

6.3.3.15 enum Couenne::TrilinDecompType

Enumerator:

- rAI*
- treeDecomp*
- bi_tri*
- tri_bi*

Definition at line 136 of file CouenneProblem.hpp.

6.3.4 Function Documentation

6.3.4.1 CouNumber Couenne::minMaxDelta (funtriplet * *ft*, CouNumber *lb*, CouNumber *ub*)

6.3.4.2 CouNumber Couenne::maxHeight (funtriplet * *ft*, CouNumber *lb*, CouNumber *ub*)

6.3.4.3 CouNumber Couenne::project (CouNumber *a*, CouNumber *b*, CouNumber *c*, CouNumber *x0*, CouNumber *y0*, CouNumber *lb*, CouNumber *ub*, int *sign*, CouNumber * *xp* = NULL, CouNumber * *yp* = NULL)

Compute projection of point (*x0*, *y0*) on the segment defined by line $ax + by + c \leq 0$ (sign provided by parameter *sign*) and bounds [*lb*, *ub*] on *x*.

Return distance from segment, 0 if satisfied

6.3.4.4 CouNumber Couenne::projectSeg (CouNumber *x0*, CouNumber *y0*, CouNumber *x1*, CouNumber *y1*, CouNumber *x2*, CouNumber *y2*, int *sign*, CouNumber * *xp* = NULL, CouNumber * *yp* = NULL)

Compute projection of point (*x0*, *y0*) on the segment defined by two points (*x1,y1*), (*x2, y2*) – sign provided by parameter *sign*.

Return distance from segment, 0 if on it.

6.3.4.5 void Couenne::sparse2dense (int *ncols*, t_chg_bounds * *chg_bds*, int * & *changed*, int & *nchanged*)

translate sparse to dense vector (should be replaced)

6.3.4.6 bool Couenne::operator< (const CouenneScalar & *first*, const CouenneScalar & *second*) [inline]

Definition at line 62 of file CouenneMatrix.hpp.

6.3.4.7 void Couenne::CoinInvN (register const double * *orig*, register int *n*, register double * *inverted*) [inline]

invert all contents

Definition at line 194 of file CouenneDisjCuts.hpp.

6.3.4.8 void Couenne::CoinCopyDisp (register const int * *src*, register int *num*, register int * *dst*, register int *displacement*) [inline]

a CoinCopyN with a $+=$ on each element

Definition at line 203 of file CouenneDisjCuts.hpp.

6.3.4.9 void Couenne::draw_cuts (OsiCuts &, const CouenneCutGenerator * , int , expression * , expression *)

allow to draw function within intervals and cuts introduced

6.3.4.10 `bool Couenne::updateBound (register int sign, register CouNumber * dst, register CouNumber src) [inline]`

updates maximum violation.

Used with all impliedBound. Returns true if a bound has been modified, false otherwise

Definition at line 279 of file CouenneExpression.hpp.

6.3.4.11 `int Couenne::compareExpr (const void * e0, const void * e1) [inline]`

independent comparison

Definition at line 304 of file CouenneExpression.hpp.

6.3.4.12 `bool Couenne::isInteger (CouNumber x) [inline]`

is this number integer?

Definition at line 309 of file CouenneExpression.hpp.

6.3.4.13 `expression* Couenne::getOriginal (expression * e) [inline]`

get original expression (can't make it an expression method as I need a non-const, what "this" would return)

Definition at line 315 of file CouenneExpression.hpp.

6.3.4.14 `CouNumber Couenne::zero_fun (CouNumber x) [inline]`

zero function (used by default by [exprUnary](#))

Definition at line 22 of file CouenneExprUnary.hpp.

6.3.4.15 `CouExpr Couenne::operator+ (CouExpr & e1, CouExpr & e2)`

6.3.4.16 `CouExpr& Couenne::operator/ (CouExpr & e1, CouExpr & e2)`

6.3.4.17 `CouExpr& Couenne::operator% (CouExpr & e1, CouExpr & e2)`

6.3.4.18 `CouExpr& Couenne::operator- (CouExpr & e1, CouExpr & e2)`

6.3.4.19 `CouExpr& Couenne::operator* (CouExpr & e1, CouExpr & e2)`

6.3.4.20 `CouExpr& Couenne::operator^ (CouExpr & e1, CouExpr & e2)`

6.3.4.21 `CouExpr& Couenne::sin (CouExpr & e)`

6.3.4.22 `CouExpr& Couenne::cos (CouExpr & e)`

6.3.4.23 `CouExpr& Couenne::log (CouExpr & e)`

6.3.4.24 `CouExpr& Couenne::exp (CouExpr & e)`

6.3.4.25 `CouExpr& Couenne::operator+ (CouNumber & e1, CouExpr & e2)`

6.3.4.26 `CouExpr& Couenne::operator/ (CouNumber & e1, CouExpr & e2)`

6.3.4.27 `CouExpr& Couenne::operator% (CouNumber & e1, CouExpr & e2)`

6.3.4.28 `CouExpr& Couenne::operator- (CouNumber & e1, CouExpr & e2)`

- 6.3.4.29 **CouExpr& Couenne::operator*** (**CouNumber & e1, CouExpr & e2**)
- 6.3.4.30 **CouExpr& Couenne::operator^** (**CouNumber & e1, CouExpr & e2**)
- 6.3.4.31 **CouExpr& Couenne::sin** (**CouNumber & e**)
- 6.3.4.32 **CouExpr& Couenne::cos** (**CouNumber & e**)
- 6.3.4.33 **CouExpr& Couenne::log** (**CouNumber & e**)
- 6.3.4.34 **CouExpr& Couenne::exp** (**CouNumber & e**)
- 6.3.4.35 **CouExpr& Couenne::operator+** (**CouExpr & e1, CouNumber & e2**)
- 6.3.4.36 **CouExpr& Couenne::operator/** (**CouExpr & e1, CouNumber & e2**)
- 6.3.4.37 **CouExpr& Couenne::operator%** (**CouExpr & e1, CouNumber & e2**)
- 6.3.4.38 **CouExpr& Couenne::operator-** (**CouExpr & e1, CouNumber & e2**)
- 6.3.4.39 **CouExpr& Couenne::operator*** (**CouExpr & e1, CouNumber & e2**)
- 6.3.4.40 **CouExpr& Couenne::operator^** (**CouExpr & e1, CouNumber & e2**)
- 6.3.4.41 **static CouNumber Couenne::safeDiv** (**register CouNumber a, register CouNumber b, int sign**) [inline],
[static]

division that avoids NaN's and considers a sign when returning infinity

Definition at line 19 of file CouenneExprBDiv.hpp.

- 6.3.4.42 **CouNumber Couenne::safeProd** (**register CouNumber a, register CouNumber b**) [inline]

product that avoids NaN's

Definition at line 25 of file CouenneExprBMul.hpp.

- 6.3.4.43 **CouNumber Couenne::trigNewton** (**CouNumber , CouNumber , CouNumber**)

common convexification method used by both cos and sin

- 6.3.4.44 **bool Couenne::is_boundbox_regular** (**register CouNumber b1, register CouNumber b2**) [inline]

check if bounding box is suitable for a multiplication/division convexification constraint

Definition at line 124 of file CouenneExprDiv.hpp.

- 6.3.4.45 **CouNumber Couenne::inv** (**register CouNumber arg**) [inline]

the operator itself

Definition at line 19 of file CouenneExprInv.hpp.

- 6.3.4.46 **CouNumber Couenne::oppInvSqr** (**register CouNumber x**) [inline]

derivative of inv (x)

Definition at line 24 of file CouenneExprInv.hpp.

6.3.4.47 CouNumber Couenne::inv_dblprime (register CouNumber *x*) [inline]

inv_dblprime, second derivative of inv (*x*)

Definition at line 29 of file CouenneExprInv.hpp.

6.3.4.48 CouNumber Couenne::opp (register CouNumber *arg*) [inline]

operator opp: returns the opposite of a number

Definition at line 21 of file CouenneExprOpp.hpp.

6.3.4.49 CouNumber Couenne::safe_pow (CouNumber *base*, CouNumber *exponent*, bool *signpower*=false) [inline]

compute power and check for integer-and-odd inverse exponent

Definition at line 129 of file CouenneExprPow.hpp.

6.3.4.50 void Couenne::addPowEnvelope (const CouenneCutGenerator *, OsiCuts &, int, int, CouNumber, CouNumber, CouNumber, CouNumber, CouNumber, int, bool =false)

add upper/lower envelope to power in convex/concave areas

6.3.4.51 CouNumber Couenne::powNewton (CouNumber, CouNumber, unary_function, unary_function, unary_function)

find proper tangent point to add deepest tangent cut

6.3.4.52 CouNumber Couenne::powNewton (CouNumber, CouNumber, funtriplet *)

find proper tangent point to add deepest tangent cut

6.3.4.53 CouNumber Couenne::modulo (register CouNumber *a*, register CouNumber *b*) [inline]

normalize angle within [0,b] (typically, pi or 2pi)

Definition at line 27 of file CouenneExprSin.hpp.

6.3.4.54 CouNumber Couenne::trigSelBranch (const CouenneObject * *obj*, const OsiBranchingInformation * *info*, expression *& *var*, double *& *brpts*, double *& *brDist*, int & *way*, enum cou_trig type)

generalized procedure for both sine and cosine

6.3.4.55 bool Couenne::trigImpliedBound (enum cou_trig, int, int, CouNumber *, CouNumber *, t_chg_bounds *)

generalized implied bound procedure for sine/cosine

6.3.4.56 bool Couenne::operator< (const CouenneFPsolution & *one*, const CouenneFPsolution & *two*) [inline]

compare, base version

Definition at line 76 of file CouenneFPpool.hpp.

6.3.4.57 const Ipopt::EJournalCategory Couenne::J_BRANCHING (Ipopt::J_USER1)**6.3.4.58 const Ipopt::EJournalCategory Couenne::J_BOUNDTIGHTENING (Ipopt::J_USER2)****6.3.4.59 const Ipopt::EJournalCategory Couenne::J_CONVEXIFYING (Ipopt::J_USER3)****6.3.4.60 const Ipopt::EJournalCategory Couenne::J_PROBLEM (Ipopt::J_USER4)**

6.3.4.61 const Ipopt::EJournalCategory Couenne::J_NLPHEURISTIC (Ipopt::J_USER5)

6.3.4.62 const Ipopt::EJournalCategory Couenne::J_DISJCUTS (Ipopt::J_USER6)

6.3.4.63 const Ipopt::EJournalCategory Couenne::J_REFORMULATE (Ipopt::J_USER7)

6.3.4.64 const Ipopt::EJournalCategory Couenne::J_COUENNE (Ipopt::J_USER8)

6.3.4.65 CouNumber Couenne::rootQ (int k)

Find roots of polynomial $Q^k k(x) = \sum_{i=1}^{2k} ix^{i-1}$.

Used in convexification of powers with odd exponent

6.3.5 Variable Documentation

6.3.5.1 const CouNumber Couenne::default_alpha = 0.25

Definition at line 23 of file CouenneObject.hpp.

6.3.5.2 const CouNumber Couenne::default_clamp = 0.2

Definition at line 24 of file CouenneObject.hpp.

6.3.5.3 const CouNumber Couenne::max_pseudocost = 1000.

Definition at line 25 of file CouenneObject.hpp.

6.3.5.4 const double Couenne::large_bound = 1e9

if |branching point| > this, change it

Definition at line 28 of file CouenneObject.hpp.

6.3.5.5 const CouNumber Couenne::closeToBounds = .05

Definition at line 33 of file CouenneObject.hpp.

6.3.5.6 const double Couenne::Couenne_large_bound = 9.999e12

used to declare LP unbounded

Definition at line 50 of file CouennePrecisions.hpp.

6.3.5.7 const double Couenne::maxNlpInf_0 = 1e-5

A heuristic to call an NlpSolver if all CouenneObjects are close to be satisfied (for other integer objects, rounding is performed, if SOS's are not satisfied it does not run).

Definition at line 26 of file BonNlpHeuristic.hpp.

6.3.5.8 enum Couenne::what_to_compare Couenne::comparedTerm_ [static]

6.3.5.9 const CouNumber Couenne::feas_tolerance_default = 1e-5

Definition at line 159 of file CouenneProblem.hpp.

6.4 Ipopt Namespace Reference

6.5 Osi Namespace Reference

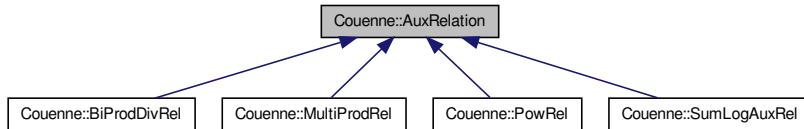
7 Class Documentation

7.1 Couenne::AuxRelation Class Reference

Base class definition for relations between auxiliaries.

```
#include <CouenneCrossConv.hpp>
```

Inheritance diagram for Couenne::AuxRelation:



Public Member Functions

- virtual int [findRelations \(\)=0](#)
- virtual void [generateCuts \(const OsiSolverInterface &, OsiCuts &, const CglTreeInfo=CglTreeInfo\(\)\) const](#)

7.1.1 Detailed Description

Base class definition for relations between auxiliaries.

Definition at line 32 of file [CouenneCrossConv.hpp](#).

7.1.2 Member Function Documentation

7.1.2.1 virtual int Couenne::AuxRelation::findRelations() [pure virtual]

Implemented in [Couenne::PowRel](#), [Couenne::BiProdDivRel](#), [Couenne::MultiProdRel](#), and [Couenne::SumLogAuxRel](#).

7.1.2.2 virtual void Couenne::AuxRelation::generateCuts(const OsiSolverInterface &, OsiCuts &, const CglTreeInfo = CglTreeInfo()) const [virtual]

Reimplemented in [Couenne::PowRel](#), [Couenne::BiProdDivRel](#), [Couenne::MultiProdRel](#), and [Couenne::SumLogAuxRel](#).

The documentation for this class was generated from the following file:

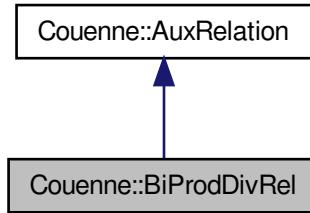
- /home/ted/COIN/trunk/Couenne/src/cut/crossconv/[CouenneCrossConv.hpp](#)

7.2 Couenne::BiProdDivRel Class Reference

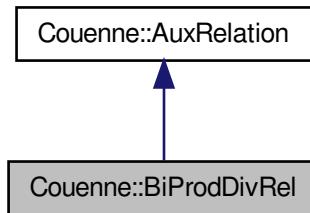
Identifies 5-tuple of the form.

```
#include <CouenneCrossConv.hpp>
```

Inheritance diagram for Couenne::BiProdDivRel:



Collaboration diagram for Couenne::BiProdDivRel:



Public Member Functions

- virtual int [findRelations \(\)](#)
- virtual void [generateCuts](#) (const OsiSolverInterface &, OsiCuts &, const CglTreeInfo=CglTreeInfo()) const

7.2.1 Detailed Description

Identifies 5-tuple of the form.

$x_j := x_i / x_k$ $x_p := x_i / x_q$

$x_l := x_j / x_p$ OR $x_l := x_j x_k$ $x_m := x_q / x_k$ $x_m := x_p x_q$

and generates, ONLY once, a cut

x_l = x_m (in both cases).

Definition at line 105 of file CouenneCrossConv.hpp.

7.2.2 Member Function Documentation

7.2.2.1 virtual int Couenne::BiProdDivRel::findRelations() [virtual]

Implements [Couenne::AuxRelation](#).

7.2.2.2 virtual void Couenne::BiProdDivRel::generateCuts(const OsiSolverInterface &, OsiCuts &, const CglTreeInfo = CglTreeInfo()) const [virtual]

Reimplemented from [Couenne::AuxRelation](#).

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/cut/crossconv/CouenneCrossConv.hpp

7.3 Couenne::CouenneExprMatrix::compare_pair_ind Struct Reference

```
#include <CouenneMatrix.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (register const std::pair< int, CouenneSparseVector * > &a, register const std::pair< int, CouenneSparseVector * > &b) const

7.3.1 Detailed Description

Definition at line 108 of file CouenneMatrix.hpp.

7.3.2 Member Function Documentation

7.3.2.1 bool Couenne::CouenneExprMatrix::compare_pair_ind::operator()(register const std::pair< int, CouenneSparseVector * > &a, register const std::pair< int, CouenneSparseVector * > &b) const [inline]

Definition at line 109 of file CouenneMatrix.hpp.

The documentation for this struct was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/[CouenneMatrix.hpp](#)

7.4 Couenne::CouenneSparseVector::compare_scalars Struct Reference

```
#include <CouenneMatrix.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (register CouenneScalar *const &a, register CouenneScalar *const &b)

7.4.1 Detailed Description

Definition at line 70 of file CouenneMatrix.hpp.

7.4.2 Member Function Documentation

7.4.2.1 `bool Couenne::CouenneSparseVector::compare_scalars::operator() (register CouenneScalar *const & a, register CouenneScalar *const & b) [inline]`

Definition at line 71 of file CouenneMatrix.hpp.

The documentation for this struct was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneMatrix.hpp

7.5 Couenne::compareSol Class Reference

class for comparing solutions (used in tabu list)

```
#include <CouenneFPpool.hpp>
```

Public Member Functions

- `bool operator() (const CouenneFPSolution &one, const CouenneFPSolution &two) const`

7.5.1 Detailed Description

class for comparing solutions (used in tabu list)

Definition at line 82 of file CouenneFPpool.hpp.

7.5.2 Member Function Documentation

7.5.2.1 `bool Couenne::compareSol::operator() (const CouenneFPSolution & one, const CouenneFPSolution & two) const`

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/heuristics/CouenneFPpool.hpp

7.6 Couenne::compExpr Struct Reference

Structure for comparing expressions.

```
#include <CouenneExprAux.hpp>
```

Public Member Functions

- `bool operator() (exprAux *e0, exprAux *e1) const`

7.6.1 Detailed Description

Structure for comparing expressions.

Used in compare() method for same-class expressions

Definition at line 210 of file CouenneExprAux.hpp.

7.6.2 Member Function Documentation

7.6.2.1 `bool Couenne::compExpr::operator() (exprAux * e0, exprAux * e1) const [inline]`

Definition at line 211 of file CouenneExprAux.hpp.

The documentation for this struct was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprAux.hpp](#)

7.7 Couenne::compNode Struct Reference

structure for comparing nodes in the dependence graph

```
#include <CouenneDepGraph.hpp>
```

Public Member Functions

- `bool operator() (const DepNode *n0, const DepNode *n1) const`
structure for comparing nodes

7.7.1 Detailed Description

structure for comparing nodes in the dependence graph

Definition at line 25 of file CouenneDepGraph.hpp.

7.7.2 Member Function Documentation

7.7.2.1 `bool Couenne::compNode::operator() (const DepNode * n0, const DepNode * n1) const [inline]`

structure for comparing nodes

Definition at line 108 of file CouenneDepGraph.hpp.

The documentation for this struct was generated from the following file:

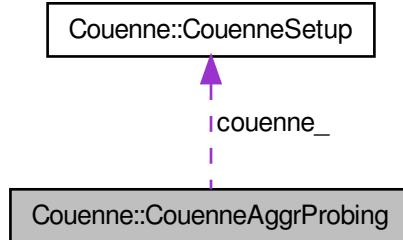
- /home/ted/COIN/trunk/Couenne/src/problem/depGraph/[CouenneDepGraph.hpp](#)

7.8 Couenne::CouenneAggrProbing Class Reference

Cut Generator for aggressive BT; i.e., an aggressive probing.

```
#include <CouenneAggrProbing.hpp>
```

Collaboration diagram for Couenne::CouenneAggrProbing:



Public Member Functions

- [CouenneAggrProbing \(CouenneSetup *couenne, const Ipopt::SmartPtr<Ipopt::OptionsList> options\)](#)
Constructor.
- [CouenneAggrProbing \(const CouenneAggrProbing &rhs\)](#)
Copy constructor.
- [~CouenneAggrProbing \(\)](#)
Destructor.
- [CouenneAggrProbing * clone \(\) const](#)
Clone method (necessary for the abstract CglCutGenerator class)
- void [generateCuts \(const OsiSolverInterface &solver, OsiCuts &cuts, const CglTreeInfo=CglTreeInfo\(\)\) const](#)
The main CglCutGenerator; not implemented yet.
- double [probeVariable \(int index, bool probeLower\)](#)
Probe one variable (try to tighten the lower or the upper bound, depending on the value of the second argument), so that we can generate the corresponding column cut.
- double [probeVariable2 \(int index, bool lower\)](#)
Alternative probing algorithm.
- void [setMaxTime \(double value\)](#)
Set/get maximum time to probe one variable.
- double [getMaxTime \(\) const](#)
- void [setMaxFailedSteps \(int value\)](#)
Set/get maximum number of failed steps.
- int [getMaxFailedSteps \(\) const](#)
- void [setMaxNodes \(int value\)](#)
Set/get maximum number of nodes to probe one variable.
- int [getMaxNodes \(\) const](#)
- void [setRestoreCutoff \(bool value\)](#)
Set/get restoreCutoff parameter (should we restore the initial cutoff value after each probing run?)
- bool [getRestoreCutoff \(\) const](#)

Static Public Member Functions

- static void [registerOptions](#) (Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions)
Add list of options to be read from file.

Protected Attributes

- [CouenneSetup * couenne_](#)
Pointer to the CouenneProblem representation.
- int [numCols_](#)
Number of columns (want to have this handy)
- double [maxTime_](#)
Maximum time to probe one variable.
- int [maxFailedSteps_](#)
Maximum number of failed iterations.
- int [maxNodes_](#)
Maximum number of nodes in probing.
- bool [restoreCutoff_](#)
Restore initial cutoff (value and solution)?
- double [initCutoff_](#)
Initial cutoff.

7.8.1 Detailed Description

Cut Generator for aggressive BT; i.e., an aggressive probing.

This probing strategy is very expensive and was initially developed to be run in parallel; hence, the user can choose to probe just a particular variable, without adding this cut generator to the list of cut generators normally employed by [Couenne](#). However, it can also be used in the standard way; in that case, it chooses automatically the variables to probe (in a very naive way, for the moment). TODO: Implement some way to automatically choose the variables TODO: Implement the generateCuts method, for use in Branch-and-Bound

Definition at line 37 of file CouenneAggrProbing.hpp.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 Couenne::CouenneAggrProbing::CouenneAggrProbing (CouenneSetup * couenne, const Ipopt::SmartPtr< Ipopt::OptionsList > options)

Constructor.

7.8.2.2 Couenne::CouenneAggrProbing::CouenneAggrProbing (const CouenneAggrProbing & rhs)

Copy constructor.

7.8.2.3 Couenne::CouenneAggrProbing::~CouenneAggrProbing ()

Destructor.

7.8.3 Member Function Documentation

7.8.3.1 `CouenneAggrProbing* Couenne::CouenneAggrProbing::clone() const [inline]`

Clone method (necessary for the abstract CglCutGenerator class)

Definition at line 52 of file CouenneAggrProbing.hpp.

7.8.3.2 `void Couenne::CouenneAggrProbing::generateCuts(const OsiSolverInterface & solver, OsiCuts & cuts, const CglTreeInfo =CglTreeInfo ()) const [inline]`

The main CglCutGenerator; not implemented yet.

Definition at line 56 of file CouenneAggrProbing.hpp.

7.8.3.3 `double Couenne::CouenneAggrProbing::probeVariable(int index, bool probeLower)`

Probe one variable (try to tighten the lower or the upper bound, depending on the value of the second argument), so that we can generate the corresponding column cut.

This runs the main algorithm. It returns the new bound (equal to the initial one if we could not tighten)

7.8.3.4 `double Couenne::CouenneAggrProbing::probeVariable2(int index, bool lower)`

Alternative probing algorithm.

This one does not work yet! Do not use, will probably segfault.

7.8.3.5 `static void Couenne::CouenneAggrProbing::registerOptions(Ipopt::SmartPtr< Bonmin::RegisteredOptions > options) [static]`

Add list of options to be read from file.

7.8.3.6 `void Couenne::CouenneAggrProbing::setMaxTime(double value)`

Set/get maximum time to probe one variable.

7.8.3.7 `double Couenne::CouenneAggrProbing::getMaxTime() const`

7.8.3.8 `void Couenne::CouenneAggrProbing::setMaxFailedSteps(int value)`

Set/get maximum number of failed steps.

7.8.3.9 `int Couenne::CouenneAggrProbing::getMaxFailedSteps() const`

7.8.3.10 `void Couenne::CouenneAggrProbing::setMaxNodes(int value)`

Set/get maximum number of nodes to probe one variable.

7.8.3.11 `int Couenne::CouenneAggrProbing::getMaxNodes() const`

7.8.3.12 `void Couenne::CouenneAggrProbing::setRestoreCutoff(bool value)`

Set/get restoreCutoff parameter (should we restore the initial cutoff value after each probing run?)

7.8.3.13 `bool Couenne::CouenneAggrProbing::getRestoreCutoff() const`

7.8.4 Member Data Documentation

7.8.4.1 CouenneSetup* Couenne::CouenneAggrProbing::couenne_ [protected]

Pointer to the [CouenneProblem](#) representation.

Definition at line 98 of file CouenneAggrProbing.hpp.

7.8.4.2 int Couenne::CouenneAggrProbing::numCols_ [protected]

Number of columns (want to have this handy)

Definition at line 101 of file CouenneAggrProbing.hpp.

7.8.4.3 double Couenne::CouenneAggrProbing::maxTime_ [protected]

Maximum time to probe one variable.

Definition at line 104 of file CouenneAggrProbing.hpp.

7.8.4.4 int Couenne::CouenneAggrProbing::maxFailedSteps_ [protected]

Maximum number of failed iterations.

Definition at line 107 of file CouenneAggrProbing.hpp.

7.8.4.5 int Couenne::CouenneAggrProbing::maxNodes_ [protected]

Maximum number of nodes in probing.

Definition at line 110 of file CouenneAggrProbing.hpp.

7.8.4.6 bool Couenne::CouenneAggrProbing::restoreCutoff_ [protected]

Restore initial cutoff (value and solution)?

Definition at line 113 of file CouenneAggrProbing.hpp.

7.8.4.7 double Couenne::CouenneAggrProbing::initCutoff_ [protected]

Initial cutoff.

Definition at line 116 of file CouenneAggrProbing.hpp.

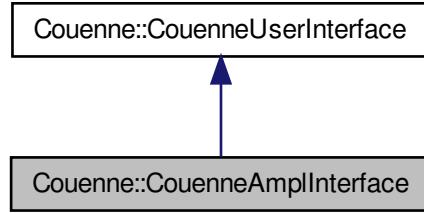
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/bound_tightening/[CouenneAggrProbing.hpp](#)

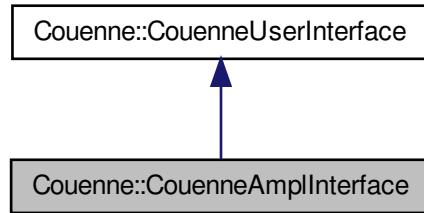
7.9 Couenne::CouenneAmplInterface Class Reference

```
#include <CouenneAmplInterface.hpp>
```

Inheritance diagram for Couenne::CouenneAmplInterface:



Collaboration diagram for Couenne::CouenneAmplInterface:



Public Member Functions

- [CouenneAmplInterface](#) (`Ipopt::SmartPtr< Ipopt::OptionsList > options_, Ipopt::SmartPtr< Ipopt::Journalist > jnlst_`)
- [~CouenneAmplInterface \(\)](#)
- [CouenneProblem * getCouenneProblem \(\)](#)
Should return the problem to solve in algebraic form.
- [Ipopt::SmartPtr< Bonmin::TMINLP > getTMINLP \(\)](#)
Should return the problem to solve as TMINLP.
- [bool writeSolution \(Bonmin::Bab &bab\)](#)
Called after B&B finished.
- [void setRegisteredOptions \(Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions_\)](#)

Static Public Member Functions

- static void [registerOptions](#) (`Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions`)

Additional Inherited Members

7.9.1 Detailed Description

Definition at line 26 of file CouenneAmplInterface.hpp.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 `Couenne::CouenneAmplInterface::CouenneAmplInterface (Iopt::SmartPtr< Iopt::OptionsList > options_, Iopt::SmartPtr< Iopt::Journalist > jnlist_) [inline]`

Definition at line 41 of file CouenneAmplInterface.hpp.

7.9.2.2 `Couenne::CouenneAmplInterface::~CouenneAmplInterface ()`

7.9.3 Member Function Documentation

7.9.3.1 `static void Couenne::CouenneAmplInterface::registerOptions (Iopt::SmartPtr< Bonmin::RegisteredOptions > roptions) [static]`

7.9.3.2 `CouenneProblem* Couenne::CouenneAmplInterface::getCouenneProblem () [virtual]`

Should return the problem to solve in algebraic form.

NOTE: Couenne is (currently) going to modify this problem!

Implements [Couenne::CouenneUserInterface](#).

7.9.3.3 `Iopt::SmartPtr<Bonmin::TMINLP> Couenne::CouenneAmplInterface::getTMINLP () [virtual]`

Should return the problem to solve as TMINLP.

Implements [Couenne::CouenneUserInterface](#).

7.9.3.4 `bool Couenne::CouenneAmplInterface::writeSolution (Bonmin::Bab & bab) [virtual]`

Called after B&B finished.

Should write solution information.

Reimplemented from [Couenne::CouenneUserInterface](#).

7.9.3.5 `void Couenne::CouenneAmplInterface::setRegisteredOptions (Iopt::SmartPtr< Bonmin::RegisteredOptions > roptions_) [inline]`

Definition at line 53 of file CouenneAmplInterface.hpp.

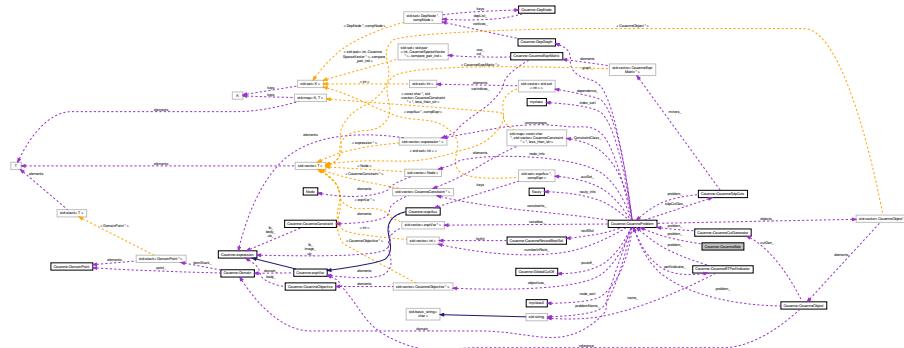
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/readnl/[CouenneAmplInterface.hpp](#)

7.10 Couenne::CouenneBab Class Reference

```
#include <CouenneBab.hpp>
```

Collaboration diagram for Couenne::CouenneBab:



Public Member Functions

- [CouenneBab \(\)](#)
Constructor.
- [virtual ~CouenneBab \(\)](#)
Destructor.
- [void setProblem \(CouenneProblem *p\)](#)
- [virtual void branchAndBound \(Bonmin::BabSetupBase &s\)](#)
Carry out branch and bound.
- [const double * bestSolution \(\) const](#)
Get the best solution known to the problem (is optimal if MipStatus is FeasibleOptimal).
- [double bestObj \(\) const](#)
Return objective value of the bestSolution.
- [double bestBound \(\)](#)
return the best known lower bound on the objective value

Protected Attributes

- [CouenneProblem * problem_](#)

7.10.1 Detailed Description

Definition at line 21 of file CouenneBab.hpp.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 Couenne::CouenneBab::CouenneBab ()

Constructor.

7.10.2.2 virtual Couenne::CouenneBab::~CouenneBab () [virtual]

Destructor.

7.10.3 Member Function Documentation

7.10.3.1 `void Couenne::CouenneBab::setProblem (CouenneProblem * p)`

7.10.3.2 `virtual void Couenne::CouenneBab::branchAndBound (Bonmin::BabSetupBase & s) [virtual]`

Carry out branch and bound.

7.10.3.3 `const double* Couenne::CouenneBab::bestSolution () const`

Get the best solution known to the problem (is optimal if MipStatus is FeasibleOptimal).

If no solution is known returns NULL.

7.10.3.4 `double Couenne::CouenneBab::bestObj () const`

Return objective value of the bestSolution.

7.10.3.5 `double Couenne::CouenneBab::bestBound () [inline]`

return the best known lower bound on the objective value

Definition at line 42 of file CouenneBab.hpp.

7.10.4 Member Data Documentation

7.10.4.1 `CouenneProblem* Couenne::CouenneBab::problem_ [protected]`

Definition at line 46 of file CouenneBab.hpp.

The documentation for this class was generated from the following file:

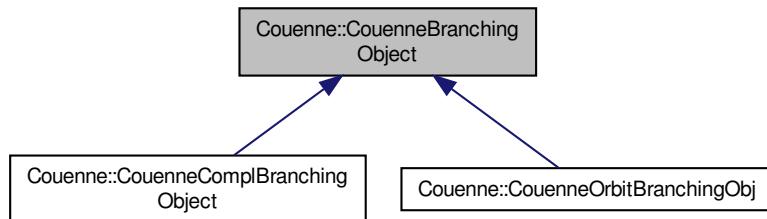
- /home/ted/COIN/trunk/Couenne/src/main/[CouenneBab.hpp](#)

7.11 Couenne::CouenneBranchingObject Class Reference

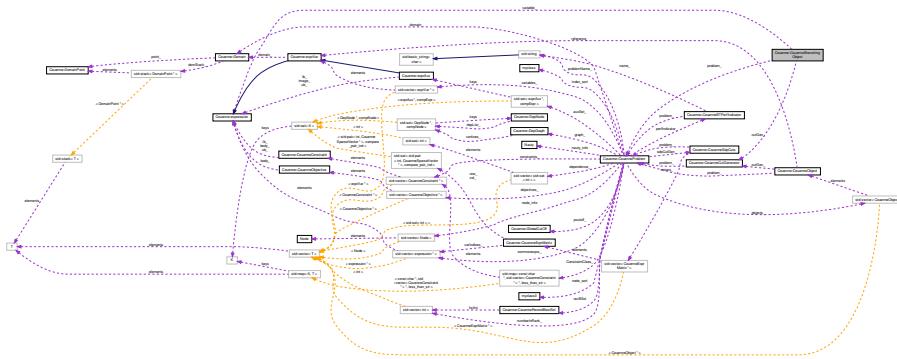
"Spatial" branching object.

```
#include <CouenneBranchingObject.hpp>
```

Inheritance diagram for Couenne::CouenneBranchingObject:



Collaboration diagram for Couenne::CouenneBranchingObject:



Public Member Functions

- **CouenneBranchingObject** (OsiSolverInterface *solver, const OsiObject *originalObject, JnlstPtr jnlst, **CouenneCutGenerator** *c, **CouenneProblem** *p, **expression** *var, int way, **CouNumber** brpoint, bool doFBBT, bool doConvCuts)

Constructor.
- **CouenneBranchingObject** (const **CouenneBranchingObject** &src)

Copy constructor.
- virtual OsiBranchingObject * **clone** () const

cloning method
- virtual double **branch** (OsiSolverInterface *solver=NULL)

Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.
- virtual bool **boundBranch** () const

does this branching object only change variable bounds?
- void **setSimulate** (bool s)

set simulate_ field below
- **expression** * **variable** ()

return branching variable
- void **branchCore** (OsiSolverInterface *, int, int, bool, double, **t_chg_bounds** *&)

Perform branching step.

Static Public Attributes

- static int **nOrbBr**
- static int **maxDepthOrbBranch**
- static int **nSGcomputations**

Protected Attributes

- **CouenneCutGenerator** * **cutGen_**

*Pointer to **CouenneCutGenerator** (if any); if not NULL, allows to do extra cut generation during branching.*
- **CouenneProblem** * **problem_**

- **`Pointer to CouenneProblem (necessary to allow FBBT)`**
- **`expression * variable_`**
The index of the variable this branching object refers to.
- **`JnlstPtr jnlst_`**
SmartPointer to the Journalist.
- **`bool doFBBT_`**
shall we do Feasibility based Bound Tightening (FBBT) at branching?
- **`bool doConvCuts_`**
shall we add convexification cuts at branching?
- **`double downEstimate_`**
down branch estimate (done at selectBranch with reduced costs)
- **`double upEstimate_`**
up branch estimate
- **`bool simulate_`**
are we currently in strong branching?

7.11.1 Detailed Description

"Spatial" branching object.

Branching can also be performed on continuous variables.

Definition at line 37 of file CouenneBranchingObject.hpp.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Couenne::CouenneBranchingObject::CouenneBranchingObject (`OsiSolverInterface * solver, const OsiObject * originalObject, JnlstPtr jnlst, CouenneCutGenerator * c, CouenneProblem * p, expression * var, int way, CouNumber brpoint, bool doFBBT, bool doConvCuts`)

Constructor.

**7.11.2.2 Couenne::CouenneBranchingObject::CouenneBranchingObject (`const CouenneBranchingObject & src`)
`[inline]`**

Copy constructor.

Definition at line 54 of file CouenneBranchingObject.hpp.

7.11.3 Member Function Documentation

7.11.3.1 virtual OsiBranchingObject* Couenne::CouenneBranchingObject::clone () const [inline], [virtual]

cloning method

Reimplemented in [Couenne::CouenneOrbitBranchingObj](#), and [Couenne::CouenneComplBranchingObject](#).

Definition at line 68 of file CouenneBranchingObject.hpp.

7.11.3.2 virtual double Couenne::CouenneBranchingObject::branch (OsiSolverInterface * *solver* =NULL) [virtual]

Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.

Returns change in guessed objective on next branch

Reimplemented in [Couenne::CouenneOrbitBranchingObj](#), and [Couenne::CouenneComplBranchingObject](#).

7.11.3.3 virtual bool Couenne::CouenneBranchingObject::boundBranch () const [inline], [virtual]

does this branching object only change variable bounds?

Reimplemented in [Couenne::CouenneOrbitBranchingObj](#).

Definition at line 79 of file CouenneBranchingObject.hpp.

7.11.3.4 void Couenne::CouenneBranchingObject::setSimulate (bool *s*) [inline]

set simulate_ field below

Reimplemented in [Couenne::CouenneOrbitBranchingObj](#).

Definition at line 83 of file CouenneBranchingObject.hpp.

7.11.3.5 expression* Couenne::CouenneBranchingObject::variable () [inline]

return branching variable

Definition at line 87 of file CouenneBranchingObject.hpp.

7.11.3.6 void Couenne::CouenneBranchingObject::branchCore (OsiSolverInterface * , int , int , bool , double , t_chg_bounds * &)

Perform branching step.

7.11.4 Member Data Documentation

7.11.4.1 int Couenne::CouenneBranchingObject::nOrbBr [static]

Definition at line 94 of file CouenneBranchingObject.hpp.

7.11.4.2 int Couenne::CouenneBranchingObject::maxDepthOrbBranch [static]

Definition at line 95 of file CouenneBranchingObject.hpp.

7.11.4.3 int Couenne::CouenneBranchingObject::nSGcomputations [static]

Definition at line 96 of file CouenneBranchingObject.hpp.

7.11.4.4 CouenneCutGenerator* Couenne::CouenneBranchingObject::cutGen_ [protected]

Pointer to [CouenneCutGenerator](#) (if any); if not NULL, allows to do extra cut generation during branching.

Definition at line 102 of file CouenneBranchingObject.hpp.

7.11.4.5 CouenneProblem* Couenne::CouenneBranchingObject::problem_ [protected]

Pointer to [CouenneProblem](#) (necessary to allow FBBT)

Definition at line 105 of file CouenneBranchingObject.hpp.

7.11.4.6 expression* Couenne::CouenneBranchingObject::variable_ [protected]

The index of the variable this branching object refers to.

If the corresponding [CouenneObject](#) was created on $w=f(x,y)$, it is either x or y , chosen previously with a call to `getFixVar()` expression *reference_;

Definition at line 111 of file CouenneBranchingObject.hpp.

7.11.4.7 JnlstPtr Couenne::CouenneBranchingObject::jnlst_ [protected]

SmartPointer to the Journalist.

Definition at line 114 of file CouenneBranchingObject.hpp.

7.11.4.8 bool Couenne::CouenneBranchingObject::doFBBT_ [protected]

shall we do Feasibility based Bound Tightening (FBBT) at branching?

Definition at line 117 of file CouenneBranchingObject.hpp.

7.11.4.9 bool Couenne::CouenneBranchingObject::doConvCuts_ [protected]

shall we add convexification cuts at branching?

Definition at line 120 of file CouenneBranchingObject.hpp.

7.11.4.10 double Couenne::CouenneBranchingObject::downEstimate_ [protected]

down branch estimate (done at selectBranch with reduced costs)

Definition at line 123 of file CouenneBranchingObject.hpp.

7.11.4.11 double Couenne::CouenneBranchingObject::upEstimate_ [protected]

up branch estimate

Definition at line 126 of file CouenneBranchingObject.hpp.

7.11.4.12 bool Couenne::CouenneBranchingObject::simulate_ [protected]

are we currently in strong branching?

Definition at line 129 of file CouenneBranchingObject.hpp.

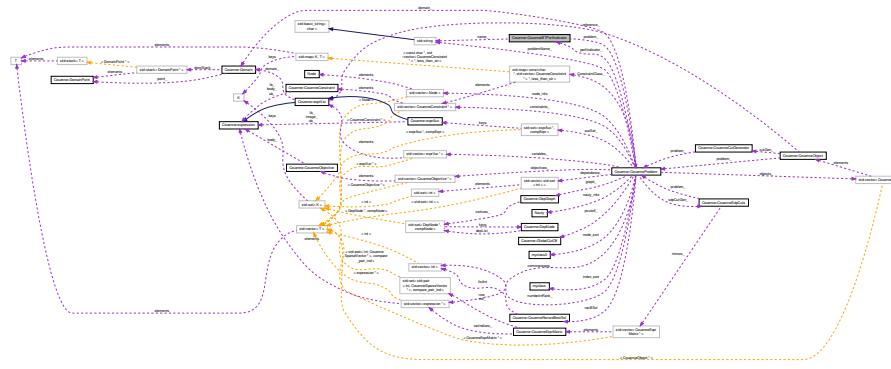
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/branch/[CouenneBranchingObject.hpp](#)

7.12 Couenne::CouenneBTPerfIndicator Class Reference

```
#include <CouenneBTPerfIndicator.hpp>
```

Collaboration diagram for Couenne::CouenneBTPerIndicator:



Public Member Functions

- `CouenneBTPerIndicator (CouenneProblem *p, const std::string &name)`
Should stats be printed at the end? Copied from problem_ -> Jnlst () -> ProduceOutput (ERROR, BOUNDSTIGHTENING)
- `~CouenneBTPerIndicator ()`
- `CouenneBTPerIndicator (const CouenneBTPerIndicator &rhs)`
- `CouenneBTPerIndicator & operator= (const CouenneBTPerIndicator &rhs)`
- `void setOldBounds (const CouNumber *lb, const CouNumber *ub) const`
- `void addToTimer (double time) const`
add to timer
- `void update (const CouNumber *lb, const CouNumber *ub, int depth) const`

Protected Attributes

- `std::string name_`
- `double nFixed_`
Whose performance is this?
- `double boundRatio_`
number of fixed variables
- `double shrunkInf_`
average bound width shrinkage
- `double shrunkDoubleInf_`
average # bounds that went from infinite to finite (counts twice if [-inf,inf] to [a,b])
- `double nProvedInfeas_`
average # bounds that went from doubly infinite to infinite
- `double weightSum_`
average # proofs of infeasibility
- `double * oldLB_`
total weight (used to give an average indicator at the end of Couenne)
- `double * oldUB_`
old lower bounds (initial, i.e. before BT)
- `double totalTime_`
old upper bounds

- int `nRuns_`
CPU time spent on this.
- `CouenneProblem * problem_`
number of runs
- bool `stats_`
Couenne problem info.

7.12.1 Detailed Description

Definition at line 23 of file CouenneBTPerIndicator.hpp.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Couenne::CouenneBTPerIndicator::CouenneBTPerIndicator (`CouenneProblem * p, const std::string & name`)

Should stats be printed at the end? Copied from `problem_->Jnlst () -> ProduceOutput (ERROR, BOUND TIGHTENING)`

7.12.2.2 Couenne::CouenneBTPerIndicator::~CouenneBTPerIndicator ()

7.12.2.3 Couenne::CouenneBTPerIndicator::CouenneBTPerIndicator (`const CouenneBTPerIndicator & rhs`)

7.12.3 Member Function Documentation

7.12.3.1 CouenneBTPerIndicator& Couenne::CouenneBTPerIndicator::operator= (`const CouenneBTPerIndicator & rhs`)

7.12.3.2 void Couenne::CouenneBTPerIndicator::setOldBounds (`const CouNumber * lb, const CouNumber * ub`) const

7.12.3.3 void Couenne::CouenneBTPerIndicator::addToTimer (`double time`) const

add to timer

7.12.3.4 void Couenne::CouenneBTPerIndicator::update (`const CouNumber * lb, const CouNumber * ub, int depth`) const

7.12.4 Member Data Documentation

7.12.4.1 std::string Couenne::CouenneBTPerIndicator::name_ [protected]

Definition at line 27 of file CouenneBTPerIndicator.hpp.

7.12.4.2 double Couenne::CouenneBTPerIndicator::nFixed_ [mutable], [protected]

Whose performance is this?

Definition at line 29 of file CouenneBTPerIndicator.hpp.

7.12.4.3 double Couenne::CouenneBTPerIndicator::boundRatio_ [mutable], [protected]

number of fixed variables

Definition at line 30 of file CouenneBTPerIndicator.hpp.

7.12.4.4 double Couenne::CouenneBTPerIndicator::shrunkenf_ [mutable], [protected]

average bound width shrinkage

Definition at line 31 of file CouenneBTPerIndicator.hpp.

7.12.4.5 double Couenne::CouenneBTPerIndicator::shrunkenDoublef_ [mutable], [protected]

average # bounds that went from infinite to finite (counts twice if [-inf,inf] to [a,b])

Definition at line 32 of file CouenneBTPerIndicator.hpp.

7.12.4.6 double Couenne::CouenneBTPerIndicator::nProvedInfeas_ [mutable], [protected]

average # bounds that went from doubly infinite to infinite

Definition at line 33 of file CouenneBTPerIndicator.hpp.

7.12.4.7 double Couenne::CouenneBTPerIndicator::weightSum_ [mutable], [protected]

average # proofs of infeasibility

Definition at line 35 of file CouenneBTPerIndicator.hpp.

7.12.4.8 double* Couenne::CouenneBTPerIndicator::oldLB_ [mutable], [protected]

total weight (used to give an average indicator at the end of [Couenne](#))

Definition at line 37 of file CouenneBTPerIndicator.hpp.

7.12.4.9 double* Couenne::CouenneBTPerIndicator::oldUB_ [mutable], [protected]

old lower bounds (initial, i.e. before BT)

Definition at line 38 of file CouenneBTPerIndicator.hpp.

7.12.4.10 double Couenne::CouenneBTPerIndicator::totalTime_ [mutable], [protected]

old upper bounds

Definition at line 40 of file CouenneBTPerIndicator.hpp.

7.12.4.11 int Couenne::CouenneBTPerIndicator::nRuns_ [mutable], [protected]

CPU time spent on this.

Definition at line 42 of file CouenneBTPerIndicator.hpp.

7.12.4.12 CouenneProblem* Couenne::CouenneBTPerIndicator::problem_ [protected]

number of runs

Definition at line 44 of file CouenneBTPerIndicator.hpp.

7.12.4.13 bool Couenne::CouenneBTPerIndicator::stats_ [protected]

[Couenne](#) problem info.

Definition at line 46 of file CouenneBTPerIndicator.hpp.

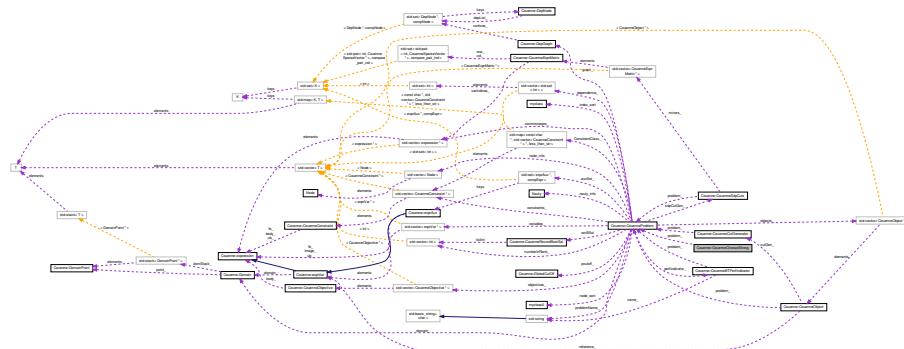
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/bound_tightening/[CouenneBTPerIndicator.hpp](#)

7.13 Couenne::CouenneChooseStrong Class Reference

```
#include <CouenneChooseStrong.hpp>
```

Collaboration diagram for Couenne::CouenneChooseStrong:



Public Member Functions

- **CouenneChooseStrong** (Bonmin::BabSetupBase &b, CouenneProblem *problem, JnlstPtr jnlst)

Constructor from solver (so we can set up arrays etc)
- **CouenneChooseStrong** (const CouenneChooseStrong &)

Copy constructor.
- **CouenneChooseStrong & operator=** (const CouenneChooseStrong &rhs)

Assignment operator.
- **virtual OsiChooseVariable * clone () const**

Clone.
- **virtual ~CouenneChooseStrong ()**

Destructor.
- **virtual int setupList (OsiBranchingInformation *info, bool initialize)**

Sets up strong list and clears all if initialize is true.
- **int gutsOfSetupList (OsiBranchingInformation *info, bool initialize)**
- **virtual int doStrongBranching (OsiSolverInterface *OsiSolver, OsiBranchingInformation *info, int numberToDo, int returnCriterion)**

This is a utility function which does strong branching on a list of objects and stores the results in OsiHotInfo.objects.
- **virtual bool feasibleSolution (const OsiBranchingInformation *info, const double *solution, int numberObjects, const OsiObject **objects)**

Returns true if solution looks feasible against given objects.
- **virtual int chooseVariable (OsiSolverInterface *solver, OsiBranchingInformation *info, bool fixVariables)**

choose object to branch based on earlier setup

Static Public Member Functions

- **static void registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions)**

Add list of options to be read from file.

Protected Member Functions

- int **simulateBranch** (OsiObject *Object, OsiBranchingInformation *info, OsiBranchingObject *branch, OsiSolverInterface *solver, Bonmin::HotInfo *result, int direction)

does one side of the branching

Protected Attributes

- **CouenneProblem * problem_**

Pointer to the associated MINLP problem.
- bool **pseudoUpdateLP_**

should we update the pseudocost multiplier with the distance between the LP point and the solution of the resulting branches' LPs? If so, this only happens in strong branching
- bool **estimateProduct_**

Normally, a convex combination of the min/max lower bounds' estimates is taken to select a branching variable, as in the original definition of strong branching.
- **JnlstPtr jnlst_**

pointer to journalist for detailed information
- double **branchtime_**

total time spent in strong branching

7.13.1 Detailed Description

Definition at line 23 of file CouenneChooseStrong.hpp.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 Couenne::CouenneChooseStrong::CouenneChooseStrong (Bonmin::BabSetupBase & b, CouenneProblem * problem, JnlstPtr jnlst)

Constructor from solver (so we can set up arrays etc)

7.13.2.2 Couenne::CouenneChooseStrong::CouenneChooseStrong (const CouenneChooseStrong &)

Copy constructor.

7.13.2.3 virtual Couenne::CouenneChooseStrong::~CouenneChooseStrong () [virtual]

Destructor.

7.13.3 Member Function Documentation

7.13.3.1 CouenneChooseStrong& Couenne::CouenneChooseStrong::operator= (const CouenneChooseStrong & rhs)

Assignment operator.

7.13.3.2 virtual OsiChooseVariable* Couenne::CouenneChooseStrong::clone () const [virtual]

Clone.

7.13.3.3 `virtual int Couenne::CouenneChooseStrong::setupList (OsiBranchingInformation * info, bool initialize) [virtual]`

Sets up strong list and clears all if initialize is true.

Returns number of infeasibilities.

7.13.3.4 `int Couenne::CouenneChooseStrong::gutsOfSetupList (OsiBranchingInformation * info, bool initialize)`

7.13.3.5 `virtual int Couenne::CouenneChooseStrong::doStrongBranching (OsiSolverInterface * OsiSolver, OsiBranchingInformation * info, int numberToDo, int returnCriterion) [virtual]`

This is a utility function which does strong branching on a list of objects and stores the results in OsiHotInfo.objects.

On entry the object sequence is stored in the OsiHotInfo object and maybe more. It returns -1 - one branch was infeasible both ways 0 - all inspected - nothing can be fixed 1 - all inspected - some can be fixed (returnCriterion==0) 2 - may be returning early - one can be fixed (last one done) (returnCriterion==1) 3 - returning because max time

7.13.3.6 `virtual bool Couenne::CouenneChooseStrong::feasibleSolution (const OsiBranchingInformation * info, const double * solution, int numberObjects, const OsiObject ** objects) [virtual]`

Returns true if solution looks feasible against given objects.

7.13.3.7 `virtual int Couenne::CouenneChooseStrong::chooseVariable (OsiSolverInterface * solver, OsiBranchingInformation * info, bool fixVariables) [virtual]`

choose object to branch based on earlier setup

7.13.3.8 `static void Couenne::CouenneChooseStrong::registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions) [static]`

Add list of options to be read from file.

7.13.3.9 `int Couenne::CouenneChooseStrong::simulateBranch (OsiObject * Object, OsiBranchingInformation * info, OsiBranchingObject * branch, OsiSolverInterface * solver, Bonmin::HotInfo * result, int direction) [protected]`

does one side of the branching

7.13.4 Member Data Documentation

7.13.4.1 `CouenneProblem* Couenne::CouenneChooseStrong::problem_ [protected]`

Pointer to the associated MINLP problem.

Definition at line 119 of file CouenneChooseStrong.hpp.

7.13.4.2 `bool Couenne::CouenneChooseStrong::pseudoUpdateLP_ [protected]`

should we update the pseudocost multiplier with the distance between the LP point and the solution of the resulting branches' LPs? If so, this only happens in strong branching

Definition at line 124 of file CouenneChooseStrong.hpp.

7.13.4.3 `bool Couenne::CouenneChooseStrong::estimateProduct_ [protected]`

Normally, a convex combination of the min/max lower bounds' estimates is taken to select a branching variable, as in the original definition of strong branching.

If this option is set to true, their product is taken instead:

$(1e-6+\min) * \max$

where the $1e-6$ is used to ensure that even those with one subproblem with no improvement are compared.

Definition at line 135 of file CouenneChooseStrong.hpp.

7.13.4.4 `JnlstPtr Couenne::CouenneChooseStrong::jnlist_` [protected]

pointer to journalist for detailed information

Definition at line 138 of file CouenneChooseStrong.hpp.

7.13.4.5 `double Couenne::CouenneChooseStrong::branchtime_` [protected]

total time spent in strong branching

Definition at line 141 of file CouenneChooseStrong.hpp.

The documentation for this class was generated from the following file:

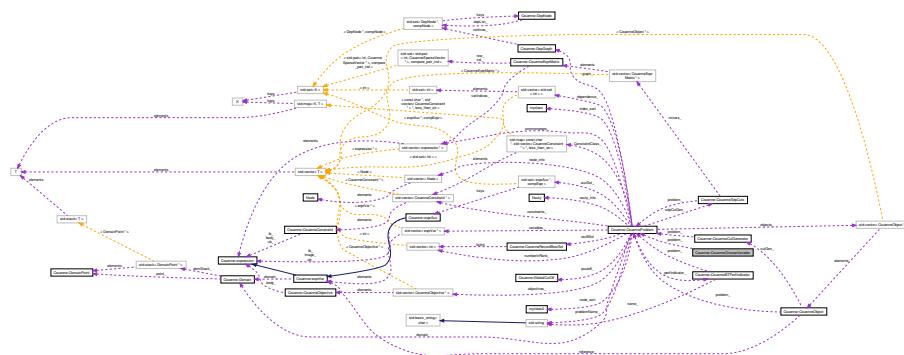
- /home/ted/COIN/trunk/Couenne/src/branch/CouenneChooseStrong.hpp

7.14 Couenne::CouenneChooseVariable Class Reference

Choose a variable for branching.

```
#include <CouenneChooseVariable.hpp>
```

Collaboration diagram for Couenne::CouenneChooseVariable:



Public Member Functions

- `CouenneChooseVariable ()`
Default Constructor.
- `CouenneChooseVariable (const OsiSolverInterface *, CouenneProblem *, JnlstPtr jnlist)`
Constructor from solver (so we can set up arrays etc)
- `CouenneChooseVariable (const CouenneChooseVariable &)`
Copy constructor.
- `CouenneChooseVariable & operator= (const CouenneChooseVariable &)`
Assignment operator.
- `virtual OsiChooseVariable * clone () const`
Clone.

- virtual `~CouenneChooseVariable ()`
Destructor.
- virtual int `setupList (OsiBranchingInformation *, bool)`
Sets up strong list and clears all if initialize is true.
- virtual bool `feasibleSolution (const OsiBranchingInformation *info, const double *solution, int numberObjects, const OsiObject **objects)`
Returns true if solution looks feasible against given objects.

Static Public Member Functions

- static void `registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions)`
Add list of options to be read from file.

Protected Attributes

- `CouenneProblem * problem_`
Pointer to the associated MINLP problem.
- `JnlstPtr jnlst_`
journalist for detailed debug information

7.14.1 Detailed Description

Choose a variable for branching.

Definition at line 27 of file CouenneChooseVariable.hpp.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 Couenne::CouenneChooseVariable::CouenneChooseVariable ()

Default Constructor.

7.14.2.2 Couenne::CouenneChooseVariable::CouenneChooseVariable (const OsiSolverInterface *, CouenneProblem *, JnlstPtr jnlst)

Constructor from solver (so we can set up arrays etc)

7.14.2.3 Couenne::CouenneChooseVariable::CouenneChooseVariable (const CouenneChooseVariable &)

Copy constructor.

7.14.2.4 virtual Couenne::CouenneChooseVariable::~CouenneChooseVariable () [inline], [virtual]

Destructor.

Definition at line 48 of file CouenneChooseVariable.hpp.

7.14.3 Member Function Documentation

7.14.3.1 CouenneChooseVariable& Couenne::CouenneChooseVariable::operator= (const CouenneChooseVariable &)

Assignment operator.

7.14.3.2 virtual OsiChooseVariable* Couenne::CouenneChooseVariable::clone() const [inline], [virtual]

Clone.

Definition at line 44 of file CouenneChooseVariable.hpp.

7.14.3.3 virtual int Couenne::CouenneChooseVariable::setupList(OsiBranchingInformation * , bool) [virtual]

Sets up strong list and clears all if initialize is true.

Returns number of infeasibilities. If returns -1 then has worked out node is infeasible!

7.14.3.4 virtual bool Couenne::CouenneChooseVariable::feasibleSolution(const OsiBranchingInformation * info, const double * solution, int numberObjects, const OsiObject ** objects) [virtual]

Returns true if solution looks feasible against given objects.

7.14.3.5 static void Couenne::CouenneChooseVariable::registerOptions(Ipopt::SmartPtr< Bonmin::RegisteredOptions > options) [static]

Add list of options to be read from file.

7.14.4 Member Data Documentation

7.14.4.1 CouenneProblem* Couenne::CouenneChooseVariable::problem_ [protected]

Pointer to the associated MINLP problem.

Definition at line 72 of file CouenneChooseVariable.hpp.

7.14.4.2 JnlstPtr Couenne::CouenneChooseVariable::jnlst_ [protected]

journalist for detailed debug information

Definition at line 75 of file CouenneChooseVariable.hpp.

The documentation for this class was generated from the following file:

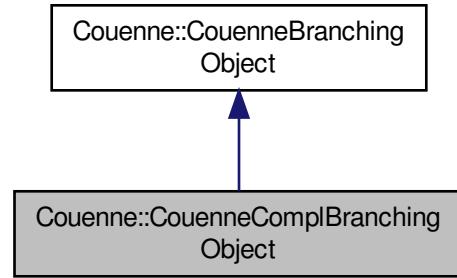
- /home/ted/COIN/trunk/Couenne/src/branch/CouenneChooseVariable.hpp

7.15 Couenne::CouenneComplBranchingObject Class Reference

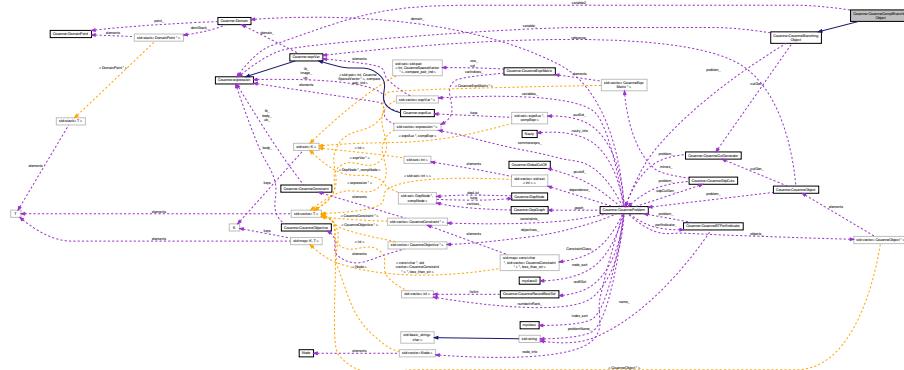
"Spatial" branching object for complementarity constraints.

```
#include <CouenneComplBranchingObject.hpp>
```

Inheritance diagram for Couenne::CouenneComplBranchingObject:



Collaboration diagram for Couenne::CouenneComplBranchingObject:



Public Member Functions

- **CouenneComplBranchingObject** (OsiSolverInterface *solver, const OsiObject *originalObject, JnlstPtr jnlst, CouenneCutGenerator *c, CouenneProblem *p, expression *var, expression *var2, int way, CouNumber brpoint, bool doFFBT, bool doConvCuts, int sign)

Constructor.
- **CouenneComplBranchingObject** (const CouenneComplBranchingObject &src)

Copy constructor.
- virtual OsiBranchingObject * **clone** () const

cloning method
- virtual double **branch** (OsiSolverInterface *solver=NULL)

Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.

Protected Attributes

- `expression * variable2_`
`use CouenneBranchingObject::variable_ as the first variable to set to 0, and this one as the second`
- `int sign_`
`-1 if object is for $xi * xj \leq 0$ +1 if object is for $xi * xj \leq 0$ 0 if object is for $xi * xj = 0$ (classical)`

Additional Inherited Members

7.15.1 Detailed Description

"Spatial" branching object for complementarity constraints.

Branching on such an object $x_1 x_2 = 0$ is performed by setting either $x_1=0$ or $x_2=0$

Definition at line 24 of file CouenneComplBranchingObject.hpp.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 Couenne::CouenneComplBranchingObject::CouenneComplBranchingObject (OsiSolverInterface * solver, const OsiObject * originalObject, JnlstPtr jnlst, CouenneCutGenerator * c, CouenneProblem * p, expression * var, expression * var2, int way, CouNumber brpoint, bool doFBBT, bool doConvCuts, int sign)

Constructor.

7.15.2.2 Couenne::CouenneComplBranchingObject::CouenneComplBranchingObject (const CouenneComplBranchingObject & src) [inline]

Copy constructor.

Definition at line 43 of file CouenneComplBranchingObject.hpp.

7.15.3 Member Function Documentation

7.15.3.1 virtual OsiBranchingObject* Couenne::CouenneComplBranchingObject::clone () const [inline], [virtual]

cloning method

Reimplemented from [Couenne::CouenneBranchingObject](#).

Definition at line 49 of file CouenneComplBranchingObject.hpp.

7.15.3.2 virtual double Couenne::CouenneComplBranchingObject::branch (OsiSolverInterface * solver = NULL) [virtual]

Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.

Returns change in guessed objective on next branch

Reimplemented from [Couenne::CouenneBranchingObject](#).

7.15.4 Member Data Documentation

7.15.4.1 expression* Couenne::CouenneComplBranchingObject::variable2_ [protected]

use [CouenneBranchingObject::variable_](#) as the first variable to set to 0, and this one as the second

Definition at line 63 of file CouenneComplBranchingObject.hpp.

7.15.4.2 int Couenne::CouenneComplBranchingObject::sign_ [protected]

-1 if object is for $x_i * x_j \leq 0$ +1 if object is for $x_i * x_j \leq 0$ 0 if object is for $x_i * x_j = 0$ (classical)

Definition at line 68 of file CouenneComplBranchingObject.hpp.

The documentation for this class was generated from the following file:

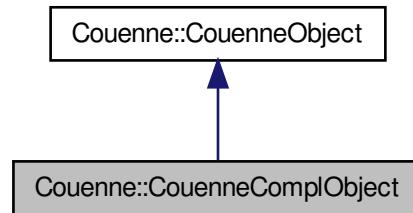
- [/home/ted/COIN/trunk/Couenne/src/branch/CouenneComplBranchingObject.hpp](#)

7.16 Couenne::CouenneComplObject Class Reference

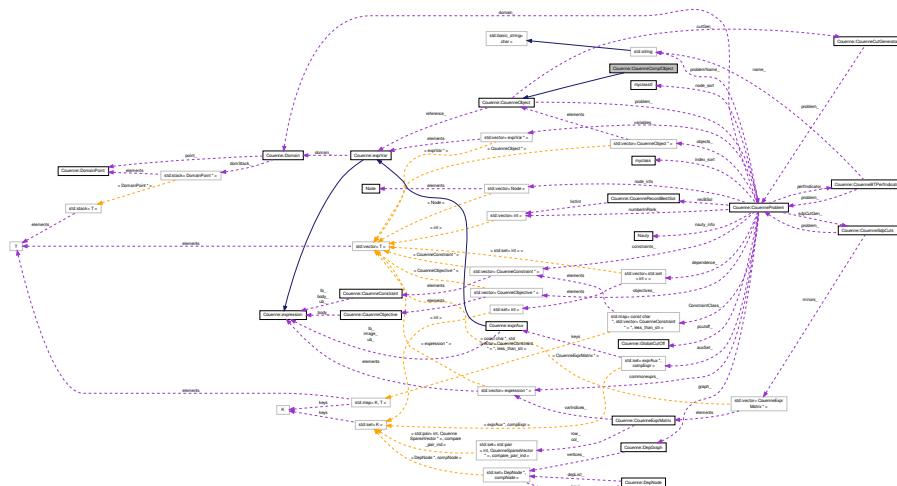
OsiObject for complementarity constraints $x_1 x_2 \geq, \leq, = 0$.

```
#include <CouenneComplObject.hpp>
```

Inheritance diagram for Couenne::CouenneComplObject:



Collaboration diagram for Couenne::CouenneComplObject



Public Member Functions

- `CouenneComplObject (CouenneCutGenerator *c, CouenneProblem *p, exprVar *ref, Bonmin::BabSetupBase *base, JnlstPtr jnlst, int sign)`
Constructor with information for branching point selection strategy.
- `CouenneComplObject (exprVar *ref, Bonmin::BabSetupBase *base, JnlstPtr jnlst, int sign)`
Constructor with lesser information, used for infeasibility only.
- `~CouenneComplObject ()`
Destructor.
- `CouenneComplObject (const CouenneComplObject &src)`
Copy constructor.
- `virtual CouenneObject * clone () const`
Cloning method.
- `virtual double infeasibility (const OsiBranchingInformation *info, int &way) const`
compute infeasibility of this variable, $|w - f(x)|$ (where w is the auxiliary variable defined as $w = f(x)$)
- `virtual double checkInfeasibility (const OsiBranchingInformation *info) const`
compute infeasibility of this variable, $|w - f(x)|$, where w is the auxiliary variable defined as $w = f(x)$
- `virtual OsiBranchingObject * createBranch (OsiSolverInterface *, const OsiBranchingInformation *, int way) const`
create CouenneBranchingObject or CouenneThreeWayBranchObj based on this object

Additional Inherited Members

7.16.1 Detailed Description

OsiObject for complementarity constraints $x_1x_2 \geq, \leq, = 0$.

Associated with two variables x_1 and x_2 , branches with either $x_1 = 0$ or $x_2 = 0$

Definition at line 22 of file CouenneComplObject.hpp.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 Couenne::CouenneComplObject::CouenneComplObject (CouenneCutGenerator * c, CouenneProblem * p, exprVar * ref, Bonmin::BabSetupBase * base, JnlstPtr jnlst, int sign)

Constructor with information for branching point selection strategy.

7.16.2.2 Couenne::CouenneComplObject::CouenneComplObject (exprVar * ref, Bonmin::BabSetupBase * base, JnlstPtr jnlst, int sign)

Constructor with lesser information, used for infeasibility only.

7.16.2.3 Couenne::CouenneComplObject::~CouenneComplObject () [inline]

Destructor.

Definition at line 37 of file CouenneComplObject.hpp.

7.16.2.4 Couenne::CouenneComplObject::CouenneComplObject (const CouenneComplObject & src)

Copy constructor.

7.16.3 Member Function Documentation

7.16.3.1 `virtual CouenneObject* Couenne::CouenneComplObject::clone() const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::CouenneObject](#).

Definition at line 43 of file [CouenneComplObject.hpp](#).

7.16.3.2 `virtual double Couenne::CouenneComplObject::infeasibility(const OsiBranchingInformation * info, int & way) const [virtual]`

compute infeasibility of this variable, $|w - f(x)|$ (where w is the auxiliary variable defined as $w = f(x)$)

Reimplemented from [Couenne::CouenneObject](#).

7.16.3.3 `virtual double Couenne::CouenneComplObject::checkInfeasibility(const OsiBranchingInformation * info) const [virtual]`

compute infeasibility of this variable, $|w - f(x)|$, where w is the auxiliary variable defined as $w = f(x)$

Reimplemented from [Couenne::CouenneObject](#).

7.16.3.4 `virtual OsiBranchingObject* Couenne::CouenneComplObject::createBranch(OsiSolverInterface *, const OsiBranchingInformation *, int way) const [virtual]`

create [CouenneBranchingObject](#) or [CouenneThreeWayBranchObj](#) based on this object

Reimplemented from [Couenne::CouenneObject](#).

The documentation for this class was generated from the following file:

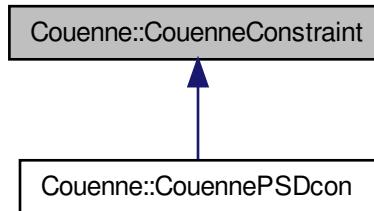
- /home/ted/COIN/trunk/Couenne/src/branch/[CouenneComplObject.hpp](#)

7.17 Couenne::CouenneConstraint Class Reference

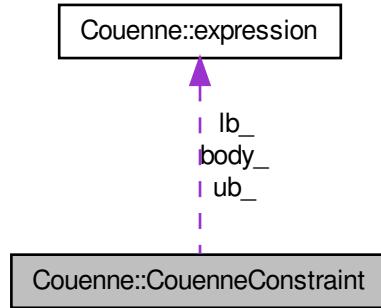
Class to represent nonlinear constraints.

```
#include <CouenneProblemElem.hpp>
```

Inheritance diagram for Couenne::CouenneConstraint:



Collaboration diagram for Couenne::CouenneConstraint:



Public Member Functions

- `CouenneConstraint (expression *body=NULL, expression *lb=NULL, expression *ub=NULL)`
Constructor.
- `virtual ~CouenneConstraint ()`
Destructor.
- `CouenneConstraint (const CouenneConstraint &c, Domain *d=NULL)`
Copy constructor.
- `virtual CouenneConstraint * clone (Domain *d=NULL) const`
Cloning method.
- `virtual expression * Lb () const`
Expression of lower bound.
- `virtual expression * Ub () const`
Expression of upper bound.
- `virtual expression * Body () const`
Expression of body of constraint.
- `virtual expression * Body (expression *newBody)`
Set body of constraint.
- `virtual exprAux * standardize (CouenneProblem *)`
decompose body of constraint through auxiliary variables
- `virtual void print (std::ostream &=std::cout)`
print constraint

Protected Attributes

- `expression * body_`
Body of constraint.
- `expression * lb_`
Lower bound (expression)

- `expression * ub_`
Upper bound (expression)

7.17.1 Detailed Description

Class to represent nonlinear constraints.

It consists of an expression as the body and two range expressions as lower- and upper bounds.

A general constraint is defined as `lb_ <= body_ <= ub_`, where all three components are expressions, depending on variables, auxiliaries and bounds. If the constraint is $2 \leq \exp(x_1+x_2) \leq 4$, then:

`body_ = exp (x1+x2)`, that is,

```
new exprExp (new exprSum (new exprVar (1), new exprVar (2))
while lb_ = new exprConst (2.) and ub_ = new exprConst (4.).
```

Definition at line 39 of file CouenneProblemElem.hpp.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Couenne::CouenneConstraint::CouenneConstraint (`expression * body = NULL, expression * lb = NULL, expression * ub = NULL`) [inline]

Constructor.

Definition at line 50 of file CouenneProblemElem.hpp.

7.17.2.2 virtual Couenne::CouenneConstraint::~CouenneConstraint () [inline], [virtual]

Destructor.

Definition at line 67 of file CouenneProblemElem.hpp.

7.17.2.3 Couenne::CouenneConstraint::CouenneConstraint (`const CouenneConstraint & c, Domain * d = NULL`) [inline]

Copy constructor.

Definition at line 74 of file CouenneProblemElem.hpp.

7.17.3 Member Function Documentation

7.17.3.1 virtual CouenneConstraint* Couenne::CouenneConstraint::clone (`Domain * d = NULL`) const [inline], [virtual]

Cloning method.

Reimplemented in [Couenne::CouennePSDcon](#).

Definition at line 80 of file CouenneProblemElem.hpp.

7.17.3.2 virtual expression* Couenne::CouenneConstraint::Lb () const [inline], [virtual]

Expression of lower bound.

Definition at line 84 of file CouenneProblemElem.hpp.

7.17.3.3 **virtual expression* Couenne::CouenneConstraint::Ub() const [inline], [virtual]**

Expression of upper bound.

Definition at line 85 of file CouenneProblemElem.hpp.

7.17.3.4 **virtual expression* Couenne::CouenneConstraint::Body() const [inline], [virtual]**

Expression of body of constraint.

Definition at line 86 of file CouenneProblemElem.hpp.

7.17.3.5 **virtual expression* Couenne::CouenneConstraint::Body(expression * newBody) [inline], [virtual]**

Set body of constraint.

Definition at line 89 of file CouenneProblemElem.hpp.

7.17.3.6 **virtual exprAux* Couenne::CouenneConstraint::standardize(CouenneProblem *) [virtual]**

decompose body of constraint through auxiliary variables

Reimplemented in [Couenne::CouennePSDcon](#).

7.17.3.7 **virtual void Couenne::CouenneConstraint::print(std::ostream & = std::cout) [virtual]**

print constraint

Reimplemented in [Couenne::CouennePSDcon](#).

7.17.4 Member Data Documentation

7.17.4.1 **expression* Couenne::CouenneConstraint::body_ [protected]**

Body of constraint.

Definition at line 43 of file CouenneProblemElem.hpp.

7.17.4.2 **expression* Couenne::CouenneConstraint::lb_ [protected]**

Lower bound (expression)

Definition at line 44 of file CouenneProblemElem.hpp.

7.17.4.3 **expression* Couenne::CouenneConstraint::ub_ [protected]**

Upper bound (expression)

Definition at line 45 of file CouenneProblemElem.hpp.

The documentation for this class was generated from the following file:

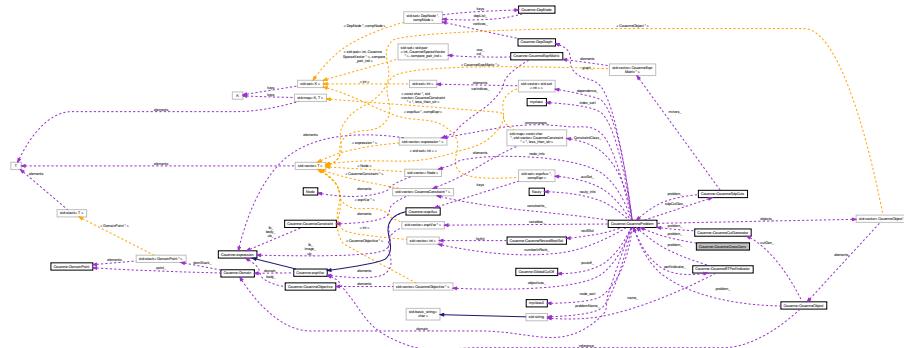
- /home/ted/COIN/trunk/Couenne/src/problem/[CouenneProblemElem.hpp](#)

7.18 Couenne::CouenneCrossConv Class Reference

Cut Generator that uses relationships between auxiliaries.

```
#include <CouenneCrossConv.hpp>
```

Collaboration diagram for Couenne::CouenneCrossConv:



Public Member Functions

- [CouenneCrossConv \(CouenneProblem *, JnlstPtr, const Ipopt::SmartPtr< Ipopt::OptionsList >\)](#)
constructor
- [CouenneCrossConv \(const CouenneCrossConv &\)](#)
copy constructor
- [virtual ~CouenneCrossConv \(\)](#)
destructor
- [virtual CouenneCrossConv * clone \(\) const](#)
clone method (necessary for the abstract CglCutGenerator class)
- [virtual void generateCuts \(const OsiSolverInterface &, OsiCuts &, const CglTreeInfo=CglTreeInfo\(\)\) const](#)
the main CglCutGenerator
- [virtual void setup \(\)](#)
Set up data structure to detect redundancies.

Static Public Member Functions

- [static void registerOptions \(Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions\)](#)
Add list of options to be read from file.

Protected Attributes

- [JnlstPtr jnlst_](#)
Journalist.
- [CouenneProblem * problem_](#)
pointer to the CouenneProblem representation

7.18.1 Detailed Description

Cut Generator that uses relationships between auxiliaries.

Definition at line 139 of file CouenneCrossConv.hpp.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 `Couenne::CouenneCrossConv::CouenneCrossConv (CouenneProblem * , JnlstPtr , const Ipopt::SmartPtr< Ipopt::OptionsList >)`

constructor

7.18.2.2 `Couenne::CouenneCrossConv::CouenneCrossConv (const CouenneCrossConv &)`

copy constructor

7.18.2.3 `virtual Couenne::CouenneCrossConv::~CouenneCrossConv () [virtual]`

destructor

7.18.3 Member Function Documentation

7.18.3.1 `virtual CouenneCrossConv* Couenne::CouenneCrossConv::clone () const [inline], [virtual]`

clone method (necessary for the abstract CglCutGenerator class)

Definition at line 155 of file CouenneCrossConv.hpp.

7.18.3.2 `virtual void Couenne::CouenneCrossConv::generateCuts (const OsiSolverInterface & , OsiCuts & , const CglTreeInfo = CglTreeInfo ()) const [virtual]`

the main CglCutGenerator

7.18.3.3 `static void Couenne::CouenneCrossConv::registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > options) [static]`

Add list of options to be read from file.

7.18.3.4 `virtual void Couenne::CouenneCrossConv::setup () [virtual]`

Set up data structure to detect redundancies.

7.18.4 Member Data Documentation

7.18.4.1 `JnlstPtr Couenne::CouenneCrossConv::jnlst_ [protected]`

Journalist.

Definition at line 176 of file CouenneCrossConv.hpp.

7.18.4.2 `CouenneProblem* Couenne::CouenneCrossConv::problem_ [protected]`

pointer to the `CouenneProblem` representation

Definition at line 179 of file CouenneCrossConv.hpp.

The documentation for this class was generated from the following file:

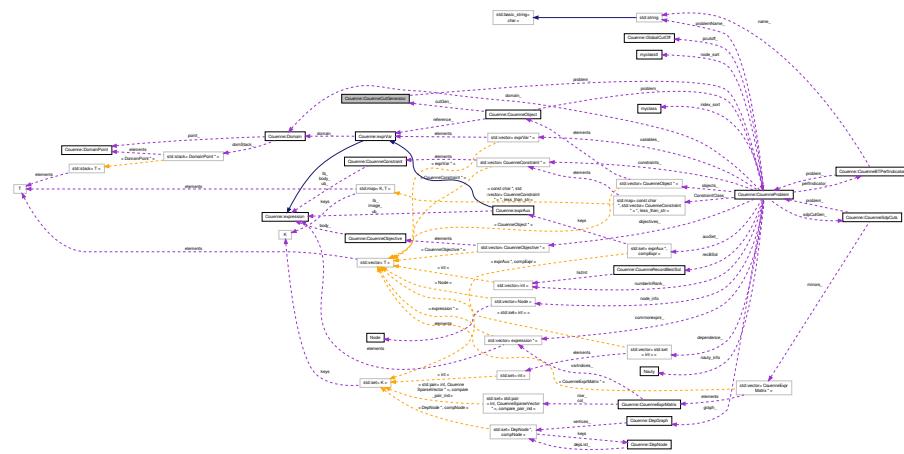
- /home/ted/COIN/trunk/Couenne/src/cut/crossconv/CouenneCrossConv.hpp

7.19 Couenne::CouenneCutGenerator Class Reference

Cut Generator for linear convexifications.

```
#include <CouenneCutGenerator.hpp>
```

Collaboration diagram for Couenne::CouenneCutGenerator:



Public Member Functions

- `CouenneCutGenerator` (`Bonmin::OsiTMINLPIInterface` *=`NULL`, `Bonmin::BabSetupBase` *`base`=`NULL`, `CouenneProblem` *=`NULL`, `struct ASL` *=`NULL`)
 - constructor*
 - `CouenneCutGenerator` (`const CouenneCutGenerator &`)
 - copy constructor*
 - `~CouenneCutGenerator` ()
 - destructor*
 - `CouenneCutGenerator * clone` () const
 - clone method (necessary for the abstract CglCutGenerator class)*
 - `CouenneProblem * Problem` () const
 - return pointer to symbolic problem*
 - void `setProblem` (`CouenneProblem` *`p`)
 - return pointer to symbolic problem*
 - int `getnvars` () const
 - total number of variables (original + auxiliary)*
 - bool `isFirst` () const
 - has generateCuts been called yet?*
 - bool `addViolated` () const
 - should we add the violated cuts only (true), or all of them (false)?*
 - enum `conv_type ConvType` () const
 - get convexification type (see CouenneTypes.h)*
 - int `nSamples` () const
 - get number of convexification samples*
 - void `generateCuts` (`const OsiSolverInterface &`, `OsiCuts &`, `const CglTreeInfo=CglTreeInfo()`) const

- *the main CglCutGenerator*
- int **createCut** (OsiCuts &, CouNumber, CouNumber, int, CouNumber, int=-1, CouNumber=0., int=-1, CouNumber=0., bool=false) const
 - create cut and check violation. Insert and return status*
- int **createCut** (OsiCuts &, CouNumber, int, int, CouNumber, int=-1, CouNumber=0., int=-1, CouNumber=0., bool=false) const
 - create cut and check violation. Other version with only one bound*
- void **addEnvelope** (OsiCuts &, int, unary_function, unary_function, int, int, CouNumber, CouNumber, CouNumber, t_chg_bounds *=NULL, bool=false) const
 - Add general linear envelope to convex function, given its variables' indices, the (univariate) function and its first derivative.*
- void **addEnvelope** (OsiCuts &, int, funtriplet *, int, int, CouNumber, CouNumber, CouNumber, CouNumber, t_chg_bounds *=NULL, bool=false) const
 - Add general linear envelope to convex function, given its variables' indices, the (univariate) function and its first derivative.*
- int **addSegment** (OsiCuts &, int, int, CouNumber, CouNumber, CouNumber, CouNumber, int) const
 - Add half-plane through (x1,y1) and (x2,y2) – resp.*
- int **addTangent** (OsiCuts &, int, int, CouNumber, CouNumber, CouNumber, CouNumber, int) const
 - add tangent at given poing (x,w) with given slope*
- void **setBabPtr** (Bonmin::Bab *p)
 - Method to set the Bab pointer.*
- void **getStats** (int &nrc, int &ntc, double &st)
 - Get statistics.*
- bool & **infeasNode** () const
 - Allow to get and set the infeasNode_ flag (used only in generateCuts())*
- void **genRowCuts** (const OsiSolverInterface &, OsiCuts &cs, int, int *, t_chg_bounds *=NULL) const
 - generate OsiRowCuts for current convexification*
- void **genColCuts** (const OsiSolverInterface &, OsiCuts &, int, int *) const
 - generate OsiColCuts for improved (implied and propagated) bounds*
- void **printLineInfo** () const
 - print node, depth, LB/UB/LP info*
- ConstJnlstPtr **Jnlst** () const
 - Provide Journalist.*
- void **setJnlst** (JnlstPtr jnlst__)
 -
- double & **rootTime** ()
 - Time spent at root node.*
- bool **check_lp** () const
 - return check_lp flag (used in CouenneSolverInterface)*
- bool **enableLpImpliedBounds** () const
 - returns value of enable_lp_implied_bounds_*

Static Public Member Functions

- static void **registerOptions** (Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions)
 - Add list of options to be read from file.*

Protected Attributes

- bool `firstcall_`
True if no convexification cuts have been generated yet for this problem.
- bool `addviolated_`
True if we should add the violated cuts only, false if all of them should be added.
- enum `conv_type convtype_`
what kind of sampling should be performed?
- int `nSamples_`
how many cuts should be added for each function?
- `CouenneProblem * problem_`
pointer to symbolic repr. of constraint, variables, and bounds
- int `nrootcuts_`
number of cuts generated at the first call
- int `ntotalcuts_`
total number of cuts generated
- double `septime_`
separation time (includes generation of problem)
- double `objValue_`
Record obj value at final point of CouenneConv.
- Bonmin::OsiTMINLPInterface * `nlp_`
nonlinear solver interface as used within Bonmin (used at first Couenne pass of each b&b node)
- Bonmin::Bab * `BabPtr_`
pointer to the Bab object (used to retrieve the current primal bound through bestObj())
- bool `infeasNode_`
signal infeasibility of current node (found through bound tightening)
- `JnlstPtr jnlst_`
SmartPointer to the Journalist.
- double `rootTime_`
Time spent at the root node.
- bool `check_lp_`
Check all generated LPs through an independent call to OsiClpSolverInterface::initialSolve()
- bool `enable_lp_implied_bounds_`
Take advantage of OsiClpSolverInterface::tightenBounds (), known to have caused some problems some time ago.
- int `lastPrintLine`
Running count of printed info lines.

7.19.1 Detailed Description

Cut Generator for linear convexifications.

Definition at line 49 of file CouenneCutGenerator.hpp.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 `Couenne::CouenneCutGenerator::CouenneCutGenerator (Bonmin::OsiTMINLPInterface * = NULL,
 Bonmin::BabSetupBase * base = NULL, CouenneProblem * = NULL, struct ASL * = NULL)`

constructor

7.19.2.2 Couenne::CouenneCutGenerator::CouenneCutGenerator (const CouenneCutGenerator &)

copy constructor

7.19.2.3 Couenne::CouenneCutGenerator::~CouenneCutGenerator ()

destructor

7.19.3 Member Function Documentation**7.19.3.1 CouenneCutGenerator* Couenne::CouenneCutGenerator::clone () const [inline]**

clone method (necessary for the abstract CglCutGenerator class)

Definition at line 125 of file CouenneCutGenerator.hpp.

7.19.3.2 CouenneProblem* Couenne::CouenneCutGenerator::Problem () const [inline]

return pointer to symbolic problem

Definition at line 129 of file CouenneCutGenerator.hpp.

7.19.3.3 void Couenne::CouenneCutGenerator::setProblem (CouenneProblem * p) [inline]

return pointer to symbolic problem

Definition at line 133 of file CouenneCutGenerator.hpp.

7.19.3.4 int Couenne::CouenneCutGenerator::getnvars () const

total number of variables (original + auxiliary)

7.19.3.5 bool Couenne::CouenneCutGenerator::isFirst () const [inline]

has generateCuts been called yet?

Definition at line 140 of file CouenneCutGenerator.hpp.

7.19.3.6 bool Couenne::CouenneCutGenerator::addViolated () const [inline]

should we add the violated cuts only (true), or all of them (false)?

Definition at line 144 of file CouenneCutGenerator.hpp.

7.19.3.7 enum conv_type Couenne::CouenneCutGenerator::ConvType () const [inline]

get convexification type (see CouenneTypes.h)

Definition at line 148 of file CouenneCutGenerator.hpp.

7.19.3.8 int Couenne::CouenneCutGenerator::nSamples () const [inline]

get number of convexification samples

Definition at line 152 of file CouenneCutGenerator.hpp.

7.19.3.9 void Couenne::CouenneCutGenerator::generateCuts (const OsiSolverInterface &, OsiCuts &, const CglTreeInfo = CglTreeInfo ()) const

the main CglCutGenerator

7.19.3.10 `int Couenne::CouenneCutGenerator::createCut(OsiCuts &, CouNumber, CouNumber, int, CouNumber, int = -1, CouNumber = 0., int = -1, CouNumber = 0., bool = false) const`

create cut and check violation. Insert and return status

7.19.3.11 `int Couenne::CouenneCutGenerator::createCut(OsiCuts &, CouNumber, int, int, CouNumber, int = -1, CouNumber = 0., int = -1, CouNumber = 0., bool = false) const`

create cut and check violation. Other version with only one bound

7.19.3.12 `void Couenne::CouenneCutGenerator::addEnvelope(OsiCuts &, int, unary_function, unary_function, int, int, CouNumber, CouNumber, CouNumber, t_chg_bounds * =NULL, bool = false) const`

Add general linear envelope to convex function, given its variables' indices, the (univariate) function and its first derivative.

7.19.3.13 `void Couenne::CouenneCutGenerator::addEnvelope(OsiCuts &, int, funtriplet *, int, int, CouNumber, CouNumber, CouNumber, t_chg_bounds * =NULL, bool = false) const`

Add general linear envelope to convex function, given its variables' indices, the (univariate) function and its first derivative.

7.19.3.14 `int Couenne::CouenneCutGenerator::addSegment(OsiCuts &, int, int, CouNumber, CouNumber, CouNumber, CouNumber, int) const`

Add half-plane through (x1,y1) and (x2,y2) – resp.

4th, 5th, 6th, and 7th argument

7.19.3.15 `int Couenne::CouenneCutGenerator::addTangent(OsiCuts &, int, int, CouNumber, CouNumber, CouNumber, CouNumber, int) const`

add tangent at given poing (x,w) with given slope

7.19.3.16 `void Couenne::CouenneCutGenerator::setBabPtr(Bonmin::Bab * p) [inline]`

Method to set the Bab pointer.

Definition at line 218 of file CouenneCutGenerator.hpp.

7.19.3.17 `void Couenne::CouenneCutGenerator::getStats(int & nrc, int & ntc, double & st) [inline]`

Get statistics.

Definition at line 222 of file CouenneCutGenerator.hpp.

7.19.3.18 `bool& Couenne::CouenneCutGenerator::infeasNode() const [inline]`

Allow to get and set the infeasNode_ flag (used only in [generateCuts\(\)](#))

Definition at line 229 of file CouenneCutGenerator.hpp.

7.19.3.19 `void Couenne::CouenneCutGenerator::genRowCuts(const OsiSolverInterface &, OsiCuts & cs, int, int *, t_chg_bounds * =NULL) const`

generate OsiRowCuts for current convexification

7.19.3.20 void Couenne::CouenneCutGenerator::genColCuts (const OsiSolverInterface & , OsiCuts & , int , int *) const
generate OsiColCuts for improved (implied and propagated) bounds

7.19.3.21 static void Couenne::CouenneCutGenerator::registerOptions (Iopt::SmartPtr< Bonmin::RegisteredOptions > *options*)
[static]

Add list of options to be read from file.

7.19.3.22 void Couenne::CouenneCutGenerator::printLineInfo () const

print node, depth, LB/UB/LP info

7.19.3.23 ConstJnlstPtr Couenne::CouenneCutGenerator::Jnlst () const [inline]

Provide Journalist.

Definition at line 246 of file CouenneCutGenerator.hpp.

7.19.3.24 void Couenne::CouenneCutGenerator::setJnlst (JnlstPtr *jnlst_*) [inline]

Definition at line 249 of file CouenneCutGenerator.hpp.

7.19.3.25 double& Couenne::CouenneCutGenerator::rootTime () [inline]

Time spent at root node.

Definition at line 253 of file CouenneCutGenerator.hpp.

7.19.3.26 bool Couenne::CouenneCutGenerator::check_lp () const [inline]

return check_lp flag (used in [CouenneSolverInterface](#))

Definition at line 257 of file CouenneCutGenerator.hpp.

7.19.3.27 bool Couenne::CouenneCutGenerator::enableLpImpliedBounds () const [inline]

returns value of enable_lp_implied_bounds_

Definition at line 261 of file CouenneCutGenerator.hpp.

7.19.4 Member Data Documentation

7.19.4.1 bool Couenne::CouenneCutGenerator::firstcall_ [mutable], [protected]

True if no convexification cuts have been generated yet for this problem.

Definition at line 55 of file CouenneCutGenerator.hpp.

7.19.4.2 bool Couenne::CouenneCutGenerator::addviolated_ [mutable], [protected]

True if we should add the violated cuts only, false if all of them should be added.

Definition at line 59 of file CouenneCutGenerator.hpp.

7.19.4.3 enum conv_type Couenne::CouenneCutGenerator::convtype_ [protected]

what kind of sampling should be performed?

Definition at line 62 of file CouenneCutGenerator.hpp.

7.19.4.4 `int Couenne::CouenneCutGenerator::nSamples_` [protected]

how many cuts should be added for each function?

Definition at line 65 of file CouenneCutGenerator.hpp.

7.19.4.5 `CouenneProblem* Couenne::CouenneCutGenerator::problem_` [protected]

pointer to symbolic repr. of constraint, variables, and bounds

Definition at line 68 of file CouenneCutGenerator.hpp.

7.19.4.6 `int Couenne::CouenneCutGenerator::nrootcuts_` [mutable], [protected]

number of cuts generated at the first call

Definition at line 71 of file CouenneCutGenerator.hpp.

7.19.4.7 `int Couenne::CouenneCutGenerator::ntotalcuts_` [mutable], [protected]

total number of cuts generated

Definition at line 74 of file CouenneCutGenerator.hpp.

7.19.4.8 `double Couenne::CouenneCutGenerator::septime_` [mutable], [protected]

separation time (includes generation of problem)

Definition at line 77 of file CouenneCutGenerator.hpp.

7.19.4.9 `double Couenne::CouenneCutGenerator::objValue_` [mutable], [protected]

Record obj value at final point of CouenneConv.

Definition at line 80 of file CouenneCutGenerator.hpp.

7.19.4.10 `Bonmin::OsiTMINLPInterface* Couenne::CouenneCutGenerator::nlp_` [protected]

nonlinear solver interface as used within `Bonmin` (used at first `Couenne` pass of each b&b node

Definition at line 84 of file CouenneCutGenerator.hpp.

7.19.4.11 `Bonmin::Bab* Couenne::CouenneCutGenerator::BabPtr_` [protected]

pointer to the Bab object (used to retrieve the current primal bound through bestObj())

Definition at line 88 of file CouenneCutGenerator.hpp.

7.19.4.12 `bool Couenne::CouenneCutGenerator::infeasNode_` [mutable], [protected]

signal infeasibility of current node (found through bound tightening)

Definition at line 91 of file CouenneCutGenerator.hpp.

7.19.4.13 `JnlstPtr Couenne::CouenneCutGenerator::jnlst_` [protected]

SmartPointer to the Journalist.

Definition at line 94 of file CouenneCutGenerator.hpp.

7.19.4.14 double Couenne::CouenneCutGenerator::rootTime_ [mutable], [protected]

Time spent at the root node.

Definition at line 97 of file CouenneCutGenerator.hpp.

7.19.4.15 bool Couenne::CouenneCutGenerator::check_lp_ [protected]

Check all generated LPs through an independent call to OsiClpSolverInterface::initialSolve()

Definition at line 101 of file CouenneCutGenerator.hpp.

7.19.4.16 `bool Couenne::CouenneCutGenerator::enable_lp_implied_bounds_` [protected]

Take advantage of `OsiClpSolverInterface::tightenBounds()`, known to have caused some problems some time ago.

Definition at line 105 of file CouenneCutGenerator.hpp.

7.19.4.17 `int Couenne::CouenneCutGenerator::lastPrintLine` [mutable], [protected]

Running count of printed info lines.

Definition at line 108 of file CouenneCutGenerator.hpp.

The documentation for this class was generated from the following file:

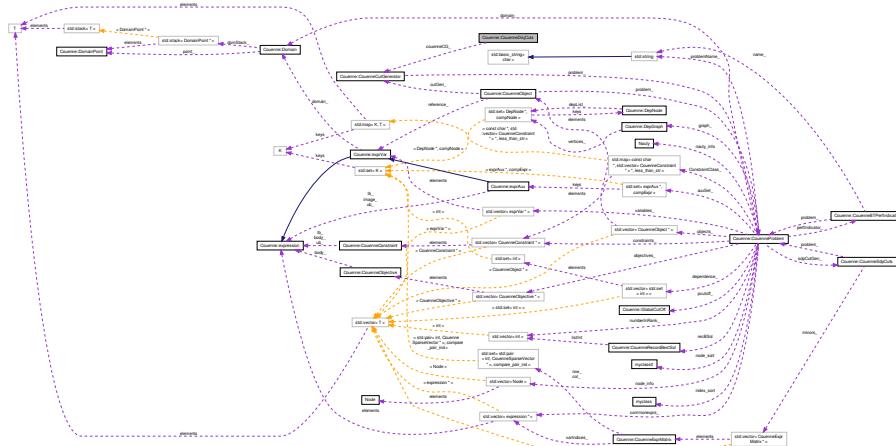
- `/home/ted/COIN/trunk/Couenne/src/convex/CouenneCutGenerator.hpp`

7.20 Couenne::CouenneDisjCuts Class Reference

Cut Generator for linear convexifications.

```
#include <CouenneDisjCuts.hpp>
```

Collaboration diagram for Couenne::CouenneDisjCuts:



Public Member Functions

- `CouenneDisjCuts` (`Bonmin::OsiTMINLPInterface *minlp=NULL`, `Bonmin::BabSetupBase *base=NULL`, `CouenneCutGenerator *cq=NULL`, `OsiChooseVariable *bcv=NULL`, `bool is_strong=false`, `JnlstPtr journalist=NULL`, `const`

- `Iopt::SmartPtr< Iopt::OptionsList > options=NULL)`
constructor
- `CouenneDisjCuts (const CouenneDisjCuts &)`
copy constructor
 - `~CouenneDisjCuts ()`
destructor
 - `CouenneDisjCuts * clone () const`
clone method (necessary for the abstract CglCutGenerator class)
 - `CouenneCutGenerator * couenneCG () const`
return pointer to symbolic problem
 - `void generateCuts (const OsiSolverInterface &, OsiCuts &, const CglTreeInfo=CglTreeInfo()) const`
the main CglCutGenerator
 - `ConstJnlstPtr Jnlst () const`
Provide Journalist.
 - `int getDisjunctions (std::vector< std::pair< OsiCuts *, OsiCuts * > > &disjunctions, OsiSolverInterface &si, OsiCuts &cs, const CglTreeInfo &info) const`
get all disjunctions
 - `int separateWithDisjunction (OsiCuts *cuts, OsiSolverInterface &si, OsiCuts &cs, const CglTreeInfo &info) const`
separate couenne cuts on both sides of single disjunction
 - `int generateDisjCuts (std::vector< std::pair< OsiCuts *, OsiCuts * > > &disjs, OsiSolverInterface &si, OsiCuts &cs, const CglTreeInfo &info) const`
generate one disjunctive cut from one CGLP
 - `int checkDisjSide (OsiSolverInterface &si, OsiCuts *cuts) const`
check if (column!) cuts compatible with solver interface
 - `int getBoxUnion (OsiSolverInterface &si, OsiCuts *left, OsiCuts *right, CoinPackedVector &lower, CoinPackedVector &upper) const`
compute smallest box containing both left and right boxes.

Static Public Member Functions

- `static void registerOptions (Iopt::SmartPtr< Bonmin::RegisteredOptions > roptions)`
Add list of options to be read from file.

Protected Member Functions

- `OsiCuts * getSingleDisjunction (OsiSolverInterface &si) const`
create single osicolcut disjunction
- `void mergeBoxes (int dir, CoinPackedVector &left, CoinPackedVector &right, CoinPackedVector merged) const`
utility to merge vectors into one
- `void applyColCuts (OsiSolverInterface &si, OsiCuts *cuts) const`
our own applyColCuts
- `void applyColCuts (OsiSolverInterface &si, OsiColCut *cut) const`
our own applyColCut, single cut
- `void OsiSI2MatrVec (CoinPackedMatrix &M, CoinPackedVector &r, OsiSolverInterface &si) const`
- `int OsiCuts2MatrVec (OsiSolverInterface *cglp, OsiCuts *cuts, int displRow, int displRhs) const`
add CGLP columns to solver interface; return number of columns added (for later removal)

Protected Attributes

- `CouenneCutGenerator * couenneCG_`
`pointer to symbolic repr. of constraint, variables, and bounds`
- `int nrootcuts_`
`number of cuts generated at the first call`
- `int ntotalcuts_`
`total number of cuts generated`
- `double septime_`
`separation time (includes generation of problem)`
- `double objValue_`
`Record obj value at final point of CouenneConv.`
- `Bonmin::OsiTMINLPInterface * minlp_`
`nonlinear solver interface as used within Bonmin (used at first Couenne pass of each b&b node)`
- `OsiChooseVariable * branchingMethod_`
`Branching scheme (if strong, we can use SB candidates)`
- `bool isBranchingStrong_`
`Is branchMethod_ referred to a strong branching scheme?`
- `JnlstPtr jnlst_`
`SmartPointer to the Journalist.`
- `int numDisjunctions_`
`Number of disjunction to consider at each separation.`
- `double initDisjPercentage_`
`Initial percentage of objects to use for generating cuts, in [0,1].`
- `int initDisjNumber_`
`Initial number of objects to use for generating cuts.`
- `int depthLevelling_`
`Depth of the BB tree where start decreasing number of objects.`
- `int depthStopSeparate_`
`Depth of the BB tree where stop separation.`
- `bool activeRows_`
`only include active rows in CGLP`
- `bool activeCols_`
`only include active columns in CGLP`
- `bool addPreviousCut_`
`add previous disj cut to current CGLP?`
- `double cpuTime_`
`maximum CPU time`

7.20.1 Detailed Description

Cut Generator for linear convexifications.

Definition at line 34 of file CouenneDisjCuts.hpp.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 `Couenne::CouenneDisjCuts::CouenneDisjCuts (Bonmin::OsiTMINLPInterface * minlp = NULL, Bonmin::BabSetupBase * base = NULL, CouenneCutGenerator * cg = NULL, OsiChooseVariable * bcv = NULL, bool is_strong = false, JnlstPtr journalist = NULL, const Ipopt::SmartPtr<Ipopt::OptionsList> options = NULL)`

constructor

7.20.2.2 `Couenne::CouenneDisjCuts::CouenneDisjCuts (const CouenneDisjCuts &)`

copy constructor

7.20.2.3 `Couenne::CouenneDisjCuts::~CouenneDisjCuts ()`

destructor

7.20.3 Member Function Documentation

7.20.3.1 `CouenneDisjCuts* Couenne::CouenneDisjCuts::clone () const [inline]`

clone method (necessary for the abstract CglCutGenerator class)

Definition at line 111 of file CouenneDisjCuts.hpp.

7.20.3.2 `CouenneCutGenerator* Couenne::CouenneDisjCuts::couenneCG () const [inline]`

return pointer to symbolic problem

Definition at line 115 of file CouenneDisjCuts.hpp.

7.20.3.3 `void Couenne::CouenneDisjCuts::generateCuts (const OsiSolverInterface &, OsiCuts &, const CglTreeInfo = CglTreeInfo ()) const`

the main CglCutGenerator

7.20.3.4 `static void Couenne::CouenneDisjCuts::registerOptions (Ipopt::SmartPtr<Bonmin::RegisteredOptions> options) [static]`

Add list of options to be read from file.

7.20.3.5 `ConstJnlstPtr Couenne::CouenneDisjCuts::Jnlst () const [inline]`

Provide Journalist.

Definition at line 131 of file CouenneDisjCuts.hpp.

7.20.3.6 `int Couenne::CouenneDisjCuts::getDisjunctions (std::vector<std::pair<OsiCuts *, OsiCuts *>> & disjunctions, OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo & info) const`

get all disjunctions

7.20.3.7 `int Couenne::CouenneDisjCuts::separateWithDisjunction (OsiCuts * cuts, OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo & info) const`

separate couenne cuts on both sides of single disjunction

7.20.3.8 int Couenne::CouenneDisjCuts::generateDisjCuts (std::vector< std::pair< OsiCuts *, OsiCuts * > > & *disjs*, OsiSolverInterface & *si*, OsiCuts & *cs*, const CglTreeInfo & *info*) const

generate one disjunctive cut from one CGLP

7.20.3.9 int Couenne::CouenneDisjCuts::checkDisjSide (OsiSolverInterface & *si*, OsiCuts * *cuts*) const

check if (column!) cuts compatible with solver interface

7.20.3.10 int Couenne::CouenneDisjCuts::getBoxUnion (OsiSolverInterface & *si*, OsiCuts * *left*, OsiCuts * *right*, CoinPackedVector & *lower*, CoinPackedVector & *upper*) const

compute smallest box containing both left and right boxes.

7.20.3.11 OsiCuts* Couenne::CouenneDisjCuts::getSingleDisjunction (OsiSolverInterface & *si*) const [protected]

create single osicolcut disjunction

7.20.3.12 void Couenne::CouenneDisjCuts::mergeBoxes (int *dir*, CoinPackedVector & *left*, CoinPackedVector & *right*, CoinPackedVector *merged*) const [protected]

utility to merge vectors into one

7.20.3.13 void Couenne::CouenneDisjCuts::applyColCuts (OsiSolverInterface & *si*, OsiCuts * *cuts*) const [protected]

our own applyColCuts

7.20.3.14 void Couenne::CouenneDisjCuts::applyColCut (OsiSolverInterface & *si*, OsiColCut * *cut*) const [protected]

our own applyColCut, single cut

7.20.3.15 void Couenne::CouenneDisjCuts::OsiSI2MatrVec (CoinPackedMatrix & *M*, CoinPackedVector & *r*, OsiSolverInterface & *si*) const [protected]

7.20.3.16 int Couenne::CouenneDisjCuts::OsiCuts2MatrVec (OsiSolverInterface * *cglp*, OsiCuts * *cuts*, int *displRow*, int *displRhs*) const [protected]

add CGLP columns to solver interface; return number of columns added (for later removal)

7.20.4 Member Data Documentation

7.20.4.1 CouenneCutGenerator* Couenne::CouenneDisjCuts::couenneCG_ [protected]

pointer to symbolic repr. of constraint, variables, and bounds

Definition at line 39 of file CouenneDisjCuts.hpp.

7.20.4.2 int Couenne::CouenneDisjCuts::nrootcuts_ [mutable], [protected]

number of cuts generated at the first call

Definition at line 42 of file CouenneDisjCuts.hpp.

7.20.4.3 int Couenne::CouenneDisjCuts::ntotalcuts_ [mutable], [protected]

total number of cuts generated

Definition at line 45 of file CouenneDisjCuts.hpp.

7.20.4.4 double Couenne::CouenneDisjCuts::septime_ [mutable], [protected]

separation time (includes generation of problem)

Definition at line 48 of file CouenneDisjCuts.hpp.

7.20.4.5 double Couenne::CouenneDisjCuts::objValue_ [mutable], [protected]

Record obj value at final point of CouenneConv.

Definition at line 51 of file CouenneDisjCuts.hpp.

7.20.4.6 Bonmin::OsiTMINLPInterface* Couenne::CouenneDisjCuts::minlp_ [protected]

nonlinear solver interface as used within Bonmin (used at first Couenne pass of each b&b node)

Definition at line 55 of file CouenneDisjCuts.hpp.

7.20.4.7 OsiChooseVariable* Couenne::CouenneDisjCuts::branchingMethod_ [protected]

Branching scheme (if strong, we can use SB candidates)

Definition at line 58 of file CouenneDisjCuts.hpp.

7.20.4.8 bool Couenne::CouenneDisjCuts::isBranchingStrong_ [protected]

Is branchMethod_ referred to a strong branching scheme?

Definition at line 61 of file CouenneDisjCuts.hpp.

7.20.4.9 JnlstPtr Couenne::CouenneDisjCuts::jnlst_ [protected]

SmartPointer to the Journalist.

Definition at line 64 of file CouenneDisjCuts.hpp.

7.20.4.10 int Couenne::CouenneDisjCuts::numDisjunctions_ [mutable], [protected]

Number of disjunction to consider at each separation.

Definition at line 67 of file CouenneDisjCuts.hpp.

7.20.4.11 double Couenne::CouenneDisjCuts::initDisjPercentage_ [protected]

Initial percentage of objects to use for generating cuts, in [0,1].

Definition at line 70 of file CouenneDisjCuts.hpp.

7.20.4.12 int Couenne::CouenneDisjCuts::initDisjNumber_ [protected]

Initial number of objects to use for generating cuts.

Definition at line 73 of file CouenneDisjCuts.hpp.

7.20.4.13 int Couenne::CouenneDisjCuts::depthLevelling_ [protected]

Depth of the BB tree where start decreasing number of objects.

Definition at line 76 of file CouenneDisjCuts.hpp.

7.20.4.14 int Couenne::CouenneDisjCuts::depthStopSeparate_ [protected]

Depth of the BB tree where stop separation.

Definition at line 79 of file CouenneDisjCuts.hpp.

7.20.4.15 bool Couenne::CouenneDisjCuts::activeRows_ [protected]

only include active rows in CGLP

Definition at line 82 of file CouenneDisjCuts.hpp.

7.20.4.16 bool Couenne::CouenneDisjCuts::activeCols_ [protected]

only include active columns in CGLP

Definition at line 85 of file CouenneDisjCuts.hpp.

7.20.4.17 bool Couenne::CouenneDisjCuts::addPreviousCut_ [protected]

add previous disj cut to current CGLP?

Definition at line 88 of file CouenneDisjCuts.hpp.

7.20.4.18 double Couenne::CouenneDisjCuts::cpuTime_ [protected]

maximum CPU time

Definition at line 91 of file CouenneDisjCuts.hpp.

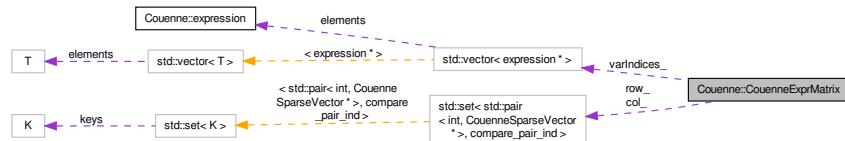
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/disjunctive/CouenneDisjCuts.hpp

7.21 Couenne::CouenneExprMatrix Class Reference

```
#include <CouenneMatrix.hpp>
```

Collaboration diagram for Couenne::CouenneExprMatrix:



Classes

- struct [compare_pair_ind](#)

Public Member Functions

- [CouenneExprMatrix \(\)](#)

- `~CouenneExprMatrix ()`
- `CouenneExprMatrix (const CouenneExprMatrix &rhs)`
- `CouenneExprMatrix & operator= (const CouenneExprMatrix &rhs)`
- `CouenneExprMatrix * clone ()`
- `const std::set< std::pair< int, CouenneSparseVector * >, compare_pair_ind > & getRows () const`
- `const std::set< std::pair< int, CouenneSparseVector * >, compare_pair_ind > & getCols () const`
- `std::vector< expression * > & varIndices ()`
- `void add_element (int row, int column, expression *elem)`
- `void print () const`
- `long unsigned int size ()`
- `CouenneSparseVector & operator* (const CouenneSparseVector &factor) const
matrix * vector`
- `CouenneExprMatrix & operator* (const CouenneExprMatrix &post) const
matrix * matrix`

Protected Attributes

- `std::set< std::pair< int, CouenneSparseVector * >, compare_pair_ind > row_
row major`
- `std::set< std::pair< int, CouenneSparseVector * >, compare_pair_ind > col_
col major`
- `std::vector< expression * > varIndices_
if used in sdp cuts, contains indices of x_i used in $X_{ij} = x_i * x_j$`

7.21.1 Detailed Description

Definition at line 104 of file CouenneMatrix.hpp.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 Couenne::CouenneExprMatrix::CouenneExprMatrix () [inline]

Definition at line 123 of file CouenneMatrix.hpp.

7.21.2.2 Couenne::CouenneExprMatrix::~CouenneExprMatrix ()

7.21.2.3 Couenne::CouenneExprMatrix::CouenneExprMatrix (const CouenneExprMatrix & rhs)

7.21.3 Member Function Documentation

7.21.3.1 CouenneExprMatrix& Couenne::CouenneExprMatrix::operator= (const CouenneExprMatrix & rhs)

7.21.3.2 CouenneExprMatrix* Couenne::CouenneExprMatrix::clone() [inline]

Definition at line 129 of file CouenneMatrix.hpp.

7.21.3.3 const std::set<std::pair <int, CouenneSparseVector *>, compare_pair_ind>& Couenne::CouenneExprMatrix::getRows() const [inline]

Definition at line 131 of file CouenneMatrix.hpp.

7.21.3.4 const std::set<std::pair <int, CouenneSparseVector *>, compare_pair_ind>& Couenne::CouenneExprMatrix::getCols() const [inline]

Definition at line 132 of file CouenneMatrix.hpp.

7.21.3.5 std::vector<expression *>& Couenne::CouenneExprMatrix::varIndices() [inline]

Definition at line 134 of file CouenneMatrix.hpp.

7.21.3.6 void Couenne::CouenneExprMatrix::add_element(int row, int column, expression * elem)

7.21.3.7 void Couenne::CouenneExprMatrix::print() const

7.21.3.8 long unsigned int Couenne::CouenneExprMatrix::size()

7.21.3.9 CouenneSparseVector& Couenne::CouenneExprMatrix::operator*(const CouenneSparseVector & factor) const

matrix * vector

7.21.3.10 CouenneExprMatrix& Couenne::CouenneExprMatrix::operator*(const CouenneExprMatrix & post) const

matrix * matrix

7.21.4 Member Data Documentation

7.21.4.1 std::set<std::pair <int, CouenneSparseVector *>, compare_pair_ind> Couenne::CouenneExprMatrix::row_ [protected]

row major

Definition at line 116 of file CouenneMatrix.hpp.

7.21.4.2 std::set<std::pair <int, CouenneSparseVector *>, compare_pair_ind> Couenne::CouenneExprMatrix::col_ [protected]

col major

Definition at line 117 of file CouenneMatrix.hpp.

7.21.4.3 std::vector<expression *> Couenne::CouenneExprMatrix::varIndices_ [protected]

if used in sdp cuts, contains indices of x_i used in $X_{ij} = x_i * x_j$

Definition at line 119 of file CouenneMatrix.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneMatrix.hpp

7.22 Couenne::CouenneFeasPump Class Reference

An implementation of the Feasibility pump that uses linearization and `Iopt` to find the two sequences of points.

```
#include <CouenneFeasPump.hpp>
```

Public Types

- enum `fpCompDistIntType` { `FP_DIST_INT`, `FP_DIST_ALL`, `FP_DIST_POST` }
- enum `fpCutPlane` { `FP_CUT_NONE`, `FP_CUT_INTEGRATED`, `FP_CUT_EXTERNAL`, `FP_CUT_POST` }
- enum `fpTabuMgtPolicy` { `FP_TABU_NONE`, `FP_TABU_POOL`, `FP_TABU_PERTURB`, `FP_TABU_CUT` }

Public Member Functions

- `CouenneFeasPump (CouenneProblem *couenne=NULL, CouenneCutGenerator *cg=NULL, Iopt::SmartPtr<Iopt::OptionsList> options=NULL)`

Constructor with (optional) MINLP pointer.
- `CouenneFeasPump (const CouenneFeasPump &other)`

Copy constructor.
- `virtual ~CouenneFeasPump ()`

Destructor.
- `virtual CbcHeuristic * clone () const`

Clone.
- `CouenneFeasPump & operator= (const CouenneFeasPump &rhs)`

Assignment operator.
- `virtual void resetModel (CbcModel *model)`

Does nothing, but necessary as CbcHeuristic declares it pure virtual.
- `virtual int solution (double &objectiveValue, double *newSolution)`

Run heuristic, return 1 if a better solution than the one passed is found and 0 otherwise.
- `void setNumberSolvePerLevel (int value)`

set number of nlp's solved for each given level of the tree
- `virtual CouNumber solveMILP (const CouNumber *nSol, CouNumber *&iSol, int niter, int *nsuciter)`

find integer (possibly NLP-infeasible) point isol closest (according to the l-1 norm of the hessian) to the current NLP-feasible (but fractional) solution nsol
- `virtual CouNumber solveNLP (const CouNumber *nSol, CouNumber *&iSol)`

obtain solution to NLP
- `expression * updateNLPObj (const double *)`

set new expression as the NLP objective function using argument as point to minimize distance from.
- `bool fixIntVariables (const double *sol)`

admits a (possibly fractional) solution and fixes the integer components in the nonlinear problem for later re-solve.
- `double findSolution (const double *nSol, double *&sol, int niter, int *nsuciter)`

find feasible solution (called by solveMILP ())
- `void init_MILP ()`

initialize all solvers at the first call, where the initial MILP is built
- `void initIoptApp ()`

Common code for initializing non-smartptr ipopt application.
- `CouenneProblem * Problem () const`

return pointer to problem

- enum [fpCompDistIntType compDistInt \(\) const](#)
return type of MILP solved
- double [multDistNLP \(\) const](#)
Return Weights in computing distance, in both MILP and NLP (must sum up to 1 for MILP and for NLP):
- double [multHessNLP \(\) const](#)
weight of Hessian in NLP
- double [multObjFNLP \(\) const](#)
weight of objective in NLP
- double [multDistMILP \(\) const](#)
weight of distance in MILP
- double [multHessMILP \(\) const](#)
weight of Hessian in MILP
- double [multObjFMILP \(\) const](#)
weight of objective in MILP
- [CouenneTNLP * nlp \(\) const](#)
return NLP
- int & [nCalls \(\)](#)
return number of calls (can be changed)
- int [milpPhase \(double *nSol, double *iSol\)](#)
MILP phase of the FP.
- int [nlpPhase \(double *iSol, double *nSol\)](#)
NLP phase of the FP.

Static Public Member Functions

- static void [registerOptions \(Ipopt::SmartPtr< Bonmin::RegisteredOptions >\)](#)
initialize options to be read later

7.22.1 Detailed Description

An implementation of the Feasibility pump that uses linearization and [Ipopt](#) to find the two sequences of points.

Definition at line 57 of file CouenneFeasPump.hpp.

7.22.2 Member Enumeration Documentation

7.22.2.1 enum Couenne::CouenneFeasPump::fpCompDistIntType

Enumerator:

FP_DIST_INT
FP_DIST_ALL
FP_DIST_POST

Definition at line 61 of file CouenneFeasPump.hpp.

7.22.2.2 enum Couenne::CouenneFeasPump::fpCutPlane

Enumerator:

FP_CUT_NONE
FP_CUT_INTEGRATED
FP_CUT_EXTERNAL
FP_CUT_POST

Definition at line 62 of file CouenneFeasPump.hpp.

7.22.2.3 enum Couenne::CouenneFeasPump::fpTabuMgtPolicy

Enumerator:

FP_TABU_NONE
FP_TABU_POOL
FP_TABU_PERTURB
FP_TABU_CUT

Definition at line 63 of file CouenneFeasPump.hpp.

7.22.3 Constructor & Destructor Documentation

7.22.3.1 Couenne::CouenneFeasPump::CouenneFeasPump (CouenneProblem * *couenne* = NULL, CouenneCutGenerator * *cg* = NULL, Ipopt::SmartPtr<Ipopt::OptionsList> *options* = NULL)

Constructor with (optional) MINLP pointer.

7.22.3.2 Couenne::CouenneFeasPump::CouenneFeasPump (const CouenneFeasPump & *other*)

Copy constructor.

7.22.3.3 virtual Couenne::CouenneFeasPump::~CouenneFeasPump () [virtual]

Destructor.

7.22.4 Member Function Documentation

7.22.4.1 virtual CbcHeuristic* Couenne::CouenneFeasPump::clone () const [virtual]

Clone.

7.22.4.2 CouenneFeasPump& Couenne::CouenneFeasPump::operator= (const CouenneFeasPump & *rhs*)

Assignment operator.

7.22.4.3 virtual void Couenne::CouenneFeasPump::resetModel (CbcModel * *model*) [inline], [virtual]

Does nothing, but necessary as CbcHeuristic declares it pure virtual.

Definition at line 83 of file CouenneFeasPump.hpp.

7.22.4.4 virtual int Couenne::CouenneFeasPump::solution (double & *objectiveValue*, double * *newSolution*) [virtual]

Run heuristic, return 1 if a better solution than the one passed is found and 0 otherwise.

objectiveValue Best known solution in input and value of solution found in output

newSolution Solution found by heuristic.

7.22.4.5 void Couenne::CouenneFeasPump::setNumberSolvePerLevel (int *value*) [inline]

set number of nlp's solved for each given level of the tree

Definition at line 95 of file CouenneFeasPump.hpp.

7.22.4.6 virtual CouNumber Couenne::CouenneFeasPump::solveMILP (const CouNumber * *nSol*, CouNumber *& *iSol*, int *niter*, int * *nsuciter*) [virtual]

find integer (possibly NLP-infeasible) point isol closest (according to the l-1 norm of the hessian) to the current NLP-feasible (but fractional) solution nsol

7.22.4.7 virtual CouNumber Couenne::CouenneFeasPump::solveNLP (const CouNumber * *nSol*, CouNumber *& *iSol*) [virtual]

obtain solution to NLP

7.22.4.8 expression* Couenne::CouenneFeasPump::updateNLPObj (const double *)

set new expression as the NLP objective function using argument as point to minimize distance from.

Return new objective function

7.22.4.9 bool Couenne::CouenneFeasPump::fixIntVariables (const double * *sol*)

admits a (possibly fractional) solution and fixes the integer components in the nonlinear problem for later re-solve.

Returns false if restriction infeasible, true otherwise

7.22.4.10 static void Couenne::CouenneFeasPump::registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions >) [static]

initialize options to be read later

7.22.4.11 double Couenne::CouenneFeasPump::findSolution (const double * *nSol*, double *& *sol*, int *niter*, int * *nsuciter*)

find feasible solution (called by solveMILP ())

7.22.4.12 void Couenne::CouenneFeasPump::init_MILP ()

initialize all solvers at the first call, where the initial MILP is built

7.22.4.13 void Couenne::CouenneFeasPump::initIpoptApp ()

Common code for initializing non-smartptr ipopt application.

7.22.4.14 CouenneProblem* Couenne::CouenneFeasPump::Problem () const [inline]

return pointer to problem

Definition at line 135 of file CouenneFeasPump.hpp.

7.22.4.15 enum fpCompDistIntType Couenne::CouenneFeasPump::compDistInt() const [inline]

return type of MILP solved

Definition at line 139 of file CouenneFeasPump.hpp.

7.22.4.16 double Couenne::CouenneFeasPump::multDistNLP() const [inline]

Return Weights in computing distance, in both MILP and NLP (must sum up to 1 for MILP and for NLP):

weight of distance in NLP

Definition at line 145 of file CouenneFeasPump.hpp.

7.22.4.17 double Couenne::CouenneFeasPump::multHessNLP() const [inline]

weight of Hessian in NLP

Definition at line 146 of file CouenneFeasPump.hpp.

7.22.4.18 double Couenne::CouenneFeasPump::multObjFNLP() const [inline]

weight of objective in NLP

Definition at line 147 of file CouenneFeasPump.hpp.

7.22.4.19 double Couenne::CouenneFeasPump::multDistMILP() const [inline]

weight of distance in MILP

Definition at line 149 of file CouenneFeasPump.hpp.

7.22.4.20 double Couenne::CouenneFeasPump::multHessMILP() const [inline]

weight of Hessian in MILP

Definition at line 150 of file CouenneFeasPump.hpp.

7.22.4.21 double Couenne::CouenneFeasPump::multObjFMILP() const [inline]

weight of objective in MILP

Definition at line 151 of file CouenneFeasPump.hpp.

7.22.4.22 CouenneTNLP* Couenne::CouenneFeasPump::nlp() const [inline]

return NLP

Definition at line 154 of file CouenneFeasPump.hpp.

7.22.4.23 int& Couenne::CouenneFeasPump::nCalls() [inline]

return number of calls (can be changed)

Definition at line 158 of file CouenneFeasPump.hpp.

7.22.4.24 int Couenne::CouenneFeasPump::milpPhase(double * nSol, double * iSol)

MILP phase of the FP.

7.22.4.25 `int Couenne::CouenneFeasPump::nlPPhase(double * iSol, double * nSol)`

NLP phase of the FP.

The documentation for this class was generated from the following file:

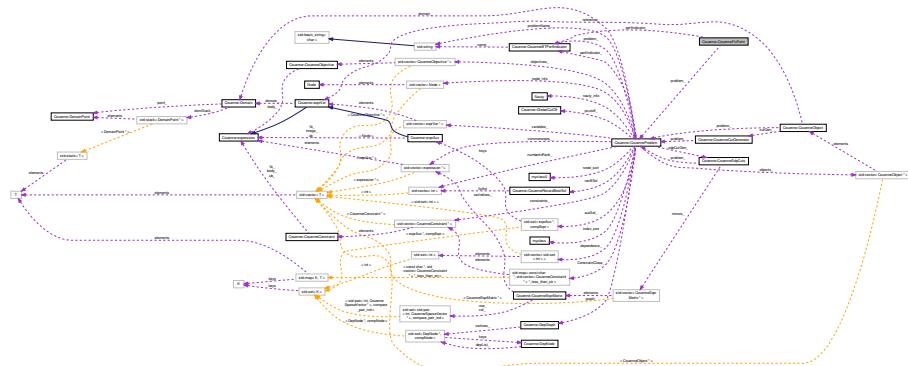
- /home/ted/COIN/trunk/Couenne/src/heuristics/[CouenneFeasPump.hpp](#)

7.23 Couenne::CouenneFixPoint Class Reference

Cut Generator for FBBT fixpoint.

```
#include <CouenneFixPoint.hpp>
```

Collaboration diagram for Couenne::CouenneFixPoint:



Public Member Functions

- **CouenneFixPoint** (*CouenneProblem* *, const *Iopt::SmartPtr< Iopt::OptionsList >*)
constructor
 - **CouenneFixPoint** (const **CouenneFixPoint** &)
copy constructor
 - **~CouenneFixPoint** ()
destructor
 - **CouenneFixPoint** * **clone** () const
clone method (necessary for the abstract CglCutGenerator class)
 - void **generateCuts** (const *OsiSolverInterface* &, *OsiCuts* &, const *CglTreeInfo*=*CglTreeInfo()*) const
the main CglCutGenerator

Static Public Member Functions

- static void `registerOptions` (Ipopt::SmartPtr< Bonmin::RegisteredOptions > options)
Add list of options to be read from file.

Protected Member Functions

- void [createRow](#) (int, int, int, OsiSolverInterface *, const int *, const double *, const double, const int, bool, int, int)
const
Create a single cut.

Protected Attributes

- bool [extendedModel_](#)
should we use an extended model or a more compact one?
- [CouenneProblem * problem_](#)
pointer to the CouenneProblem representation
- bool [firstCall_](#)
Is this the first call?
- double [CPUtime_](#)
CPU time.
- int [nTightened_](#)
Number of bounds tightened.
- [CouenneBTPerfIndicator perfIndicator_](#)
Performance indicator.

7.23.1 Detailed Description

Cut Generator for FBBT fixpoint.

Definition at line 30 of file CouenneFixPoint.hpp.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 Couenne::CouenneFixPoint::CouenneFixPoint (CouenneProblem *, const Ipopt::SmartPtr<Ipopt::OptionsList>)

constructor

7.23.2.2 Couenne::CouenneFixPoint::CouenneFixPoint (const CouenneFixPoint &)

copy constructor

7.23.2.3 Couenne::CouenneFixPoint::~CouenneFixPoint ()

destructor

7.23.3 Member Function Documentation

7.23.3.1 CouenneFixPoint* Couenne::CouenneFixPoint::clone () const [inline]

clone method (necessary for the abstract CglCutGenerator class)

Definition at line 45 of file CouenneFixPoint.hpp.

7.23.3.2 void Couenne::CouenneFixPoint::generateCuts (const OsiSolverInterface & , OsiCuts & , const CglTreeInfo = CglTreeInfo ()) const

the main CglCutGenerator

7.23.3.3 static void Couenne::CouenneFixPoint::registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > *options*) [static]

Add list of options to be read from file.

7.23.3.4 void Couenne::CouenneFixPoint::createRow (int , int , int , OsiSolverInterface * , const int * , const double * , const double , const int , bool , int , int) const [protected]

Create a single cut.

7.23.4 Member Data Documentation

7.23.4.1 bool Couenne::CouenneFixPoint::extendedModel_ [protected]

should we use an extended model or a more compact one?

Definition at line 63 of file CouenneFixPoint.hpp.

7.23.4.2 CouenneProblem* Couenne::CouenneFixPoint::problem_ [protected]

pointer to the [CouenneProblem](#) representation

Definition at line 66 of file CouenneFixPoint.hpp.

7.23.4.3 bool Couenne::CouenneFixPoint::firstCall_ [mutable], [protected]

Is this the first call?

Definition at line 69 of file CouenneFixPoint.hpp.

7.23.4.4 double Couenne::CouenneFixPoint::CPUtime_ [mutable], [protected]

CPU time.

Definition at line 72 of file CouenneFixPoint.hpp.

7.23.4.5 int Couenne::CouenneFixPoint::nTightened_ [mutable], [protected]

Number of bounds tightened.

Definition at line 75 of file CouenneFixPoint.hpp.

7.23.4.6 CouenneBTPerIndicator Couenne::CouenneFixPoint::perIndicator_ [protected]

Performance indicator.

Definition at line 89 of file CouenneFixPoint.hpp.

The documentation for this class was generated from the following file:

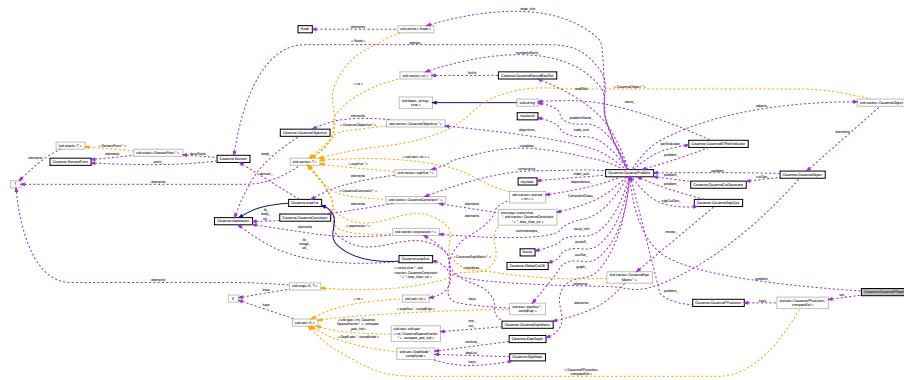
- /home/ted/COIN/trunk/Couenne/src/bound_tightening/[CouenneFixPoint.hpp](#)

7.24 Couenne::CouenneFPpool Class Reference

Pool of solutions.

```
#include <CouenneFPpool.hpp>
```

Collaboration diagram for Couenne::CouenneFPool:



Public Member Functions

- `CouenneFPpool` (`CouenneProblem` *`p`, enum `what_to_compare` `c`)
simple constructor (empty pool)
 - `CouenneFPpool` (`const CouenneFPpool` &`src`)
copy constructor
 - `CouenneFPpool & operator=` (`const CouenneFPpool` &`src`)
assignment
 - `std::set< CouenneFPsolution,`
`compareSol > & Set ()`
return the main object in this class
 - `CouenneProblem * Problem ()`
return the problem pointer
 - `void findClosestAndReplace` (`double * &sol`, `const double * nSol`, `int nvars`)
finds, in pool, solution x closest to sol; removes it from the pool and overwrites it to sol

Protected Attributes

- std::set< CouenneFPSolution,
 compareSol > set_
 Pool.
 - CouenneProblem * problem_
 Problem pointer.

7.24.1 Detailed Description

Pool of solutions.

Definition at line 91 of file CouenneFPpool.hpp.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 **Couenne::CouenneFPpool::CouenneFPpool (CouenneProblem * *p*, enum what_to_compare *c*)** [inline]

simple constructor (empty pool)

Definition at line 104 of file CouenneFPpool.hpp.

7.24.2.2 **Couenne::CouenneFPpool::CouenneFPpool (const CouenneFPpool & *src*)**

copy constructor

7.24.3 Member Function Documentation

7.24.3.1 **CouenneFPpool& Couenne::CouenneFPpool::operator= (const CouenneFPpool & *src*)**

assignment

7.24.3.2 **std::set<CouenneFPsolution, compareSol>& Couenne::CouenneFPpool::Set ()** [inline]

return the main object in this class

Definition at line 114 of file CouenneFPpool.hpp.

7.24.3.3 **CouenneProblem* Couenne::CouenneFPpool::Problem ()** [inline]

return the problem pointer

Definition at line 118 of file CouenneFPpool.hpp.

7.24.3.4 **void Couenne::CouenneFPpool::findClosestAndReplace (double *& *sol*, const double * *nSol*, int *nvars*)**

finds, in pool, solution x closest to sol; removes it from the pool and overwrites it to sol

7.24.4 Member Data Documentation

7.24.4.1 **std::set<CouenneFPsolution, compareSol> Couenne::CouenneFPpool::set_** [protected]

Pool.

Definition at line 96 of file CouenneFPpool.hpp.

7.24.4.2 **CouenneProblem* Couenne::CouenneFPpool::problem_** [protected]

Problem pointer.

Definition at line 99 of file CouenneFPpool.hpp.

The documentation for this class was generated from the following file:

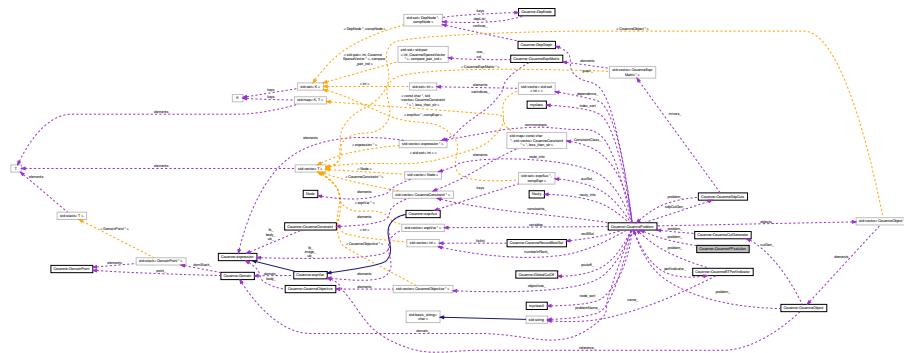
- /home/ted/COIN/trunk/Couenne/src/heuristics/CouenneFPpool.hpp

7.25 Couenne::CouenneFPsolution Class Reference

Class containing a solution with infeasibility evaluation.

```
#include <CouenneFPpool.hpp>
```

Collaboration diagram for Couenne::CouenneFPSolution:



Public Member Functions

- **CouenneFPSolution (CouenneProblem *p, CouNumber **x, bool copied=false)**
CouenneProblem-aware constructor.
- **CouenneFPSolution (const CouenneFPSolution &src)**
copy constructor
- **CouenneFPSolution & operator= (const CouenneFPSolution &src)**
assignment
- **~CouenneFPSolution ()**
destructor
- **const int n () const**
returns size
- **const double * x () const**
returns vector
- **bool compare (const CouenneFPSolution &other, enum what_to_compare comparedTerm) const**
basic comparison procedure – what to compare depends on user's choice

Protected Attributes

- **CouNumber * x_**
solution
- **int n_**
number of variables (for independence from CouenneProblem)
- **int nNLinf_**
number of NL infeasibilities
- **int nlinf_**
number of integer infeasibilities
- **CouNumber objVal_**
objective function value
- **CouNumber maxNLinf_**
maximum NL infeasibility
- **CouNumber maxlinf_**
maximum integer infeasibility

- bool `copied_`
This is a temporary copy, not really a solution holder.
- `CouenneProblem * problem_`
holds pointer to problem to check integrality in comparison of integer variables

7.25.1 Detailed Description

Class containing a solution with infeasibility evaluation.

Definition at line 32 of file CouenneFPpool.hpp.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 `Couenne::CouenneFPsolution::CouenneFPsolution(CouenneProblem * p, CouNumber * x, bool copied = false)`

CouenneProblem-aware constructor.

7.25.2.2 `Couenne::CouenneFPsolution::CouenneFPsolution(const CouenneFPsolution & src)`

copy constructor

7.25.2.3 `Couenne::CouenneFPsolution::~CouenneFPsolution()`

destructor

7.25.3 Member Function Documentation

7.25.3.1 `CouenneFPsolution& Couenne::CouenneFPsolution::operator=(const CouenneFPsolution & src)`

assignment

7.25.3.2 `const int Couenne::CouenneFPsolution::n() const [inline]`

returns size

Definition at line 65 of file CouenneFPpool.hpp.

7.25.3.3 `const double* Couenne::CouenneFPsolution::x() const [inline]`

returns vector

Definition at line 68 of file CouenneFPpool.hpp.

7.25.3.4 `bool Couenne::CouenneFPsolution::compare(const CouenneFPsolution & other, enum what_to_compare comparedTerm) const`

basic comparison procedure – what to compare depends on user's choice

7.25.4 Member Data Documentation

7.25.4.1 `CouNumber* Couenne::CouenneFPsolution::x_ [protected]`

solution

Definition at line 36 of file CouenneFPpool.hpp.

7.25.4.2 int Couenne::CouenneFPSolution::n_ [protected]

number of variables (for independence from [CouenneProblem](#))

Definition at line 37 of file CouenneFPpool.hpp.

7.25.4.3 int Couenne::CouenneFPSolution::nNLinf_ [protected]

number of NL infeasibilities

Definition at line 38 of file CouenneFPpool.hpp.

7.25.4.4 int Couenne::CouenneFPSolution::nlinf_ [protected]

number of integer infeasibilities

Definition at line 39 of file CouenneFPpool.hpp.

7.25.4.5 CouNumber Couenne::CouenneFPSolution::objVal_ [protected]

objective function value

Definition at line 40 of file CouenneFPpool.hpp.

7.25.4.6 CouNumber Couenne::CouenneFPSolution::maxNLinf_ [protected]

maximum NL infeasibility

Definition at line 41 of file CouenneFPpool.hpp.

7.25.4.7 CouNumber Couenne::CouenneFPSolution::maxlinf_ [protected]

maximum integer infeasibility

Definition at line 42 of file CouenneFPpool.hpp.

7.25.4.8 bool Couenne::CouenneFPSolution::copied_ [protected]

This is a temporary copy, not really a solution holder.

As a result, all the above members are meaningless for copied solutions

Definition at line 48 of file CouenneFPpool.hpp.

7.25.4.9 CouenneProblem* Couenne::CouenneFPSolution::problem_ [protected]

holds pointer to problem to check integrality in comparison of integer variables

Definition at line 50 of file CouenneFPpool.hpp.

The documentation for this class was generated from the following file:

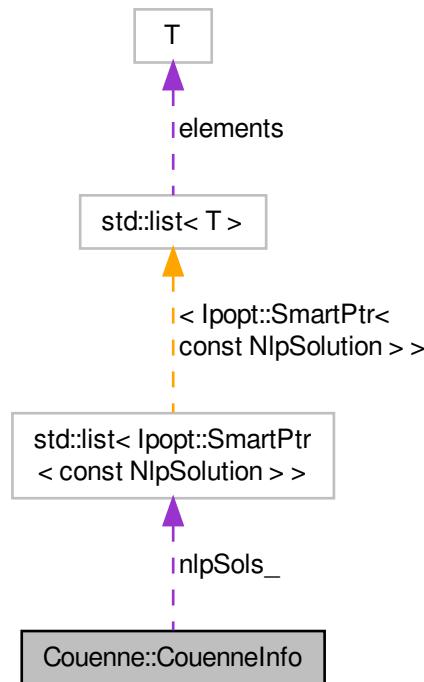
- /home/ted/COIN/trunk/Couenne/src/heuristics/[CouenneFPpool.hpp](#)

7.26 Couenne::CouenneInfo Class Reference

[Bonmin](#) class for passing info between components of branch-and-cuts.

```
#include <BonCouenneInfo.hpp>
```

Collaboration diagram for Couenne::CouenneInfo:



Classes

- class [NlpSolution](#)
Class for storing an Nlp Solution.

Public Member Functions

- [CouenneInfo](#) (int type)
Default constructor.
- [CouenneInfo](#) (const OsiBabSolver &other)
Constructor from OsiBabSolver.
- [CouenneInfo](#) (const [CouenneInfo](#) &other)
Copy constructor.
- virtual [~CouenneInfo](#) ()
Destructor.
- virtual OsiAuxInfo * [clone](#) () const
Virtual copy constructor.
- const std::list< Ipopt::SmartPtr< const NlpSolution > > & [NlpSolutions](#) () const

List of all stored NLP solutions.

- void `addSolution (Ipopt::SmartPtr< const NlpSolution > newSol)`
Add a new NLP solution.

Protected Attributes

- std::list< Ipopt::SmartPtr< const NlpSolution > > `nlpSols_`

7.26.1 Detailed Description

`Bonmin` class for passing info between components of branch-and-cuts.

Definition at line 22 of file BonCouenneInfo.hpp.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 Couenne::CouenneInfo::CouenneInfo (int type)

Default constructor.

7.26.2.2 Couenne::CouenneInfo::CouenneInfo (const OsiBabSolver & other)

Constructor from OsiBabSolver.

7.26.2.3 Couenne::CouenneInfo::CouenneInfo (const CouenneInfo & other)

Copy constructor.

7.26.2.4 virtual Couenne::CouenneInfo::~CouenneInfo () [virtual]

Destructor.

7.26.3 Member Function Documentation

7.26.3.1 virtual OsiAuxInfo* Couenne::CouenneInfo::clone () const [virtual]

Virtual copy constructor.

7.26.3.2 const std::list<Ipopt::SmartPtr<const NlpSolution>>& Couenne::CouenneInfo::NlpSolutions () const [inline]

List of all stored NLP solutions.

Definition at line 81 of file BonCouenneInfo.hpp.

7.26.3.3 void Couenne::CouenneInfo::addSolution (Ipopt::SmartPtr< const NlpSolution > newSol) [inline]

Add a new NLP solution.

Definition at line 86 of file BonCouenneInfo.hpp.

7.26.4 Member Data Documentation

7.26.4.1 `std::list<Ipopt::SmartPtr<const NlpSolution>> Couenne::CouenneInfo::nlpSols_` [protected]

Definition at line 92 of file BonCouenneInfo.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/main/BonCouenneInfo.hpp

7.27 Couenne::CouenneInterface Class Reference

```
#include <BonCouenneInterface.hpp>
```

Public Member Functions

- [CouenneInterface \(\)](#)
Default constructor.
- [CouenneInterface \(const CouenneInterface &other\)](#)
Copy constructor.
- virtual [CouenneInterface * clone \(bool CopyData\)](#)
virutal copy constructor.
- virtual [~CouenneInterface \(\)](#)
Destructor.
- virtual [std::string appName \(\)](#)

Overloaded methods to build outer approximations

- bool [have_nlp_solution_](#)
true if we got an integer feasible solution from initial solve
- virtual void [extractLinearRelaxation \(OsiSolverInterface &si, CouenneCutGenerator &couenneCg, bool getObj=1, bool solveNlp=1\)](#)
Extract a linear relaxation of the MINLP.
- virtual void [setAppDefaultOptions \(Ipopt::SmartPtr< Ipopt::OptionsList > Options\)](#)
To set some application specific defaults.
- bool [haveNlpSolution \(\)](#)
return value of have_nlp_solution_

7.27.1 Detailed Description

Definition at line 33 of file BonCouenneInterface.hpp.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 [Couenne::CouenneInterface::CouenneInterface \(\)](#)

Default constructor.

7.27.2.2 Couenne::CouenneInterface::CouenneInterface (const CouenneInterface & other)

Copy constructor.

7.27.2.3 virtual Couenne::CouenneInterface::~CouenneInterface () [virtual]

Destructor.

7.27.3 Member Function Documentation

7.27.3.1 virtual CouenneInterface* Couenne::CouenneInterface::clone (bool CopyData) [virtual]

virutal copy constructor.

7.27.3.2 virtual std::string Couenne::CouenneInterface::appName () [inline], [virtual]

Definition at line 49 of file BonCouenneInterface.hpp.

7.27.3.3 virtual void Couenne::CouenneInterface::extractLinearRelaxation (OsiSolverInterface & si, CouenneCutGenerator & couenneCg, bool getObj = 1, bool solveNlp = 1) [virtual]

Extract a linear relaxation of the MINLP.

Solve the continuous relaxation and takes first-order outer-approximation constraints at the optimum. The put everything in an OsiSolverInterface.

7.27.3.4 virtual void Couenne::CouenneInterface::setAppDefaultOptions (Ipopt::SmartPtr<Ipopt::OptionsList> Options) [virtual]

To set some application specific defaults.

7.27.3.5 bool Couenne::CouenneInterface::haveNlpSolution () [inline]

return value of have_nlp_solution_

Definition at line 73 of file BonCouenneInterface.hpp.

7.27.4 Member Data Documentation

7.27.4.1 bool Couenne::CouenneInterface::have_nlp_solution_ [protected]

true if we got an integer feasible solution from initial solve

Definition at line 79 of file BonCouenneInterface.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/interfaces/BonCouenneInterface.hpp

7.28 Couenne::CouenneIterativeRounding Class Reference

An iterative rounding heuristic, tailored for nonconvex MINLPs.

```
#include <CouenneIterativeRounding.hpp>
```

Public Member Functions

- **CouennelterativeRounding ()**
Default constructor.
- **CouennelterativeRounding (Bonmin::OsiTMINLPIterface *nlp, OsiSolverInterface *cinlp, CouenneProblem *couenne, Ipopt::SmartPtr< Ipopt::OptionsList > options)**
Constructor with model and Couenne problems.
- **CouennelterativeRounding (const CouennelterativeRounding &other)**
Copy constructor.
- **virtual ~CouennelterativeRounding ()**
Destructor.
- **virtual CbcHeuristic * clone () const**
Clone.
- **CouennelterativeRounding & operator= (const CouennelterativeRounding &rhs)**
Assignment operator.
- **void setNlp (Bonmin::OsiTMINLPIterface *nlp, OsiSolverInterface *cinlp)**
Set the minlp solver.
- **void setCouenneProblem (CouenneProblem *couenne)**
Set the couenne problem to use.
- **void resetModel (CbcModel *model)**
Does nothing.
- **int solution (double &objectiveValue, double *newSolution)**
Run heuristic, return 1 if a better solution than the one passed is found and 0 otherwise.
- **void setMaxRoundingIter (int value)**
Set maximum number of iterations for each rounding phase.
- **void setMaxFirPoints (int value)**
Set maximum number of points that we try to round in F-IR.
- **void setMaxTime (double value)**
Set maximum CPU time for the heuristic at each node.
- **void setMaxTimeFirstCall (double value)**
Set maximum CPU time for the heuristic at the root node only.
- **void setOmega (double value)**
Set the value for omega, the multiplicative factor for the minimum log-barrier parameter mu used by F-IR whenever we need to generate a new NLP feasible point (in the interior of the feasible region)
- **void setBaseLbRhs (int value)**
Set the base value for the rhs of the local branching constraint in the I-IR heuristic.
- **void setAggressiveness (int value)**
Set aggressiveness of heuristic.

Static Public Member Functions

- **static void registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions >)**
initialize options to be read later

7.28.1 Detailed Description

An iterative rounding heuristic, tailored for nonconvex MINLPs.

It solves a sequence of MILPs and NLPs for a given number of iterations, or until a better solution is found.

Definition at line 36 of file CouennelterativeRounding.hpp.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 Couenne::CouennelterativeRounding::CouennelterativeRounding()

Default constructor.

7.28.2.2 Couenne::CouennelterativeRounding::CouennelterativeRounding(Bonmin::OsiTMINLPIterface * nlp, OsiSolverInterface * cinlp, CouenneProblem * couenne, Ipopt::SmartPtr<Ipopt::OptionsList> options)

Constructor with model and [Couenne](#) problems.

7.28.2.3 Couenne::CouennelterativeRounding::CouennelterativeRounding(const CouennelterativeRounding & other)

Copy constructor.

7.28.2.4 virtual Couenne::CouennelterativeRounding::~CouennelterativeRounding() [virtual]

Destructor.

7.28.3 Member Function Documentation

7.28.3.1 virtual CbcHeuristic* Couenne::CouennelterativeRounding::clone() const [virtual]

Clone.

7.28.3.2 CouennelterativeRounding& Couenne::CouennelterativeRounding::operator=(const CouennelterativeRounding & rhs)

Assignment operator.

7.28.3.3 void Couenne::CouennelterativeRounding::setNlp(Bonmin::OsiTMINLPIterface * nlp, OsiSolverInterface * cinlp)

Set the minlp solver.

7.28.3.4 void Couenne::CouennelterativeRounding::setCouenneProblem(CouenneProblem * couenne) [inline]

Set the couenne problem to use.

Definition at line 62 of file CouennelterativeRounding.hpp.

7.28.3.5 void Couenne::CouennelterativeRounding::resetModel(CbcModel * model) [inline]

Does nothing.

Definition at line 67 of file CouennelterativeRounding.hpp.

7.28.3.6 int Couenne::CouennelterativeRounding::solution(double & objectiveValue, double * newSolution)

Run heuristic, return 1 if a better solution than the one passed is found and 0 otherwise.

objectiveValue Best known solution in input and value of solution found in output newSolution Solution found by heuristic.

7.28.3.7 void Couenne::CouennelterativeRounding::setMaxRoundingIter (int value) [inline]

Set maximum number of iterations for each rounding phase.

Definition at line 79 of file CouennelterativeRounding.hpp.

7.28.3.8 void Couenne::CouennelterativeRounding::setMaxFirPoints (int value) [inline]

Set maximum number of points that we try to round in F-IR.

Definition at line 84 of file CouennelterativeRounding.hpp.

7.28.3.9 void Couenne::CouennelterativeRounding::setMaxTime (double value) [inline]

Set maximum CPU time for the heuristic at each node.

Definition at line 89 of file CouennelterativeRounding.hpp.

7.28.3.10 void Couenne::CouennelterativeRounding::setMaxTimeFirstCall (double value) [inline]

Set maximum CPU time for the heuristic at the root node only.

Definition at line 94 of file CouennelterativeRounding.hpp.

7.28.3.11 void Couenne::CouennelterativeRounding::setOmega (double value) [inline]

Set the value for omega, the multiplicative factor for the minimum log-barrier parameter mu used by F-IR whenever we need to generate a new NLP feasible point (in the interior of the feasible region)

Definition at line 102 of file CouennelterativeRounding.hpp.

7.28.3.12 void Couenne::CouennelterativeRounding::setBaseLbRhs (int value) [inline]

Set the base value for the rhs of the local branching constraint in the I-IR heuristic.

The actual rhs is then computed depending on current variable bounds

Definition at line 109 of file CouennelterativeRounding.hpp.

7.28.3.13 void Couenne::CouennelterativeRounding::setAggressiveness (int value)

Set aggressiveness of heuristic.

Three levels, that sets various parameters accordingly.

The levels are: 0: maxRoundingIter = 5, maxTimeFirstCall = 300, maxFirPoints = 5, maxTime = 60 1: maxRoundingIter = 10, maxTimeFirstCall = 300, maxFirPoints = 5, maxTime = 120 2: maxRoundingIter = 20, maxTimeFirstCall = 1000, maxFirPoints = 5, maxTime = 300

7.28.3.14 static void Couenne::CouennelterativeRounding::registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions >) [static]

initialize options to be read later

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/heuristics/[CouennelterativeRounding.hpp](#)

7.29 Couenne::CouenneMINLPInterface Class Reference

This class provides an [Osi](#) interface for a Mixed Integer Linear Program expressed as a TMINLP (so that we can use it for example as the continuous solver in Cbc).

```
#include <CouenneMINLPInterface.hpp>
```

Public Member Functions

- void [setObj](#) (int index, [expression](#) *newObj)
REMOVE — backward compatibility sets objective[index] at newObj.
- void [setInitSol](#) (const [CouNumber](#) *sol)
sets the initial solution for the NLP solver
- [CouNumber](#) [solve](#) ([CouNumber](#) *solution)
solves and returns the optimal objective function and the solution
- [CouenneProblem](#) * [problem](#) () const
return pointer to Couenne problem
- [Ipopt::OptionsList](#) * [options](#) () const
return pointer to options

7.29.1 Detailed Description

This class provides an [Osi](#) interface for a Mixed Integer Linear Program expressed as a TMINLP (so that we can use it for example as the continuous solver in Cbc).

Definition at line 59 of file CouenneMINLPInterface.hpp.

7.29.2 Member Function Documentation

7.29.2.1 void Couenne::CouenneMINLPInterface::setObj(int index, expression * newObj) [inline]

REMOVE — backward compatibility sets objective[index] at newObj.

Definition at line 65 of file CouenneMINLPInterface.hpp.

7.29.2.2 void Couenne::CouenneMINLPInterface::setInitSol(const CouNumber * sol)

sets the initial solution for the NLP solver

7.29.2.3 CouNumber Couenne::CouenneMINLPInterface::solve(CouNumber * solution)

solves and returns the optimal objective function and the solution

7.29.2.4 CouenneProblem* Couenne::CouenneMINLPInterface::problem() const [inline]

return pointer to Couenne problem

Definition at line 75 of file CouenneMINLPInterface.hpp.

7.29.2.5 Ipopt::OptionsList* Couenne::CouenneMINLPInterface::options() const [inline]

return pointer to options

Definition at line 79 of file CouenneMINLPInterface.hpp.

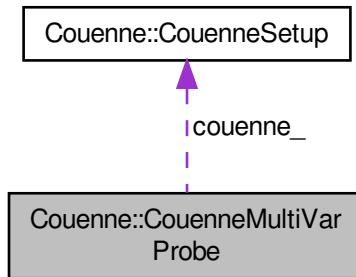
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/interfaces/CouenneMINLPInterface.hpp

7.30 Couenne::CouenneMultiVarProbe Class Reference

```
#include <CouenneMultiVarProbe.hpp>
```

Collaboration diagram for Couenne::CouenneMultiVarProbe:



Public Member Functions

- [CouenneMultiVarProbe \(CouenneSetup *couenne, const Ipopt::SmartPtr<Ipopt::OptionsList> options\)](#)
Constructor.
- [CouenneMultiVarProbe \(const CouenneMultiVarProbe &rhs\)](#)
Copy constructor.
- [~CouenneMultiVarProbe \(\)](#)
Destructor.
- [CouenneMultiVarProbe * clone \(\) const](#)
Clone method (necessary for the abstract CglCutGenerator class)
- [void generateCuts \(const OsiSolverInterface &solver, OsiCuts &cuts, const CglTreeInfo=CglTreeInfo\(\)\) const](#)
The main CglCutGenerator; not implemented yet.

Protected Attributes

- [CouenneSetup * couenne_](#)
Pointer to the CouenneProblem representation.
- int [numCols_](#)
Number of columns (want to have this handy)
- double [maxTime_](#)
Maximum time to probe one variable.

7.30.1 Detailed Description

Definition at line 25 of file CouenneMultiVarProbe.hpp.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 `Couenne::CouenneMultiVarProbe::CouenneMultiVarProbe (CouenneSetup * couenne, const Ipopt::SmartPtr<Ipopt::OptionsList > options)`

Constructor.

7.30.2.2 `Couenne::CouenneMultiVarProbe::CouenneMultiVarProbe (const CouenneMultiVarProbe & rhs)`

Copy constructor.

7.30.2.3 `Couenne::CouenneMultiVarProbe::~CouenneMultiVarProbe ()`

Destructor.

7.30.3 Member Function Documentation

7.30.3.1 `CouenneMultiVarProbe* Couenne::CouenneMultiVarProbe::clone () const [inline]`

Clone method (necessary for the abstract CglCutGenerator class)

Definition at line 40 of file CouenneMultiVarProbe.hpp.

7.30.3.2 `void Couenne::CouenneMultiVarProbe::generateCuts (const OsiSolverInterface & solver, OsiCuts & cuts, const CglTreeInfo = CglTreeInfo ()) const`

The main CglCutGenerator; not implemented yet.

7.30.4 Member Data Documentation

7.30.4.1 `CouenneSetup* Couenne::CouenneMultiVarProbe::couenne_ [protected]`

Pointer to the [CouenneProblem](#) representation.

Definition at line 51 of file CouenneMultiVarProbe.hpp.

7.30.4.2 `int Couenne::CouenneMultiVarProbe::numCols_ [protected]`

Number of columns (want to have this handy)

Definition at line 54 of file CouenneMultiVarProbe.hpp.

7.30.4.3 `double Couenne::CouenneMultiVarProbe::maxTime_ [protected]`

Maximum time to probe one variable.

Definition at line 57 of file CouenneMultiVarProbe.hpp.

The documentation for this class was generated from the following file:

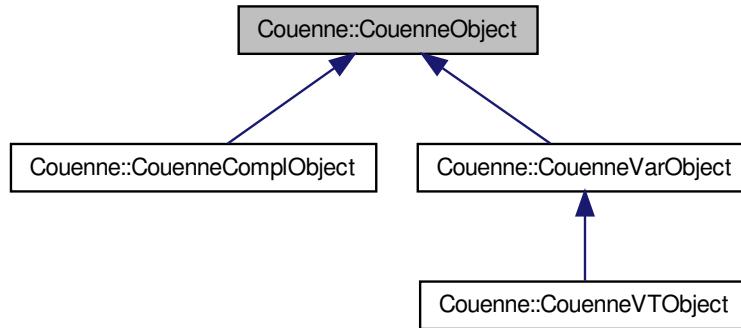
- /home/ted/COIN/trunk/Couenne/src/bound_tightening/[CouenneMultiVarProbe.hpp](#)

7.31 Couenne::CouenneObject Class Reference

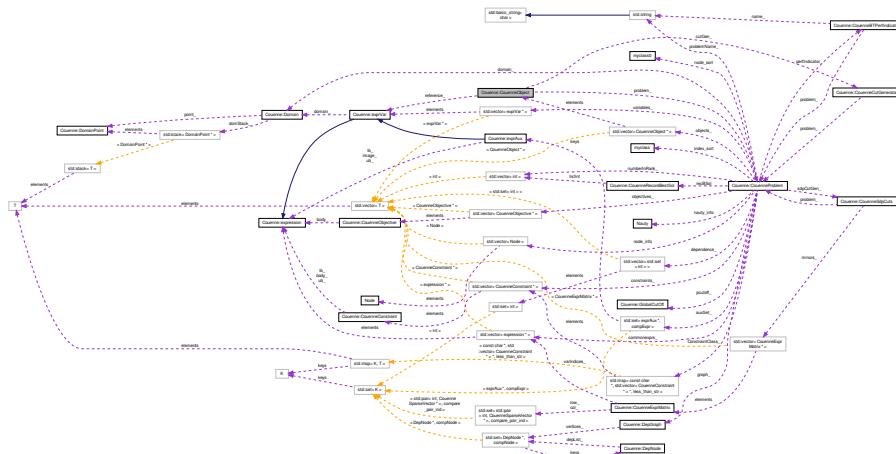
OsiObject for auxiliary variables $w=f(x)$.

```
#include <CouenneObject.hpp>
```

Inheritance diagram for Couenne::CouenneObject:



Collaboration diagram for Couenne::CouenneObject:



Public Types

- enum `pseudocostMult` {
 INFEASIBILITY, INTERVAL_LP, INTERVAL_LP_REV, INTERVAL_BR,
 INTERVAL_BR_REV, PROJECTDIST }
 type of up/down estimate to return for pseudocosts
 - enum `branch_obj` { EXPR_OBJ, VAR_OBJ, VT_OBJ }
 type of object (for branching variable selection)

- enum `brSelStrat` {
 `NO_STRATEGY`, `NO_BRANCH`, `MID_INTERVAL`, `MIN_AREA`,
`BALANCED`, `LP_CENTRAL`, `LP_CLAMPED` }
- strategy names*

Public Member Functions

- `CouenneObject ()`
empty constructor (for unused objects)
- `CouenneObject (CouenneCutGenerator *cutgen, CouenneProblem *p, exprVar *ref, Bonmin::BabSetupBase *base, JnlstPtr jnlst)`
Constructor with information for branching point selection strategy.
- `CouenneObject (exprVar *ref, Bonmin::BabSetupBase *base, JnlstPtr jnlst)`
Constructor with lesser information, used for infeasibility only.
- `~CouenneObject ()`
Destructor.
- `CouenneObject (const CouenneObject &src)`
Copy constructor.
- virtual `CouenneObject * clone () const`
Cloning method.
- void `setParameters (Bonmin::BabSetupBase *base)`
set object parameters by reading from command line
- virtual double `infeasibility (const OsiBranchingInformation *info, int &way) const`
compute infeasibility of this variable, $|w - f(x)|$ (where w is the auxiliary variable defined as $w = f(x)$)
- virtual double `checkInfeasibility (const OsiBranchingInformation *info) const`
compute infeasibility of this variable, $|w - f(x)|$, where w is the auxiliary variable defined as $w = f(x)$
- virtual double `feasibleRegion (OsiSolverInterface *, const OsiBranchingInformation *) const`
fix (one of the) arguments of reference auxiliary variable
- virtual OsiBranchingObject * `createBranch (OsiSolverInterface *, const OsiBranchingInformation *, int) const`
create `CouenneBranchingObject` or `CouenneThreeWayBranchObj` based on this object
- `exprVar * Reference () const`
return reference auxiliary variable
- enum `brSelStrat Strategy () const`
return branching point selection strategy
- `CouNumber getBrPoint (funtriplet *ft, CouNumber x0, CouNumber l, CouNumber u, const OsiBranchingInformation *info=NULL) const`
pick branching point based on current strategy
- `CouNumber midInterval (CouNumber x, CouNumber l, CouNumber u, const OsiBranchingInformation *info=NULL) const`
returns a point "inside enough" a given interval, or x if it already is.
- virtual double `downEstimate () const`
Return "down" estimate (for non-convex, distance old \leftrightarrow new LP point)
- virtual double `upEstimate () const`
Return "up" estimate (for non-convex, distance old \leftrightarrow new LP point)
- void `setEstimate (double est, int direction)`
set up/down estimate (0 for down, 1 for up).
- void `setEstimates (const OsiBranchingInformation *info, CouNumber *infeasibility, CouNumber *brpt) const`

- virtual bool `isCuttable () const`
are we on the bad or good side of the expression?
- virtual double `intInfeasibility (double value, double lb, double ub) const`
integer infeasibility: min {value - floor(value), ceil(value) - value}
- `CouNumber lp_clamp () const`
Defines safe interval percentage for using LP point as a branching point.
- virtual int `columnNumber () const`
Returns the column index.

Protected Attributes

- `CouenneCutGenerator * cutGen_`
pointer to cut generator (not necessary, can be NULL)
- `CouenneProblem * problem_`
pointer to Couenne problem
- `exprVar * reference_`
The (auxiliary) variable this branching object refers to.
- enum `brSelStrat strategy_`
Branching point selection strategy.
- `JnlstPtr jnlst_`
SmartPointer to the Journalist.
- `CouNumber alpha_`
Combination parameter for the mid-point branching point selection strategy.
- `CouNumber lp_clamp_`
Defines safe interval percentage for using LP point as a branching point.
- `CouNumber feas_tolerance_`
feasibility tolerance (equal to that of CouenneProblem)
- bool `doFBBT_`
shall we do Feasibility based Bound Tightening (FBBT) at branching?
- bool `doConvCuts_`
shall we add convexification cuts at branching?
- double `downEstimate_`
down estimate (to be used in pseudocost)
- double `upEstimate_`
up estimate (to be used in pseudocost)
- enum `pseudocostMult pseudoMultType_`
multiplier type for pseudocost

7.31.1 Detailed Description

OsiObject for auxiliary variables $w=f(x)$.

Associated with a multi-variate function $f(x)$ and a related infeasibility $|w-f(x)|$, creates branches to help restoring feasibility

Definition at line 57 of file CouenneObject.hpp.

7.31.2 Member Enumeration Documentation

7.31.2.1 enum Couenne::CouenneObject::pseudocostMult

type of up/down estimate to return for pseudocosts

Enumerator:

INFEASIBILITY
INTERVAL_LP
INTERVAL_LP_REV
INTERVAL_BR
INTERVAL_BR_REV
PROJECTDIST

Definition at line 62 of file CouenneObject.hpp.

7.31.2.2 enum Couenne::CouenneObject::branch_obj

type of object (for branching variable selection)

Enumerator:

EXPR_OBJ
VAR_OBJ
VT_OBJ

Definition at line 68 of file CouenneObject.hpp.

7.31.2.3 enum Couenne::CouenneObject::brSelStrat

strategy names

Enumerator:

NO_STRATEGY
NO_BRANCH
MID_INTERVAL
MIN_AREA
BALANCED
LP_CENTRAL
LP_CLAMPED

Definition at line 71 of file CouenneObject.hpp.

7.31.3 Constructor & Destructor Documentation

7.31.3.1 Couenne::CouenneObject::CouenneObject()

empty constructor (for unused objects)

7.31.3.2 Couenne::CouenneObject::CouenneObject (CouenneCutGenerator * *cutgen*, CouenneProblem * *p*, exprVar * *ref*, Bonmin::BabSetupBase * *base*, JnlstPtr *jnlst*)

Constructor with information for branching point selection strategy.

7.31.3.3 Couenne::CouenneObject::CouenneObject (exprVar * *ref*, Bonmin::BabSetupBase * *base*, JnlstPtr *jnlst*)

Constructor with lesser information, used for infeasibility only.

7.31.3.4 Couenne::CouenneObject::~CouenneObject () [inline]

Destructor.

Definition at line 85 of file CouenneObject.hpp.

7.31.3.5 Couenne::CouenneObject::CouenneObject (const CouenneObject & *src*)

Copy constructor.

7.31.4 Member Function Documentation

7.31.4.1 virtual CouenneObject* Couenne::CouenneObject::clone () const [inline], [virtual]

Cloning method.

Reimplemented in [Couenne::CouenneComplObject](#), [Couenne::CouenneVarObject](#), and [Couenne::CouenneVTOBJECT](#).

Definition at line 91 of file CouenneObject.hpp.

7.31.4.2 void Couenne::CouenneObject::setParameters (Bonmin::BabSetupBase * *base*)

set object parameters by reading from command line

7.31.4.3 virtual double Couenne::CouenneObject::infeasibility (const OsiBranchingInformation * *info*, int & *way*) const [virtual]

compute infeasibility of this variable, $|w - f(x)|$ (where *w* is the auxiliary variable defined as $w = f(x)$)

Reimplemented in [Couenne::CouenneVarObject](#), [Couenne::CouenneComplObject](#), and [Couenne::CouenneVTOBJECT](#).

7.31.4.4 virtual double Couenne::CouenneObject::checkInfeasibility (const OsiBranchingInformation * *info*) const [virtual]

compute infeasibility of this variable, $|w - f(x)|$, where *w* is the auxiliary variable defined as $w = f(x)$

Reimplemented in [Couenne::CouenneVarObject](#), and [Couenne::CouenneComplObject](#).

7.31.4.5 virtual double Couenne::CouenneObject::feasibleRegion (OsiSolverInterface * , const OsiBranchingInformation *) const [virtual]

fix (one of the) arguments of reference auxiliary variable

Reimplemented in [Couenne::CouenneVarObject](#).

7.31.4.6 virtual OsiBranchingObject* Couenne::CouenneObject::createBranch (OsiSolverInterface * , const OsiBranchingInformation * , int) const [virtual]

create [CouenneBranchingObject](#) or [CouenneThreeWayBranchObj](#) based on this object

Reimplemented in [Couenne::CouenneVarObject](#), and [Couenne::CouenneComplObject](#).

7.31.4.7 exprVar* Couenne::CouenneObject::Reference () const [inline]

return reference auxiliary variable

Definition at line 114 of file CouenneObject.hpp.

7.31.4.8 enum brSelStrat Couenne::CouenneObject::Strategy () const [inline]

return branching point selection strategy

Definition at line 118 of file CouenneObject.hpp.

7.31.4.9 CouNumber Couenne::CouenneObject::getBrPoint (funtriplet * ft, CouNumber x0, CouNumber l, CouNumber u, const OsiBranchingInformation * info = NULL) const

pick branching point based on current strategy

7.31.4.10 CouNumber Couenne::CouenneObject::midInterval (CouNumber x, CouNumber l, CouNumber u, const OsiBranchingInformation * info = NULL) const

returns a point "inside enough" a given interval, or x if it already is.

Modify alpha_ using gap provided by info

7.31.4.11 virtual double Couenne::CouenneObject::downEstimate () const [inline], [virtual]

Return "down" estimate (for non-convex, distance old <-> new LP point)

Definition at line 129 of file CouenneObject.hpp.

7.31.4.12 virtual double Couenne::CouenneObject::upEstimate () const [inline], [virtual]

Return "up" estimate (for non-convex, distance old <-> new LP point)

Definition at line 138 of file CouenneObject.hpp.

7.31.4.13 void Couenne::CouenneObject::setEstimate (double est, int direction) [inline]

set up/down estimate (0 for down, 1 for up).

This happens in [CouenneChooseStrong](#), where a new LP point is available and we can measure distance from old LP point. This is the denominator we use in pseudocost

Definition at line 150 of file CouenneObject.hpp.

7.31.4.14 void Couenne::CouenneObject::setEstimates (const OsiBranchingInformation * info, CouNumber * infeasibility, CouNumber * brpt) const

set up/down estimates based on branching information

7.31.4.15 virtual bool Couenne::CouenneObject::isCuttable () const [inline], [virtual]

are we on the bad or good side of the expression?

Reimplemented in [Couenne::CouenneVarObject](#).

Definition at line 159 of file CouenneObject.hpp.

7.31.4.16 virtual double Couenne::CouenneObject::intInfeasibility (double *value*, double *lb*, double *ub*) const [virtual]

integer infeasibility: min {value - floor(value), ceil(value) - value}

7.31.4.17 CouNumber Couenne::CouenneObject::lp_clamp () const [inline]

Defines safe interval percentage for using LP point as a branching point.

Definition at line 170 of file CouenneObject.hpp.

7.31.4.18 virtual int Couenne::CouenneObject::columnNumber () const [inline], [virtual]

Returns the column index.

Definition at line 174 of file CouenneObject.hpp.

7.31.5 Member Data Documentation

7.31.5.1 CouenneCutGenerator* Couenne::CouenneObject::cutGen_ [protected]

pointer to cut generator (not necessary, can be NULL)

Definition at line 180 of file CouenneObject.hpp.

7.31.5.2 CouenneProblem* Couenne::CouenneObject::problem_ [protected]

pointer to [Couenne](#) problem

Definition at line 183 of file CouenneObject.hpp.

7.31.5.3 exprVar* Couenne::CouenneObject::reference_ [protected]

The (auxiliary) variable this branching object refers to.

If the expression is w=f(x,y), this is w, as opposed to [CouenneBranchingObject](#), where it would be either x or y.

Definition at line 188 of file CouenneObject.hpp.

7.31.5.4 enum brSelStrat Couenne::CouenneObject::strategy_ [protected]

Branching point selection strategy.

Definition at line 191 of file CouenneObject.hpp.

7.31.5.5 JnlstPtr Couenne::CouenneObject::jnlst_ [protected]

SmartPointer to the Journalist.

Definition at line 194 of file CouenneObject.hpp.

7.31.5.6 CouNumber Couenne::CouenneObject::alpha_ [protected]

Combination parameter for the mid-point branching point selection strategy.

Definition at line 198 of file CouenneObject.hpp.

7.31.5.7 CouNumber Couenne::CouenneObject::lp_clamp_ [protected]

Defines safe interval percentage for using LP point as a branching point.

Definition at line 201 of file CouenneObject.hpp.

7.31.5.8 CouNumber Couenne::CouenneObject::feas_tolerance_ [protected]

feasibility tolerance (equal to that of [CouenneProblem](#))

Definition at line 204 of file CouenneObject.hpp.

7.31.5.9 bool Couenne::CouenneObject::doFBBT_ [protected]

shall we do Feasibility based Bound Tightening (FBBT) at branching?

Definition at line 207 of file CouenneObject.hpp.

7.31.5.10 bool Couenne::CouenneObject::doConvCuts_ [protected]

shall we add convexification cuts at branching?

Definition at line 210 of file CouenneObject.hpp.

7.31.5.11 double Couenne::CouenneObject::downEstimate_ [mutable], [protected]

down estimate (to be used in pseudocost)

Definition at line 213 of file CouenneObject.hpp.

7.31.5.12 double Couenne::CouenneObject::upEstimate_ [mutable], [protected]

up estimate (to be used in pseudocost)

Definition at line 216 of file CouenneObject.hpp.

7.31.5.13 enum pseudocostMult Couenne::CouenneObject::pseudoMultType_ [protected]

multiplier type for pseudocost

Definition at line 219 of file CouenneObject.hpp.

The documentation for this class was generated from the following file:

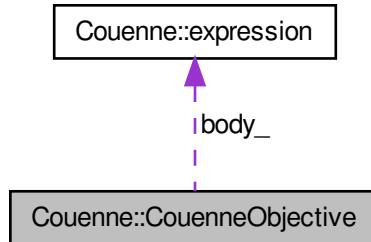
- /home/ted/COIN/trunk/Couenne/src/branch/[CouenneObject.hpp](#)

7.32 Couenne::CouenneObjective Class Reference

Objective function.

```
#include <CouenneProblemElem.hpp>
```

Collaboration diagram for Couenne::CouenneObjective:



Public Member Functions

- `CouenneObjective (expression *body)`
constructor
- `~CouenneObjective ()`
destructor
- `CouenneObjective (const CouenneObjective &o, Domain *d=NULL)`
copy constructor
- `CouenneObjective * clone (Domain *d=NULL) const`
cloning method
- `expression * Body () const`
get body
- `expression * Body (expression *newBody)`
Set body.
- `exprAux * standardize (CouenneProblem *p)`
Get standard form of this objective function.
- `void print (std::ostream &out=std::cout)`
Print to ostream.

Protected Attributes

- `expression * body_`
expression to optimize

7.32.1 Detailed Description

Objective function.

It consists of an expression only. We only assume minimization problems (proper sign changes are applied upon reading)

Definition at line 109 of file CouenneProblemElem.hpp.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 `Couenne::CouenneObjective::CouenneObjective(expression * body) [inline]`

constructor

Definition at line 119 of file CouenneProblemElem.hpp.

7.32.2.2 `Couenne::CouenneObjective::~CouenneObjective() [inline]`

destructor

Definition at line 123 of file CouenneProblemElem.hpp.

7.32.2.3 `Couenne::CouenneObjective::CouenneObjective(const CouenneObjective & o, Domain * d = NULL) [inline]`

copy constructor

Definition at line 127 of file CouenneProblemElem.hpp.

7.32.3 Member Function Documentation

7.32.3.1 `CouenneObjective* Couenne::CouenneObjective::clone(Domain * d = NULL) const [inline]`

cloning method

Definition at line 131 of file CouenneProblemElem.hpp.

7.32.3.2 `expression* Couenne::CouenneObjective::Body() const [inline]`

get body

Definition at line 135 of file CouenneProblemElem.hpp.

7.32.3.3 `expression* Couenne::CouenneObjective::Body(expression * newBody) [inline]`

Set body.

Definition at line 139 of file CouenneProblemElem.hpp.

7.32.3.4 `exprAux* Couenne::CouenneObjective::standardize(CouenneProblem * p) [inline]`

Get standard form of this objective function.

Definition at line 143 of file CouenneProblemElem.hpp.

7.32.3.5 `void Couenne::CouenneObjective::print(std::ostream & out = std::cout) [inline]`

Print to iostream.

Definition at line 147 of file CouenneProblemElem.hpp.

7.32.4 Member Data Documentation

7.32.4.1 `expression* Couenne::CouenneObjective::body_ [protected]`

expression to optimize

Definition at line 114 of file CouenneProblemElem.hpp.

The documentation for this class was generated from the following file:

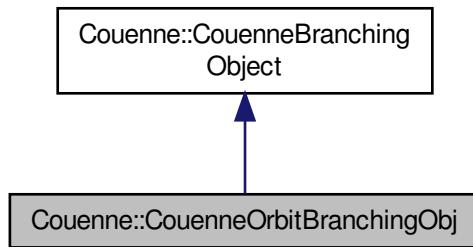
- /home/ted/COIN/trunk/Couenne/src/problem/CouenneProblemElem.hpp

7.33 Couenne::CouenneOrbitBranchingObj Class Reference

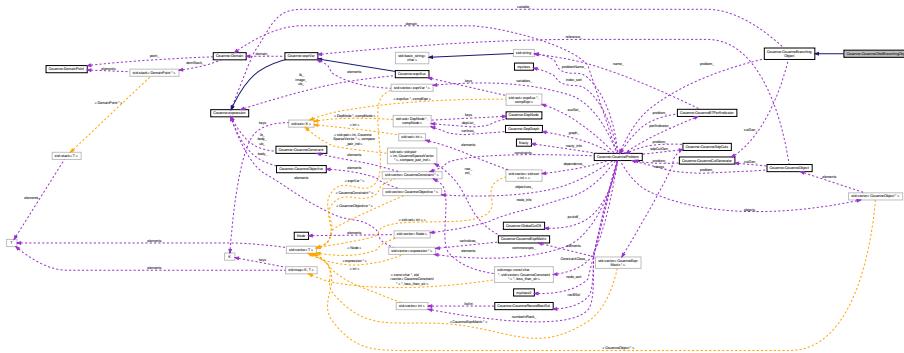
"Spatial" branching object.

```
#include <CouenneOrbitBranchingObj.hpp>
```

Inheritance diagram for Couenne::CouenneOrbitBranchingObj:



Collaboration diagram for Couenne::CouenneOrbitBranchingObj:



Public Member Functions

- **CouenneOrbitBranchingObj** (OsiSolverInterface *solver, const OsiObject *originalObject, **JnlstPtr** jnlst, **CouenneCutGenerator** *c, **CouenneProblem** *p, expression *var, int way, **CouNumber** brpoint, bool doFBBT, bool doConvCuts)

Constructor.
- **CouenneOrbitBranchingObj** (const **CouenneOrbitBranchingObj** &src)

- Copy constructor.*
- virtual OsiBranchingObject * **clone** () const
cloning method
- virtual double **branch** (OsiSolverInterface *solver=NULL)
Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.
- virtual bool **boundBranch** () const
does this branching object only change variable bounds?
- void **setSimulate** (bool s)
set simulate_ field below

Additional Inherited Members

7.33.1 Detailed Description

"Spatial" branching object.

Branching can also be performed on continuous variables.

Definition at line 36 of file CouenneOrbitBranchingObj.hpp.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 Couenne::CouenneOrbitBranchingObj::CouenneOrbitBranchingObj (OsiSolverInterface * solver, const OsiObject * originalObject, JnlstPtr jnlst, CouenneCutGenerator * c, CouenneProblem * p, expression * var, int way, CouNumber brpoint, bool doFBBT, bool doConvCuts)

Constructor.

7.33.2.2 Couenne::CouenneOrbitBranchingObj::CouenneOrbitBranchingObj (const CouenneOrbitBranchingObj & src) [inline]

Copy constructor.

Definition at line 53 of file CouenneOrbitBranchingObj.hpp.

7.33.3 Member Function Documentation

7.33.3.1 virtual OsiBranchingObject* Couenne::CouenneOrbitBranchingObj::clone () const [inline], [virtual]

cloning method

Reimplemented from [Couenne::CouenneBranchingObject](#).

Definition at line 58 of file CouenneOrbitBranchingObj.hpp.

7.33.3.2 virtual double Couenne::CouenneOrbitBranchingObj::branch (OsiSolverInterface * solver = NULL) [virtual]

Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.

Returns change in guessed objective on next branch

Reimplemented from [Couenne::CouenneBranchingObject](#).

7.33.3.3 virtual bool Couenne::CouenneOrbitBranchingObj::boundBranch() const [inline], [virtual]

does this branching object only change variable bounds?

Reimplemented from [Couenne::CouenneBranchingObject](#).

Definition at line 69 of file CouenneOrbitBranchingObj.hpp.

7.33.3.4 void Couenne::CouenneOrbitBranchingObj::setSimulate(bool s) [inline]

set simulate_ field below

Reimplemented from [Couenne::CouenneBranchingObject](#).

Definition at line 73 of file CouenneOrbitBranchingObj.hpp.

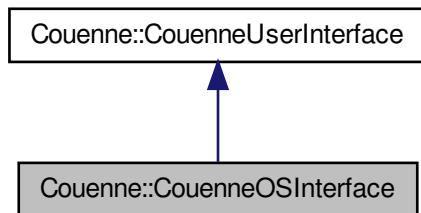
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/branch/[CouenneOrbitBranchingObj.hpp](#)

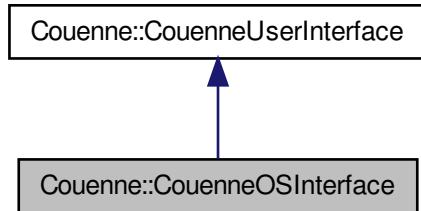
7.34 Couenne::CouenneOSInterface Class Reference

```
#include <CouenneOSInterface.hpp>
```

Inheritance diagram for Couenne::CouenneOSInterface:



Collaboration diagram for Couenne::CouenneOSInterface:



Public Member Functions

- [CouenneOSInterface \(Ipopt::SmartPtr< Ipopt::OptionsList > options_, Ipopt::SmartPtr< Ipopt::Journalist > jnlst_\)](#)
- [~CouenneOSInterface \(\)](#)
- [CouenneProblem * getCouenneProblem \(\)](#)
Should return the problem to solve in algebraic form.
- [Ipopt::SmartPtr< Bonmin::TMINLP > getTMINLP \(\)](#)
Should return the problem to solve as TMINLP.
- bool [writeSolution \(Bonmin::Bab &bab\)](#)
Called after B&B finished.

Static Public Member Functions

- static void [registerOptions \(Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions\)](#)

Additional Inherited Members

7.34.1 Detailed Description

Definition at line 36 of file CouenneOSInterface.hpp.

7.34.2 Constructor & Destructor Documentation

7.34.2.1 Couenne::CouenneOSInterface::CouenneOSInterface (Ipopt::SmartPtr< Ipopt::OptionsList > options_, Ipopt::SmartPtr< Ipopt::Journalist > jnlst_) [inline]

Definition at line 46 of file CouenneOSInterface.hpp.

7.34.2.2 Couenne::CouenneOSInterface::~CouenneOSInterface()

7.34.3 Member Function Documentation

7.34.3.1 static void Couenne::CouenneOSInterface::registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > *roptions*)
[static]

7.34.3.2 CouenneProblem* Couenne::CouenneOSInterface::getCouenneProblem () [virtual]

Should return the problem to solve in algebraic form.

NOTE: Couenne is (currently) going to modify this problem!

Implements Couenne::CouenneUserInterface.

7.34.3.3 Ipopt::SmartPtr<Bonmin::TMINLP> Couenne::CouenneOSInterface::getTMINLP () [virtual]

Should return the problem to solve as TMINLP.

Implements Couenne::CouenneUserInterface.

7.34.3.4 bool Couenne::CouenneOSInterface::writeSolution (Bonmin::Bab & *bab*) [virtual]

Called after B&B finished.

Should write solution information.

Reimplemented from Couenne::CouenneUserInterface.

The documentation for this class was generated from the following file:

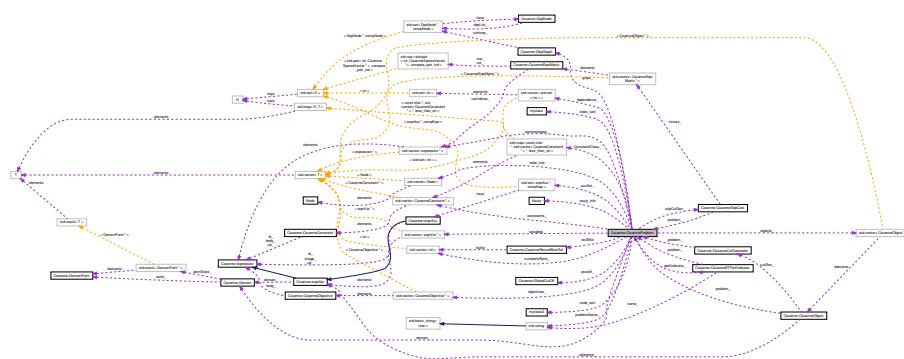
- /home/ted/COIN/trunk/Couenne/src/main/CouenneOSInterface.hpp

7.35 Couenne::CouenneProblem Class Reference

Class for MINLP problems with symbolic information.

```
#include <CouenneProblem.hpp>
```

Collaboration diagram for Couenne::CouenneProblem:



Public Types

- enum multiSep { MulSepNone, MulSepSimple, MulSepTight }

Type of multilinear separation.

Public Member Functions

- `CouenneProblem (ASL *=NULL, Bonmin::BabSetupBase *base=NULL, JnlstPtr jnlst=NULL)`
Constructor.
- `CouenneProblem (const CouenneProblem &)`
Copy constructor.
- `~CouenneProblem ()`
Destructor.
- `void initOptions (Ipopt::SmartPtr< Ipopt::OptionsList > options)`
initializes parameters like doOBBT
- `CouenneProblem * clone () const`
Clone method (for use within CouenneCutGenerator::clone)
- `int nObjs () const`
Get number of objectives.
- `int nCons () const`
Get number of constraints.
- `int nOrigCons () const`
Get number of original constraints.
- `int nOrigVars () const`
Number of orig. variables.
- `int nDefVars () const`
Number of def'd variables.
- `int nOrigIntVars () const`
Number of original integers.
- `int nIntVars () const`
Number of integer variables.
- `int nVars () const`
Total number of variables.
- `void setNDefVars (int undefined_)`
- `std::vector< int > * Find_Orbit (int) const`
- `void sym_setup ()`
- `void Compute_Symmetry () const`
- `void Print_Orbits () const`
- `void ChangeBounds (const double *, const double *, int) const`
- `bool compare (register Node &a, register Node &b) const`
- `Nauty * getNtyInfo ()`
- `void setupSymmetry ()`
empty if no NTY, symmetry data structure setup otherwise
- `int evalOrder (int i) const`
get evaluation order index
- `int * evalVector ()`
get evaluation order vector (numbering_)
- `CouenneConstraint * Con (int i) const`
i-th constraint
- `CouenneObjective * Obj (int i) const`

- i-th objective*
- `exprVar * Var (int i) const`

Return pointer to i-th variable.
- `std::vector< exprVar * > & Variables ()`

Return vector of variables (symbolic representation)
- `std::set< exprAux *, compExpr > *& AuxSet ()`

Return pointer to set for comparisons.
- `DepGraph * getDepGraph ()`

Return pointer to dependence graph.
- `Domain * domain () const`

return current point & bounds
- `std::vector< expression * > & commonExprs ()`
- `CouNumber & X (int i) const`

x_i
- `CouNumber & Lb (int i) const`

lower bound on x_i
- `CouNumber & Ub (int i) const`

upper bound on x_i
- `CouNumber * X () const`

Return vector of variables.
- `CouNumber * Lb () const`

Return vector of lower bounds.
- `CouNumber * Ub () const`

Return vector of upper bounds.
- `CouNumber *& bestSol () const`

Best known solution (read from file)
- `CouNumber bestObj () const`

Objective of best known solution.
- `bool *& Commuted ()`

Get vector of commuted variables.
- `void addObjective (expression *, const std::string &="min")`

Add (non linear) objective function.
- `void addEQConstraint (expression *, expression *=NULL)`

Add equality constraint $h(x) = b$.
- `void addGEConstraint (expression *, expression *=NULL)`

Add \geq constraint, $h(x) \geq b$.
- `void addLEConstraint (expression *, expression *=NULL)`

Add \leq constraint, $h(x) \leq b$.
- `void addRNGConstraint (expression *, expression *=NULL, expression *=NULL)`

Add range constraint, $a \leq h(x) \leq b$.
- `void setObjective (int indObj=0, expression *=NULL, const std::string &="min")`

Add (non linear) objective function.
- `expression * addVariable (bool isint=false, Domain *d=NULL)`

Add original variable.
- `exprAux * addAuxiliary (expression *)`

Add auxiliary variable and associate it with expression given as argument (used in standardization)
- `void reformulate (CouenneCutGenerator *=NULL)`

- preprocess problem in order to extract linear relaxations etc.
- bool **standardize** ()

Break problem's nonlinear constraints in simple expressions to be convexified later.
- void **print** (std::ostream &=std::cout)

Display current representation of problem: objective, linear and nonlinear constraints, and auxiliary variables.
- bool **doFBBT** () const

shall we do Feasibility Based Bound Tightening?
- bool **doRCBT** () const

shall we do reduced cost Bound Tightening?
- bool **doOBBT** () const

shall we do Optimality Based Bound Tightening?
- bool **doABT** () const

shall we do Aggressive Bound Tightening?
- int **logObbtLev** () const

How often shall we do OBBT?
- int **logAbtLev** () const

How often shall we do ABT?
- void **writeAMPL** (const std::string &fname, bool aux)

Write nonlinear problem to a .mod file (with lots of defined variables)
- void **writeGAMS** (const std::string &fname)

Write nonlinear problem to a .gms file.
- void **writeLP** (const std::string &fname)

Write nonlinear problem to a .lp file.
- void **initAuxs** () const

Initialize auxiliary variables and their bounds from original variables.
- void **getAuxs** (CouNumber *) const

Get auxiliary variables from original variables.
- bool **boundTightening** (**t_chg_bounds** *, const CglTreeInfo info, Bonmin::BabInfo *=NULL) const

tighten bounds using propagation, implied bounds and reduced costs
- bool **btCore** (**t_chg_bounds** *chg_bds) const

core of the bound tightening procedure
- int **obbt** (const **CouenneCutGenerator** *cg, const OsiSolverInterface &csi, OsiCuts &cs, const CglTreeInfo &info, Bonmin::BabInfo *babInfo, **t_chg_bounds** *chg_bds)

Optimality Based Bound Tightening.
- bool **aggressiveBT** (Bonmin::OsiTMINLPInterface *nlp, **t_chg_bounds** *, const CglTreeInfo &info, Bonmin::BabInfo *=NULL) const

aggressive bound tightening.
- int **redCostBT** (const OsiSolverInterface *psi, **t_chg_bounds** *chg_bds) const

procedure to strengthen variable bounds.
- int **tightenBounds** (**t_chg_bounds** *) const

"Forward" bound tightening, that is, propagate bound of variable x in an expression $w = f(x)$ to the bounds of w .
- int **impliedBounds** (**t_chg_bounds** *) const

"Backward" bound tightening, aka implied bounds.
- void **fillQuadIndices** ()

Look for quadratic terms to be used with SDP cuts.
- void **fillObjCoeff** (double *&)

Fill vector with coefficients of objective function.

- void `auxiliarize (exprVar *, exprVar *=NULL)`
Replace all occurrences of original variable with new aux given as argument.
- void `setCutOff (CouNumber cutoff, const CouNumber *sol=NULL) const`
Set cutoff.
- void `resetCutOff (CouNumber value=COUENNE_INFINITY) const`
Reset cutoff.
- `CouNumber getCutOff () const`
Get cutoff.
- `CouNumber * getCutOffSol () const`
Get cutoff solution.
- void `installCutOff () const`
Make cutoff known to the problem.
- `ConstJnlstPtr Jnlst () const`
Provide Journalist.
- bool `checkNLP (const double *solution, double &obj, bool recompute=false) const`
Check if solution is MINLP feasible.
- int `getIntegerCandidate (const double *xFrac, double *xInt, double *lb, double *ub) const`
generate integer NLP point Y starting from fractional solution using bound tightening
- bool `readOptimum (std::string *fname=NULL)`
Read best known solution from file given in argument.
- `exprAux * linStandardize (bool addAux, CouNumber c0, LinMap &lmap, QuadMap &qmap)`
standardization of linear exprOp's
- int `splitAux (CouNumber, expression *, expression *&, bool *, enum expression::auxSign &)`
split a constraint $w - f(x) = c$ into w's index (it is returned) and rest = $f(x) + c$
- void `indcoe2vector (int *indexL, CouNumber *coeff, std::vector< std::pair< exprVar *, CouNumber > > &lcoeff)`
translates pair (indices, coefficients) into vector with pointers to variables
- void `indcoe2vector (int *indexI, int *indexJ, CouNumber *coeff, std::vector< quadElem > &qcoeff)`
translates triplet (indicesI, indicesJ, coefficients) into vector with pointers to variables
- void `decomposeTerm (expression *term, CouNumber initCoe, CouNumber &c0, LinMap &lmap, QuadMap &qmap)`
*given (expression *) element of sum, returns (coe,ind0,ind1) depending on element:*
- const std::string & `problemName () const`
return problem name
- void `setProblemName (std::string &problemName__)`
- const std::vector< std::set< int > > & `Dependence () const`
return inverse dependence structure
- const std::vector< CouenneObject * > & `Objects () const`
return object vector
- int `findSOS (CbcModel *CbcModelPtr, OsiSolverInterface *solver, OsiObject **objects)`
find SOS constraints in problem
- void `setMaxCpuTime (double time)`
set maximum CPU time
- double `getMaxCpuTime () const`
return maximum CPU time
- void `setBase (Bonmin::BabSetupBase *base)`

- save *CouenneBase*
- void **createUnusedOriginals** ()

Some originals may be unused due to their zero multiplicity (that happens when they are duplicates).
- void **restoreUnusedOriginals** (*CouNumber* *=NULL) const

Some originals may be unused due to their zero multiplicity (that happens when they are duplicates).
- int * **unusedOriginalsIndices** ()

return indices of neglected redundant variables
- int **nUnusedOriginals** ()

number of unused originals
- enum **multiSep MultilinSep** () const

return type of separator for multilinear terms
- bool **fbbtReachedIterLimit** () const

true if latest call to FBBT terminated due to iteration limit reached
- bool **orbitalBranching** () const

return true if orbital branching activated
- void **setCheckAuxBounds** (bool value)

set the value for checkAuxBounds.
- bool **checkAuxBounds** () const

return true if bounds of auxiliary variables have to be satisfied whenever a solution is tested for MINLP feasibility
- enum **TrilinDecompType getTrilinDecompType** ()

return type of decomposition of quadrilinear terms
- Bonmin::BabSetupBase * **bonBase** () const

options
- double **constObjVal** () const

returns constant objective value if it contains no variables
- **CouenneSdpCuts** * **getSdpCutGen** ()

Returns pointer to sdp cut generator.
- int **getLastPrioSort** () const
- void **setLastPrioSort** (int givenLastPS)
- **CouenneRecordBestSol** * **getRecordBestSol** () const

returns recorded best solution
- double **getFeasTol** ()

returns feasibility tolerance
- double **checkObj** (const *CouNumber* *sol, const double &precision) const

Recompute objective value for sol.
- bool **checkInt** (const *CouNumber* *sol, const int from, const int upto, const std::vector< int > listInt, const bool origVarOnly, const bool stopAtFirstViol, const double precision, double &maxViol) const

check integrality of vars in sol with index between from and upto (original vars only if origVarOnly == true); return true if all integer vars are within precision of an integer value
- bool **checkBounds** (const *CouNumber* *sol, const bool stopAtFirstViol, const double precision, double &maxViol) const

Check bounds; returns true iff feasible for given precision.
- bool **checkAux** (const *CouNumber* *sol, const bool stopAtFirstViol, const double precision, double &maxViol) const

returns true iff value of all auxiliaries are within bounds
- bool **checkCons** (const *CouNumber* *sol, const bool stopAtFirstViol, const double precision, double &maxViol) const

returns true iff value of all auxiliaries are within bounds

- bool `checkNLP2` (const double *solution, const double obj, const bool careAboutObj, const bool stopAtFirstViol, const bool checkAll, const double precision) const

Return true if either solution or recomputed_solution obtained using `getAuxs()` from the original variables in solution is feasible within precision (the solution with minimum violation is then stored in recBSol->modSol, as well as its value and violation); return false otherwise.
- bool `checkNLP0` (const double *solution, double &obj, bool recompute_obj=false, const bool careAboutObj=false, const bool stopAtFirstViol=true, const bool checkAll=false, const double precision=-1) const

And finally a method to get both.
- std::vector< `CouenneConstraint` * > * `ConstraintClass` (const char *str)

return particular constraint class.

Static Public Member Functions

- static void `registerOptions` (Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions)

Add list of options to be read from file.

Public Attributes

- int `minDepthPrint_`
- int `minNodePrint_`
- bool `doPrint_`
- std::vector< `Node` > `node_info`
- `Nauty` * `nauty_info`
- `myclass0` `node_sort`
- `myclass` `index_sort`

Protected Member Functions

- int `fake_tighten` (char direction, int index, const double *X, `CouNumber` *olb, `CouNumber` *oub, `t_chg_bounds` *chg_bds, `t_chg_bounds` *f_chg) const

single fake tightening.
- int `obbtInner` (OsiSolverInterface *, OsiCuts &, `t_chg_bounds` *, Bonmin::BabInfo *) const

Optimality Based Bound Tightening – inner loop.
- int `obbt_iter` (OsiSolverInterface *csi, `t_chg_bounds` *chg_bds, const `CoinWarmStart` *warmstart, Bonmin::BabInfo *babInfo, double *objcoe, int sense, int index) const
- int `call_iter` (OsiSolverInterface *csi, `t_chg_bounds` *chg_bds, const `CoinWarmStart` *warmstart, Bonmin::BabInfo *babInfo, double *objcoe, enum `nodeType` type, int sense) const
- void `analyzeSparsity` (`CouNumber`, `LinMap` &, `QuadMap` &)

analyze sparsity of potential exprQuad/exprGroup and change linear/quadratic maps accordingly, if necessary by adding new auxiliary variables and including them in the linear map
- void `flattenMul` (`expression` *mul, `CouNumber` &coe, std::map< int, `CouNumber` > &indices)

re-organizes multiplication and stores indices (and exponents) of its variables
- void `realign` ()

clear all spurious variables pointers not referring to the variables_ vector
- void `fillDependence` (Bonmin::BabSetupBase *base, `CouenneCutGenerator` *=NULL)

fill dependence_ structure
- void `fillIntegerRank` () const

fill freeIntegers_ array
- int `testIntFix` (int index, `CouNumber` xFrac, enum fixType *fixed, `CouNumber` *xInt, `CouNumber` *dualL, `CouNumber` *dualR, `CouNumber` *olb, `CouNumber` *oub, bool patient) const

Test fixing of an integer variable (used in `getIntegerCandidate()`)

Protected Attributes

- std::string `problemName_`
problem name
- std::vector< `exprVar *` > `variables_`
Variables (original, auxiliary, and defined)
- std::vector< `CouenneObjective *` > `objectives_`
Objectives.
- std::vector< `CouenneConstraint *` > `constraints_`
Constraints.
- std::vector< `expression *` > `commonexprs_`
AMPL's common expressions (read from AMPL through structures cexprs and cexprs1)
- Domain `domain_`
current point and bounds;
- std::set< `exprAux *`, `compExpr` > * `auxSet_`
Expression map for comparison in standardization and to count occurrences of an auxiliary.
- int `curnvars_`
Number of elements in the `x_`, `lb_`, `ub_` arrays.
- int `nIntVars_`
Number of discrete variables.
- CouNumber * `optimum_`
Best solution known to be loaded from file – for testing purposes.
- CouNumber `bestObj_`
Best known objective function.
- bool * `commuted_`
Variables that have commuted to auxiliary.
- int * `numbering_`
numbering of variables.
- int `nDefined_`
Number of "defined variables" (aka "common expressions")
- DepGraph * `graph_`
Dependence (acyclic) graph: shows dependence of all auxiliary variables on one another and on original variables.
- int `nOrigVars_`
Number of original variables.
- int `nOrigCons_`
Number of original constraints (disregarding those that turned into auxiliary variable definition)
- int `nOrigIntVars_`
Number of original integer variables.
- GlobalCutOff * `pcutoff_`
Pointer to a global cutoff object.
- bool `created_pcutoff_`
flag indicating if this class is creator of global cutoff object
- bool `doFBBT_`
do Feasibility-based bound tightening
- bool `doRCBT_`
do reduced cost bound tightening
- bool `doOBBT_`

- do Optimality-based bound tightening
- bool **doABT_**
do Aggressive bound tightening
- int **logObbtLev_**
frequency of Optimality-based bound tightening
- int **logAbtLev_**
frequency of Aggressive bound tightening
- **JnlstPtr jnlst_**
SmartPointer to the Journalist.
- **CouNumber opt_window_**
window around known optimum (for testing purposes)
- bool **useQuadratic_**
Use quadratic expressions?
- **CouNumber feas_tolerance_**
feasibility tolerance (to be used in checkNLP)
- std::vector< std::set< int > > **dependence_**
inverse dependence structure: for each variable x give set of auxiliary variables (or better, their indices) whose expression depends on x
- std::vector< **CouenneObject *objects_**
vector of pointer to CouenneObjects.
- int * **integerRank_**
each element is true if variable is integer and, if auxiliary, depends on no integer
- std::vector< int > **numberInRank_**
 $numberInRank_{[i]}$ is the number of integer variables in rank i
- double **maxCpuTime_**
maximum cpu time
- Bonmin::BabSetupBase * **bonBase_**
options
- ASL * **asl_**
AMPL structure pointer (temporary — looking forward to embedding into OS...)
- int * **unusedOriginalsIndices_**
some originals may be unused due to their zero multiplicity (that happens when they are duplicates).
- int **nUnusedOriginals_**
number of unused originals
- int **lastPrioSort_**
- **CouenneRecordBestSol * recBSol**
- enum **multiSep multilinSep_**
Type of Multilinear separation.
- int **max_fbbt_iter_**
number of FBBT iterations
- bool **fbbtReachedIterLimit_**
true if FBBT exited for iteration limits as opposed to inability to further tighten bounds
- bool **orbitalBranching_**
use orbital branching?
- bool **checkAuxBounds_**
check bounds on auxiliary variables when verifying MINLP feasibility of a solution.
- enum **TrilinDecompType trilinDecompType_**

- `double constObjVal_`
constant value of the objective if no variable is declared in it
- `CouenneBTPerfIndicator * perfIndicator_`
Performance indicator for FBBT – to be moved away from [CouenneProblem](#) when we do it with FBBT.
- `std::map< const char *, std::vector< CouenneConstraint * > > ConstraintClass_`
Return particular constraint class.
- `CouenneSdpCuts * sdpCutGen_`
Temporary pointer to SDP cut generator.

Friends

- class `exprMul`

7.35.1 Detailed Description

Class for MINLP problems with symbolic information.

It is read from an AMPL .nl file and contains variables, AMPL's "defined variables" (aka common expressions), objective(s), and constraints in the form of expression's. Changes throughout the program occur in standardization.

Definition at line 169 of file CouenneProblem.hpp.

7.35.2 Member Enumeration Documentation**7.35.2.1 enum Couenne::CouenneProblem::multiSep**

Type of multilinear separation.

Enumerator:

- `MulSepNone`**
- `MulSepSimple`**
- `MulSepTight`**

Definition at line 179 of file CouenneProblem.hpp.

7.35.3 Constructor & Destructor Documentation**7.35.3.1 Couenne::CouenneProblem::CouenneProblem (ASL * =NULL, Bonmin::BabSetupBase * base =NULL, JnlstPtr jnlst =NULL)**

Constructor.

7.35.3.2 Couenne::CouenneProblem::CouenneProblem (const CouenneProblem &)

Copy constructor.

7.35.3.3 Couenne::CouenneProblem::~CouenneProblem ()

Destructor.

7.35.4 Member Function Documentation

7.35.4.1 void Couenne::CouenneProblem::initOptions (Ipopt::SmartPtr<Ipopt::OptionsList> options)

Initializes parameters like doOBBT

7.35.4.2 CouenneProblem* Couenne::CouenneProblem::clone () const [inline]

Clone method (for use within [CouenneCutGenerator::clone](#))

Definition at line 365 of file CouenneProblem.hpp.

7.35.4.3 int Couenne::CouenneProblem::nObjs () const [inline]

Get number of objectives.

Definition at line 368 of file CouenneProblem.hpp.

7.35.4.4 int Couenne::CouenneProblem::nCons () const [inline]

Get number of constraints.

Definition at line 369 of file CouenneProblem.hpp.

7.35.4.5 int Couenne::CouenneProblem::nOrigCons () const [inline]

Get number of original constraints.

Definition at line 370 of file CouenneProblem.hpp.

7.35.4.6 int Couenne::CouenneProblem::nOrigVars () const [inline]

Number of orig. variables.

Definition at line 372 of file CouenneProblem.hpp.

7.35.4.7 int Couenne::CouenneProblem::nDefVars () const [inline]

Number of def'd variables.

Definition at line 373 of file CouenneProblem.hpp.

7.35.4.8 int Couenne::CouenneProblem::nOrigIntVars () const [inline]

Number of original integers.

Definition at line 374 of file CouenneProblem.hpp.

7.35.4.9 int Couenne::CouenneProblem::nIntVars () const [inline]

Number of integer variables.

Definition at line 375 of file CouenneProblem.hpp.

7.35.4.10 `int Couenne::CouenneProblem::nVars() const [inline]`

Total number of variables.

Definition at line 376 of file CouenneProblem.hpp.

7.35.4.11 `void Couenne::CouenneProblem::setNDefVars(int ndefined_) [inline]`

Definition at line 378 of file CouenneProblem.hpp.

7.35.4.12 `std::vector<int>* Couenne::CouenneProblem::Find_Orbit(int) const`

7.35.4.13 `void Couenne::CouenneProblem::sym_setup()`

7.35.4.14 `void Couenne::CouenneProblem::Compute_Symmetry() const`

7.35.4.15 `void Couenne::CouenneProblem::Print_Orbits() const`

7.35.4.16 `void Couenne::CouenneProblem::ChangeBounds(const double * , const double * , int) const`

7.35.4.17 `bool Couenne::CouenneProblem::compare(register Node & a, register Node & b) const [inline]`

7.35.4.18 `Nauty* Couenne::CouenneProblem::getNtyInfo() [inline]`

Definition at line 394 of file CouenneProblem.hpp.

7.35.4.19 `void Couenne::CouenneProblem::setupSymmetry()`

empty if no NTY, symmetry data structure setup otherwise

7.35.4.20 `int Couenne::CouenneProblem::evalOrder(int i) const [inline]`

get evaluation order index

Definition at line 403 of file CouenneProblem.hpp.

7.35.4.21 `int* Couenne::CouenneProblem::evalVector() [inline]`

get evaluation order vector (numbering_)

Definition at line 407 of file CouenneProblem.hpp.

7.35.4.22 `CouenneConstraint* Couenne::CouenneProblem::Con(int i) const [inline]`

i-th constraint

Definition at line 411 of file CouenneProblem.hpp.

7.35.4.23 `CouenneObjective* Couenne::CouenneProblem::Obj(int i) const [inline]`

i-th objective

Definition at line 412 of file CouenneProblem.hpp.

7.35.4.24 `exprVar* Couenne::CouenneProblem::Var(int i) const [inline]`

Return pointer to i-th variable.

Definition at line 415 of file CouenneProblem.hpp.

7.35.4.25 `std::vector<exprVar *>& Couenne::CouenneProblem::Variables()` [inline]

Return vector of variables (symbolic representation)

Definition at line 419 of file CouenneProblem.hpp.

7.35.4.26 `std::set<exprAux *, compExpr>*& Couenne::CouenneProblem::AuxSet()` [inline]

Return pointer to set for comparisons.

Definition at line 423 of file CouenneProblem.hpp.

7.35.4.27 `DepGraph* Couenne::CouenneProblem::getDepGraph()` [inline]

Return pointer to dependence graph.

Definition at line 427 of file CouenneProblem.hpp.

7.35.4.28 `Domain* Couenne::CouenneProblem::domain() const` [inline]

return current point & bounds

Definition at line 431 of file CouenneProblem.hpp.

7.35.4.29 `std::vector<expression *>& Couenne::CouenneProblem::commonExprs()` [inline]

Definition at line 434 of file CouenneProblem.hpp.

7.35.4.30 `CouNumber& Couenne::CouenneProblem::X(int i) const` [inline]

x_i

Definition at line 437 of file CouenneProblem.hpp.

7.35.4.31 `CouNumber& Couenne::CouenneProblem::Lb(int i) const` [inline]

lower bound on x_i

Definition at line 438 of file CouenneProblem.hpp.

7.35.4.32 `CouNumber& Couenne::CouenneProblem::Ub(int i) const` [inline]

upper bound on x_i

Definition at line 439 of file CouenneProblem.hpp.

7.35.4.33 `CouNumber* Couenne::CouenneProblem::X() const` [inline]

Return vector of variables.

Definition at line 442 of file CouenneProblem.hpp.

7.35.4.34 `CouNumber* Couenne::CouenneProblem::Lb() const` [inline]

Return vector of lower bounds.

Definition at line 443 of file CouenneProblem.hpp.

7.35.4.35 `CouNumber* Couenne::CouenneProblem::Ub() const` [inline]

Return vector of upper bounds.

Definition at line 444 of file CouenneProblem.hpp.

7.35.4.36 CouNumber*& Couenne::CouenneProblem::bestSol() const [inline]

Best known solution (read from file)

Definition at line 447 of file CouenneProblem.hpp.

7.35.4.37 CouNumber Couenne::CouenneProblem::bestObj() const [inline]

Objective of best known solution.

Definition at line 448 of file CouenneProblem.hpp.

7.35.4.38 bool*& Couenne::CouenneProblem::Commuted() [inline]

Get vector of commuted variables.

Definition at line 451 of file CouenneProblem.hpp.

7.35.4.39 void Couenne::CouenneProblem::addObjective(expression *, const std::string & = "min")

Add (non linear) objective function.

7.35.4.40 void Couenne::CouenneProblem::addEQConstraint(expression *, expression * =NULL)

Add equality constraint $h(x) = b$.

7.35.4.41 void Couenne::CouenneProblem::addGEConstraint(expression *, expression * =NULL)

Add \geq constraint, $h(x) \geq b$.

7.35.4.42 void Couenne::CouenneProblem::addLEConstraint(expression *, expression * =NULL)

Add \leq constraint, $h(x) \leq b$.

7.35.4.43 void Couenne::CouenneProblem::addRNGConstraint(expression * =NULL, expression * =NULL)

Add range constraint, $a \leq h(x) \leq b$.

7.35.4.44 void Couenne::CouenneProblem::setObjective(int indObj = 0, expression * =NULL, const std::string & = "min")

Add (non linear) objective function.

7.35.4.45 expression* Couenne::CouenneProblem::addVariable(bool isint = false, Domain * d =NULL)

Add original variable.

Parameters

<i>isint</i>	if true, this variable is integer, otherwise it is continuous
--------------	---

7.35.4.46 exprAux* Couenne::CouenneProblem::addAuxiliary(expression *)

Add auxiliary variable and associate it with expression given as argument (used in standardization)

7.35.4.47 void Couenne::CouenneProblem::reformulate (CouenneCutGenerator * =NULL)

preprocess problem in order to extract linear relaxations etc.

7.35.4.48 bool Couenne::CouenneProblem::standardize ()

Break problem's nonlinear constraints in simple expressions to be convexified later.

Return true if problem looks feasible, false if proven infeasible.

7.35.4.49 void Couenne::CouenneProblem::print (std::ostream & =std::cout)

Display current representation of problem: objective, linear and nonlinear constraints, and auxiliary variables.

7.35.4.50 bool Couenne::CouenneProblem::doFBBT () const [inline]

shall we do Feasibility Based Bound Tightening?

Definition at line 498 of file CouenneProblem.hpp.

7.35.4.51 bool Couenne::CouenneProblem::doRCBT () const [inline]

shall we do reduced cost Bound Tightening?

Definition at line 499 of file CouenneProblem.hpp.

7.35.4.52 bool Couenne::CouenneProblem::doOBBT () const [inline]

shall we do Optimality Based Bound Tightening?

Definition at line 500 of file CouenneProblem.hpp.

7.35.4.53 bool Couenne::CouenneProblem::doABT () const [inline]

shall we do Aggressive Bound Tightening?

Definition at line 501 of file CouenneProblem.hpp.

7.35.4.54 int Couenne::CouenneProblem::logObbtLev () const [inline]

How often shall we do OBBT?

Definition at line 503 of file CouenneProblem.hpp.

7.35.4.55 int Couenne::CouenneProblem::logAbtLev () const [inline]

How often shall we do ABT?

Definition at line 504 of file CouenneProblem.hpp.

7.35.4.56 void Couenne::CouenneProblem::writeAMPL (const std::string & fname, bool aux)

Write nonlinear problem to a .mod file (with lots of defined variables)

Parameters

<i>fname</i>	Name of the .mod file to be written
<i>aux</i>	controls the use of auxiliaries. If true, a problem is written with auxiliary variables written with their associated expression, i.e. $w_i = h_i(x, y, w)$ and bounds $l_i \leq w_i \leq u_i$, while if false these constraints are written in the form $l_i \leq h_i(x, y) \leq u_i$.

Note: if used before standardization, writes original AMPL formulation

7.35.4.57 void Couenne::CouenneProblem::writeGAMS (const std::string & *fname*)

Write nonlinear problem to a .gms file.

Parameters

<i>fname</i>	Name of the .gams file to be written.
--------------	---------------------------------------

7.35.4.58 void Couenne::CouenneProblem::writeLP (const std::string & *fname*)

Write nonlinear problem to a .lp file.

Note: only works with MIQCQPs (and MISOCPs in the future)

Parameters

<i>fname</i>	Name of the .lp file to be written
--------------	------------------------------------

7.35.4.59 void Couenne::CouenneProblem::initAuxs () const

Initialize auxiliary variables and their bounds from original variables.

7.35.4.60 void Couenne::CouenneProblem::getAuxs (CouNumber *) const

Get auxiliary variables from original variables.

7.35.4.61 bool Couenne::CouenneProblem::boundTightening (t_chg_bounds * , const CglTreeInfo *info*, Bonmin::BabInfo * = NULL) const

tighten bounds using propagation, implied bounds and reduced costs

7.35.4.62 bool Couenne::CouenneProblem::btCore (t_chg_bounds * *chg_bds*) const

core of the bound tightening procedure

7.35.4.63 int Couenne::CouenneProblem::obbt (const CouenneCutGenerator * *cg*, const OsiSolverInterface & *csi*, OsiCuts & *cs*, const CglTreeInfo & *info*, Bonmin::BabInfo * *babInfo*, t_chg_bounds * *chg_bds*)

Optimality Based Bound Tightening.

7.35.4.64 bool Couenne::CouenneProblem::aggressiveBT (Bonmin::OsiTMINLPIInterface * *nlp*, t_chg_bounds * , const CglTreeInfo & *info*, Bonmin::BabInfo * = NULL) const

aggressive bound tightening.

Fake bounds in order to cut portions of the solution space by fathoming on bounds/infeasibility

7.35.4.65 int Couenne::CouenneProblem::redCostBT (const OsiSolverInterface * *psi*, t_chg_bounds * *chg_bds*) const

procedure to strengthen variable bounds.

Return false if problem turns out to be infeasible with given bounds, true otherwise.

7.35.4.66 int Couenne::CouenneProblem::tightenBounds (t_chg_bounds *) const

"Forward" bound tightening, that is, propagate bound of variable x in an expression $w = f(x)$ to the bounds of w .

7.35.4.67 int Couenne::CouenneProblem::impliedBounds (*t_chg_bounds* *) const

"Backward" bound tightening, aka implied bounds.

7.35.4.68 void Couenne::CouenneProblem::fillQuadIndices ()

Look for quadratic terms to be used with SDP cuts.

7.35.4.69 void Couenne::CouenneProblem::fillObjCoeff (double *&)

Fill vector with coefficients of objective function.

7.35.4.70 void Couenne::CouenneProblem::auxiliarize (*exprVar* *, *exprVar* * =NULL)

Replace all occurrences of original variable with new aux given as argument.

7.35.4.71 void Couenne::CouenneProblem::setCutOff (*CouNumber cutoff*, const *CouNumber* * *sol* =NULL) const

Set cutoff.

7.35.4.72 void Couenne::CouenneProblem::resetCutOff (*CouNumber value* =COUENNE_INFINITY) const

Reset cutoff.

7.35.4.73 *CouNumber* Couenne::CouenneProblem::getCutOff () const

Get cutoff.

7.35.4.74 *CouNumber** Couenne::CouenneProblem::getCutOffSol () const

Get cutoff solution.

7.35.4.75 void Couenne::CouenneProblem::installCutOff () const

Make cutoff known to the problem.

7.35.4.76 *ConstJnlstPtr* Couenne::CouenneProblem::Jnlst () const

Provide Journalist.

7.35.4.77 bool Couenne::CouenneProblem::checkNLP (const double * *solution*, double & *obj*, bool *recompute* = false) const

Check if solution is MINLP feasible.

7.35.4.78 int Couenne::CouenneProblem::getIntegerCandidate (const double * *xFrac*, double * *xInt*, double * *lb*, double * *ub*) const

generate integer NLP point Y starting from fractional solution using bound tightening

7.35.4.79 bool Couenne::CouenneProblem::readOptimum (std::string * *fname* =NULL)

Read best known solution from file given in argument.

7.35.4.80 static void Couenne::CouenneProblem::registerOptions (Iopt::SmartPtr< Bonmin::RegisteredOptions > *options*) [static]

Add list of options to be read from file.

7.35.4.81 `exprAux* Couenne::CouenneProblem::linStandardize (bool addAux, CouNumber c0, LinMap & lmap, QuadMap & qmap)`

standardization of linear `exprOp`'s

7.35.4.82 `int Couenne::CouenneProblem::splitAux (CouNumber , expression * , expression *& , bool * , enum expression::auxSign &)`

split a constraint $w - f(x) = c$ into w 's index (it is returned) and rest = $f(x) + c$

7.35.4.83 `void Couenne::CouenneProblem::indcoe2vector (int * indexL, CouNumber * coeff, std::vector< std::pair< exprVar *, CouNumber >> & lcoeff)`

translates pair (indices, coefficients) into vector with pointers to variables

7.35.4.84 `void Couenne::CouenneProblem::indcoe2vector (int * indexI, int * indexJ, CouNumber * coeff, std::vector< quadElem >& qcoeff)`

translates triplet (indicesI, indicesJ, coefficients) into vector with pointers to variables

7.35.4.85 `void Couenne::CouenneProblem::decomposeTerm (expression * term, CouNumber initCoe, CouNumber & c0, LinMap & lmap, QuadMap & qmap)`

given (expression *) element of sum, returns (coe,ind0,ind1) depending on element:

1) $a * x_i^2 \rightarrow (a,i,?)$ return COU_EXPRPOW 2) $a * x_i \rightarrow (a,i,?)$ return COU_EXPRVAR 3) $a * x_i * x_j \rightarrow (a,i,j)$ return COU_EXPRMUL 4) $a \rightarrow (a,?,?)$ return COU_EXPRCONST

x_i and/or x_j may come from standardizing other (linear or quadratic operator) sub-expressions

7.35.4.86 `const std::string& Couenne::CouenneProblem::problemName () const [inline]`

return problem name

Definition at line 654 of file CouenneProblem.hpp.

7.35.4.87 `void Couenne::CouenneProblem::setProblemName (std::string & problemName_) [inline]`

Definition at line 657 of file CouenneProblem.hpp.

7.35.4.88 `const std::vector<std::set<int>>& Couenne::CouenneProblem::Dependence () const [inline]`

return inverse dependence structure

Definition at line 661 of file CouenneProblem.hpp.

7.35.4.89 `const std::vector<CouenneObject *>& Couenne::CouenneProblem::Objects () const [inline]`

return object vector

Definition at line 665 of file CouenneProblem.hpp.

7.35.4.90 `int Couenne::CouenneProblem::findSOS (CbcModel * CbcModelPtr, OsiSolverInterface * solver, OsiObject ** objects)`

find SOS constraints in problem

7.35.4.91 `void Couenne::CouenneProblem::setMaxCpuTime (double time) [inline]`

set maximum CPU time

Definition at line 674 of file CouenneProblem.hpp.

7.35.4.92 double Couenne::CouenneProblem::getMaxCpuTime() const [inline]

return maximum CPU time

Definition at line 678 of file CouenneProblem.hpp.

7.35.4.93 void Couenne::CouenneProblem::setBase(Bonmin::BabSetupBase * base)

save CouenneBase

7.35.4.94 void Couenne::CouenneProblem::createUnusedOriginals()

Some originals may be unused due to their zero multiplicity (that happens when they are duplicates).

This procedure creates a structure for quickly checking and restoring their value after solving.

7.35.4.95 void Couenne::CouenneProblem::restoreUnusedOriginals(CouNumber * =NULL) const

Some originals may be unused due to their zero multiplicity (that happens when they are duplicates).

This procedure restores their value after solving

7.35.4.96 int* Couenne::CouenneProblem::unusedOriginalsIndices() [inline]

return indices of neglected redundant variables

Definition at line 696 of file CouenneProblem.hpp.

7.35.4.97 int Couenne::CouenneProblem::nUnusedOriginals() [inline]

number of unused originals

Definition at line 700 of file CouenneProblem.hpp.

7.35.4.98 enum multiSep Couenne::CouenneProblem::MultilinSep() const [inline]

return type of separator for multilinear terms

Definition at line 704 of file CouenneProblem.hpp.

7.35.4.99 bool Couenne::CouenneProblem::fbbtReachedIterLimit() const [inline]

true if latest call to FBBT terminated due to iteration limit reached

Definition at line 708 of file CouenneProblem.hpp.

7.35.4.100 bool Couenne::CouenneProblem::orbitalBranching() const [inline]

return true if orbital branching activated

Definition at line 712 of file CouenneProblem.hpp.

7.35.4.101 void Couenne::CouenneProblem::setCheckAuxBounds(bool value) [inline]

set the value for checkAuxBounds.

When true, all MINLP feasible solutions will additionally be tested for feasibility with respect to auxiliary variable bounds.
This is normally not needed.

Definition at line 718 of file CouenneProblem.hpp.

7.35.4.102 bool Couenne::CouenneProblem::checkAuxBounds() const [inline]

return true if bounds of auxiliary variables have to be satisfied whenever a solution is tested for MINLP feasibility
Definition at line 723 of file CouenneProblem.hpp.

7.35.4.103 enum TrilinDecompType Couenne::CouenneProblem::getTrilinDecompType() [inline]

return type of decomposition of quadrilinear terms
Definition at line 727 of file CouenneProblem.hpp.

7.35.4.104 Bonmin::BabSetupBase* Couenne::CouenneProblem::bonBase() const [inline]

options

Definition at line 731 of file CouenneProblem.hpp.

7.35.4.105 double Couenne::CouenneProblem::constObjVal() const [inline]

returns constant objective value if it contains no variables

Definition at line 734 of file CouenneProblem.hpp.

7.35.4.106 CouenneSdpCuts* Couenne::CouenneProblem::getSdpCutGen() [inline]

Returns pointer to sdp cut generator.

Definition at line 737 of file CouenneProblem.hpp.

7.35.4.107 int Couenne::CouenneProblem::fake_tighten(char direction, int index, const double * X, CouNumber * olb, CouNumber * oub, t_chg_bounds * chg_bds, t_chg_bounds * f_chg) const [protected]

single fake tightening.

Return

-1 if infeasible 0 if no improvement +1 if improved

Parameters

<i>direction</i>	0: left, 1: right
<i>index</i>	index of the variable tested
<i>X</i>	point round which tightening is done
<i>olb</i>	cur. lower bound
<i>oub</i>	cur. upper bound

7.35.4.108 int Couenne::CouenneProblem::obbtInner(OsiSolverInterface *, OsiCuts &, t_chg_bounds *, Bonmin::BabInfo *) const [protected]

Optimality Based Bound Tightening – inner loop.

7.35.4.109 int Couenne::CouenneProblem::obbt_iter(OsiSolverInterface * csi, t_chg_bounds * chg_bds, const CoinWarmStart * warmstart, Bonmin::BabInfo * babInfo, double * objco, int sense, int index) const [protected]

7.35.4.110 int Couenne::CouenneProblem::call_iter(OsiSolverInterface * csi, t_chg_bounds * chg_bds, const CoinWarmStart * warmstart, Bonmin::BabInfo * babInfo, double * objco, enum nodeType type, int sense) const [protected]

7.35.4.111 void Couenne::CouenneProblem::analyzeSparsity (**CouNumber** , **LinMap &** , **QuadMap &**) [protected]

analyze sparsity of potential exprQuad(exprGroup and change linear/quadratic maps accordingly, if necessary by adding new auxiliary variables and including them in the linear map

7.35.4.112 void Couenne::CouenneProblem::flattenMul (**expression * mul** , **CouNumber & coe** , **std::map< int, CouNumber > & indices**) [protected]

re-organizes multiplication and stores indices (and exponents) of its variables

7.35.4.113 void Couenne::CouenneProblem::realign () [protected]

clear all spurious variables pointers not referring to the variables_ vector

7.35.4.114 void Couenne::CouenneProblem::fillDependence (**Bonmin::BabSetupBase * base** , **CouenneCutGenerator * = NULL**) [protected]

fill dependence_ structure

7.35.4.115 void Couenne::CouenneProblem::fillIntegerRank () const [protected]

fill freeIntegers_ array

7.35.4.116 int Couenne::CouenneProblem::testIntFix (**int index** , **CouNumber xFrac** , **enum fixType * fixed** , **CouNumber * xInt** , **CouNumber * dualL** , **CouNumber * dualR** , **CouNumber * olb** , **CouNumber * oub** , **bool patient**) const [protected]

Test fixing of an integer variable (used in [getIntegerCandidate\(\)](#))

7.35.4.117 int Couenne::CouenneProblem::getLastPrioSort () const [inline]

Definition at line 810 of file CouenneProblem.hpp.

7.35.4.118 void Couenne::CouenneProblem::setLastPrioSort (**int givenLastPS**)

7.35.4.119 **CouenneRecordBestSol*** Couenne::CouenneProblem::getRecordBestSol () const [inline]

returns recorded best solution

Definition at line 817 of file CouenneProblem.hpp.

7.35.4.120 double Couenne::CouenneProblem::getFeasTol () [inline]

returns feasibility tolerance

Definition at line 821 of file CouenneProblem.hpp.

7.35.4.121 double Couenne::CouenneProblem::checkObj (**const CouNumber * sol** , **const double & precision**) const

Recompute objective value for sol.

7.35.4.122 bool Couenne::CouenneProblem::checkInt (**const CouNumber * sol** , **const int from** , **const int upto** , **const std::vector< int > listInt** , **const bool origVarOnly** , **const bool stopAtFirstViol** , **const double precision** , **double & maxViol**) const

check integrality of vars in sol with index between from and upto (original vars only if origVarOnly == true); return true if all integer vars are within precision of an integer value

7.35.4.123 `bool Couenne::CouenneProblem::checkBounds (const CouNumber * sol, const bool stopAtFirstViol, const double precision, double & maxViol) const`

Check bounds; returns true iff feasible for given precision.

7.35.4.124 `bool Couenne::CouenneProblem::checkAux (const CouNumber * sol, const bool stopAtFirstViol, const double precision, double & maxViol) const`

returns true iff value of all auxiliaries are within bounds

7.35.4.125 `bool Couenne::CouenneProblem::checkCons (const CouNumber * sol, const bool stopAtFirstViol, const double precision, double & maxViol) const`

returns true iff value of all auxiliaries are within bounds

7.35.4.126 `bool Couenne::CouenneProblem::checkNLP2 (const double * solution, const double obj, const bool careAboutObj, const bool stopAtFirstViol, const bool checkAll, const double precision) const`

Return true if either solution or recomputed_solution obtained using `getAuxs()` from the original variables in solution is feasible within precision (the solution with minimum violation is then stored in recBSol->modSol, as well as its value and violation); return false otherwise.

If stopAtFirstViol == true, recBSol->modSol is meaningless upon return. If stopAtFirstViol == false, recBSol->modSol contains the solution with minimum violation, although this violation might be larger than precision. This is useful for cases where the current solution must be considered valid (e.g., because Cbc is going to accept it anyway), although it violates precision requirements. Value of obj matters only if careAboutObj == true; the code then tries to balance violation of constraints and value of objective. if checkAll = false, check only integrality/bounds for original vars and constraints; consider only recomputed_sol if checkAll == true, check also integrality/bounds on auxs; consider both recomputed_sol and solution if careAboutObj is set to true, then stopAtFirstViol must be set to false too.

7.35.4.127 `bool Couenne::CouenneProblem::checkNLP0 (const double * solution, double & obj, bool recompute_obj = false, const bool careAboutObj = false, const bool stopAtFirstViol = true, const bool checkAll = false, const double precision = -1) const`

And finally a method to get both.

7.35.4.128 `std::vector<CouenneConstraint *>* Couenne::CouenneProblem::ConstraintClass (const char * str) [inline]`

return particular constraint class.

Classes:

1) "convex": convex constraints; 2) "PSDcon": constraints of the form X 0 3) "normal": regular constraints

Definition at line 896 of file CouenneProblem.hpp.

7.35.5 Friends And Related Function Documentation

7.35.5.1 `friend class exprMul [friend]`

Definition at line 171 of file CouenneProblem.hpp.

7.35.6 Member Data Documentation

7.35.6.1 `int Couenne::CouenneProblem::minDepthPrint_`

Definition at line 182 of file CouenneProblem.hpp.

7.35.6.2 `int Couenne::CouenneProblem::minNodePrint_`

Definition at line 185 of file CouenneProblem.hpp.

7.35.6.3 `bool Couenne::CouenneProblem::doPrint_`

Definition at line 188 of file CouenneProblem.hpp.

7.35.6.4 `std::string Couenne::CouenneProblem::problemName_ [protected]`

problem name

Definition at line 193 of file CouenneProblem.hpp.

7.35.6.5 `std::vector<exprVar *> Couenne::CouenneProblem::variables_ [protected]`

Variables (original, auxiliary, and defined)

Definition at line 195 of file CouenneProblem.hpp.

7.35.6.6 `std::vector<CouenneObjective *> Couenne::CouenneProblem::objectives_ [protected]`

Objectives.

Definition at line 196 of file CouenneProblem.hpp.

7.35.6.7 `std::vector<CouenneConstraint *> Couenne::CouenneProblem::constraints_ [protected]`

Constraints.

Definition at line 197 of file CouenneProblem.hpp.

7.35.6.8 `std::vector<expression *> Couenne::CouenneProblem::commonexprs_ [protected]`

AMPL's common expressions (read from AMPL through structures cexprs and cexprs1)

Definition at line 200 of file CouenneProblem.hpp.

7.35.6.9 `Domain Couenne::CouenneProblem::domain_ [mutable], [protected]`

current point and bounds;

Definition at line 202 of file CouenneProblem.hpp.

7.35.6.10 `std::set<exprAux *, compExpr>* Couenne::CouenneProblem::auxSet_ [protected]`

Expression map for comparison in standardization and to count occurrences of an auxiliary.

Definition at line 206 of file CouenneProblem.hpp.

7.35.6.11 `int Couenne::CouenneProblem::curnvars_ [mutable], [protected]`

Number of elements in the x_, lb_, ub_ arrays.

Definition at line 209 of file CouenneProblem.hpp.

7.35.6.12 `int Couenne::CouenneProblem::nIntVars_` [protected]

Number of discrete variables.

Definition at line 212 of file CouenneProblem.hpp.

7.35.6.13 `CouNumber* Couenne::CouenneProblem::optimum_` [mutable], [protected]

Best solution known to be loaded from file – for testing purposes.

Definition at line 215 of file CouenneProblem.hpp.

7.35.6.14 `CouNumber Couenne::CouenneProblem::bestObj_` [protected]

Best known objective function.

Definition at line 218 of file CouenneProblem.hpp.

7.35.6.15 `bool* Couenne::CouenneProblem::commuted_` [protected]

Variables that have commuted to auxiliary.

Definition at line 221 of file CouenneProblem.hpp.

7.35.6.16 `int* Couenne::CouenneProblem::numbering_` [protected]

numbering of variables.

No variable x_i with associated $p_i(i)$ greater than $p_i(j)$ should be evaluated before variable x_j

Definition at line 225 of file CouenneProblem.hpp.

7.35.6.17 `int Couenne::CouenneProblem::ndefined_` [protected]

Number of "defined variables" (aka "common expressions")

Definition at line 228 of file CouenneProblem.hpp.

7.35.6.18 `DepGraph* Couenne::CouenneProblem::graph_` [protected]

Dependence (acyclic) graph: shows dependence of all auxiliary variables on one another and on original variables.

Used to create a numbering of all variables for evaluation and bound tightening (reverse order for implied bounds)

Definition at line 234 of file CouenneProblem.hpp.

7.35.6.19 `int Couenne::CouenneProblem::nOrigVars_` [protected]

Number of original variables.

Definition at line 237 of file CouenneProblem.hpp.

7.35.6.20 `int Couenne::CouenneProblem::nOrigCons_` [protected]

Number of original constraints (disregarding those that turned into auxiliary variable definition)

Definition at line 241 of file CouenneProblem.hpp.

7.35.6.21 `int Couenne::CouenneProblem::nOrigIntVars_` [protected]

Number of original integer variables.

Definition at line 244 of file CouenneProblem.hpp.

7.35.6.22 **GlobalCutOff* Couenne::CouenneProblem::pcutoff_** [mutable], [protected]

Pointer to a global cutoff object.

Definition at line 247 of file CouenneProblem.hpp.

7.35.6.23 **bool Couenne::CouenneProblem::created_pcutoff_** [mutable], [protected]

flag indicating if this class is creator of global cutoff object

Definition at line 250 of file CouenneProblem.hpp.

7.35.6.24 **bool Couenne::CouenneProblem::doFBBT_** [protected]

do Feasibility-based bound tightening

Definition at line 252 of file CouenneProblem.hpp.

7.35.6.25 **bool Couenne::CouenneProblem::doRCBT_** [protected]

do reduced cost bound tightening

Definition at line 253 of file CouenneProblem.hpp.

7.35.6.26 **bool Couenne::CouenneProblem::doOBBT_** [protected]

do Optimality-based bound tightening

Definition at line 254 of file CouenneProblem.hpp.

7.35.6.27 **bool Couenne::CouenneProblem::doABT_** [protected]

do Aggressive bound tightening

Definition at line 255 of file CouenneProblem.hpp.

7.35.6.28 **int Couenne::CouenneProblem::logObbtLev_** [protected]

frequency of Optimality-based bound tightening

Definition at line 257 of file CouenneProblem.hpp.

7.35.6.29 **int Couenne::CouenneProblem::logAbtLev_** [protected]

frequency of Aggressive bound tightening

Definition at line 258 of file CouenneProblem.hpp.

7.35.6.30 **JnIstPtr Couenne::CouenneProblem::jnIst_** [protected]

SmartPointer to the Journalist.

Definition at line 261 of file CouenneProblem.hpp.

7.35.6.31 **CouNumber Couenne::CouenneProblem::opt_window_** [protected]

window around known optimum (for testing purposes)

Definition at line 264 of file CouenneProblem.hpp.

7.35.6.32 `bool Couenne::CouenneProblem::useQuadratic_ [protected]`

Use quadratic expressions?

Definition at line 267 of file CouenneProblem.hpp.

7.35.6.33 `CouNumber Couenne::CouenneProblem::feas_tolerance_ [protected]`

feasibility tolerance (to be used in checkNLP)

Definition at line 270 of file CouenneProblem.hpp.

7.35.6.34 `std::vector<std::set<int>> Couenne::CouenneProblem::dependence_ [protected]`

inverse dependence structure: for each variable x give set of auxiliary variables (or better, their indices) whose expression depends on x

Definition at line 275 of file CouenneProblem.hpp.

7.35.6.35 `std::vector<CouenneObject*> Couenne::CouenneProblem::objects_ [protected]`

vector of pointer to CouenneObjects.

Used by CouenneVarObjects when finding all objects related to (having as argument) a single variable

Definition at line 280 of file CouenneProblem.hpp.

7.35.6.36 `int* Couenne::CouenneProblem::integerRank_ [mutable], [protected]`

each element is true if variable is integer and, if auxiliary, depends on no integer

Definition at line 284 of file CouenneProblem.hpp.

7.35.6.37 `std::vector<int> Couenne::CouenneProblem::numberInRank_ [mutable], [protected]`

numberInRank_[i] is the number of integer variables in rank i

Definition at line 287 of file CouenneProblem.hpp.

7.35.6.38 `double Couenne::CouenneProblem::maxCpuTime_ [protected]`

maximum cpu time

Definition at line 290 of file CouenneProblem.hpp.

7.35.6.39 `Bonmin::BabSetupBase* Couenne::CouenneProblem::bonBase_ [protected]`

options

Definition at line 293 of file CouenneProblem.hpp.

7.35.6.40 `ASL* Couenne::CouenneProblem::asl_ [protected]`

AMPL structure pointer (temporary — looking forward to embedding into OS...)

Definition at line 296 of file CouenneProblem.hpp.

7.35.6.41 `int* Couenne::CouenneProblem::unusedOriginalsIndices_ [protected]`

some originals may be unused due to their zero multiplicity (that happens when they are duplicates).

This array keeps track of their indices and is sorted by evaluation order

Definition at line 301 of file CouenneProblem.hpp.

7.35.6.42 int Couenne::CouenneProblem::nUnusedOriginals_ [protected]

number of unused originals

Definition at line 304 of file CouenneProblem.hpp.

7.35.6.43 int Couenne::CouenneProblem::lastPrioSort_ [protected]

Definition at line 307 of file CouenneProblem.hpp.

7.35.6.44 CouenneRecordBestSol* Couenne::CouenneProblem::recBSol [protected]

Definition at line 310 of file CouenneProblem.hpp.

7.35.6.45 enum multiSep Couenne::CouenneProblem::multilinSep_ [protected]

Type of Multilinear separation.

Definition at line 313 of file CouenneProblem.hpp.

7.35.6.46 int Couenne::CouenneProblem::max_fbbt_iter_ [protected]

number of FBBT iterations

Definition at line 316 of file CouenneProblem.hpp.

7.35.6.47 bool Couenne::CouenneProblem::fbbtReachedIterLimit_ [mutable], [protected]

true if FBBT exited for iteration limits as opposed to inability to further tighten bounds

Definition at line 320 of file CouenneProblem.hpp.

7.35.6.48 bool Couenne::CouenneProblem::orbitalBranching_ [protected]

use orbital branching?

Definition at line 323 of file CouenneProblem.hpp.

7.35.6.49 bool Couenne::CouenneProblem::checkAuxBounds_ [protected]

check bounds on auxiliary variables when verifying MINLP feasibility of a solution.

Usually this is not needed, unless some manipulation on auxiliary variables is done before Branch-and-Bound

Definition at line 329 of file CouenneProblem.hpp.

7.35.6.50 enum TrilinDecompType Couenne::CouenneProblem::trilinDecompType_ [protected]

return type of decomposition of quadrilinear terms

Definition at line 332 of file CouenneProblem.hpp.

7.35.6.51 double Couenne::CouenneProblem::constObjVal_ [protected]

constant value of the objective if no variable is declared in it

Definition at line 335 of file CouenneProblem.hpp.

7.35.6.52 CouenneBTPerfIndicator* Couenne::CouenneProblem::perfIndicator_ [protected]

Performance indicator for FBBT – to be moved away from [CouenneProblem](#) when we do it with FBBT.

Definition at line 339 of file CouenneProblem.hpp.

7.35.6.53 std::map<const char *, std::vector <CouenneConstraint *> *, less_than_str> Couenne::CouenneProblem::ConstraintClass_ [protected]

Return particular constraint class.

Classes:

1) "convex": convex constraints; 2) "PSDcon": constraints of the form $X \geq 0$ 3) "normal": regular constraints

Definition at line 346 of file CouenneProblem.hpp.

7.35.6.54 CouenneSdpCuts* Couenne::CouenneProblem::sdpCutGen_ [protected]

Temporary pointer to SDP cut generator.

A little dirty as it is generated DURING standardization, but necessary to avoid meddling with different spaces

Definition at line 351 of file CouenneProblem.hpp.

7.35.6.55 std::vector<Node> Couenne::CouenneProblem::node_info [mutable]

Definition at line 383 of file CouenneProblem.hpp.

7.35.6.56 Nauty* Couenne::CouenneProblem::nauty_info [mutable]

Definition at line 384 of file CouenneProblem.hpp.

7.35.6.57 myclass0 Couenne::CouenneProblem::node_sort

Definition at line 386 of file CouenneProblem.hpp.

7.35.6.58 myclass Couenne::CouenneProblem::index_sort

Definition at line 387 of file CouenneProblem.hpp.

The documentation for this class was generated from the following file:

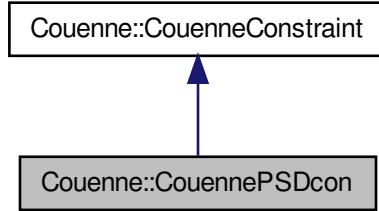
- /home/ted/COIN/trunk/Couenne/src/problem/[CouenneProblem.hpp](#)

7.36 Couenne::CouennePSDcon Class Reference

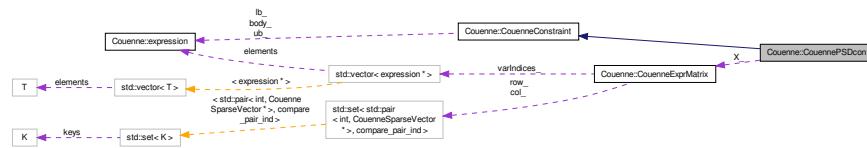
Class to represent positive semidefinite constraints ///////////////.

```
#include <CouennePSDcon.hpp>
```

Inheritance diagram for Couenne::CouennePSDcon:



Collaboration diagram for Couenne::CouennePSDcon:



Public Member Functions

- [CouennePSDcon \(CouenneExprMatrix *X\)](#)
Constructor.
- [~CouennePSDcon \(\)](#)
Destructor.
- [CouennePSDcon \(const CouennePSDcon &c, Domain *d=NULL\)](#)
Copy constructor.
- [CouennePSDcon & operator= \(const CouennePSDcon &c\)](#)
Assignment operator.
- [CouenneConstraint * clone \(Domain *d=NULL\) const](#)
Cloning method.
- [CouenneExprMatrix * getX \(\) const](#)
return X
- [exprAux * standardize \(CouenneProblem *\)](#)
Decompose body of constraint through auxiliary variables.
- [void print \(std::ostream &=std::cout\)](#)
Print constraint.

Protected Attributes

- [CouenneExprMatrix * X_](#)
contains indices of matrix X 0

7.36.1 Detailed Description

Class to represent positive semidefinite constraints //////////////.

Definition at line 24 of file CouennePSDcon.hpp.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 Couenne::CouennePSDcon::CouennePSDcon (CouenneExprMatrix * X) [inline]

Constructor.

Definition at line 33 of file CouennePSDcon.hpp.

7.36.2.2 Couenne::CouennePSDcon::~CouennePSDcon ()

Destructor.

7.36.2.3 Couenne::CouennePSDcon::CouennePSDcon (const CouennePSDcon & c, Domain * d = NULL)

Copy constructor.

7.36.3 Member Function Documentation

7.36.3.1 CouennePSDcon& Couenne::CouennePSDcon::operator= (const CouennePSDcon & c)

Assignment operator.

7.36.3.2 CouenneConstraint* Couenne::CouennePSDcon::clone (Domain * d = NULL) const [inline], [virtual]

Cloning method.

Reimplemented from [Couenne::CouenneConstraint](#).

Definition at line 47 of file CouennePSDcon.hpp.

7.36.3.3 CouenneExprMatrix* Couenne::CouennePSDcon::getX () const [inline]

return X

Definition at line 51 of file CouennePSDcon.hpp.

7.36.3.4 exprAux* Couenne::CouennePSDcon::standardize (CouenneProblem *) [virtual]

Decompose body of constraint through auxiliary variables.

Reimplemented from [Couenne::CouenneConstraint](#).

7.36.3.5 void Couenne::CouennePSDcon::print (std::ostream & = std::cout) [virtual]

Print constraint.

Reimplemented from [Couenne::CouenneConstraint](#).

7.36.4 Member Data Documentation

7.36.4.1 CouenneExprMatrix* Couenne::CouennePSDcon::X_ [protected]

contains indices of matrix X 0

Definition at line 28 of file CouennePSDcon.hpp.

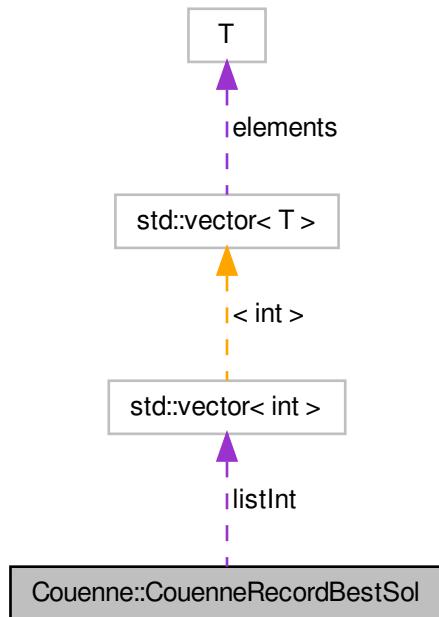
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouennePSDcon.hpp

7.37 Couenne::CouenneRecordBestSol Class Reference

```
#include <CouenneRecordBestSol.hpp>
```

Collaboration diagram for Couenne::CouenneRecordBestSol:



Public Member Functions

- `CouenneRecordBestSol ()`
Constructor.
- `CouenneRecordBestSol (const CouenneRecordBestSol &other)`
Copy constructor.
- `~CouenneRecordBestSol ()`
Destructor.
- int `getCardInitDom () const`

- `bool * getInitIsInt () const`
- `std::vector< int > getListInt () const`
- `void setInitIsInt (const bool *givenIsInt, const int givenCard)`
- `CouNumber * getInitDomLb () const`
- `void setInitDomLb (const CouNumber *givenLb, const int givenCard)`
- `CouNumber * getInitDomUb () const`
- `void setInitDomUb (const CouNumber *givenUb, const int givenCard)`
- `void setHasSol (const bool givenHasSol)`
- `bool getHasSol () const`
- `void setSol (const double *givenSol, const int givenCard, const double givenMaxViol)`
- `int getCardSol () const`
- `void setCardSol (const int givenCard)`
- `double * getSol () const`
- `double getMaxViol () const`
- `void setVal (const double givenVal)`
- `double getVal ()`
- `void update (const double *givenSol, const int givenCard, const double givenVal, const double givenMaxViol)`
- `void update ()`
- `int compareAndSave (const double *solA, const double solAVal, const double solAMaxViol, const bool solAIsFeas, const double *solB, const double solBVal, const double solBMaxViol, const bool solBIsFeas, const int cardSol, const double precision)`
- `int getCardModSol () const`
- `double * getModSol (const int expectedCard)`
- `double getModSolVal () const`
- `double getModSolMaxViol () const`
- `void setModSol (const double *givenModSol, const int givenModCard, const double givenModVal, const double givenModMaxViol)`
- `void printSol (FILE *fsol) const`

Public Attributes

- `int cardInitDom`
- `bool * initIsInt`
- `std::vector< int > listInt`
- `CouNumber * initDomLb`
- `CouNumber * initDomUb`
- `bool hasSol`
- `int cardSol`
- `double * sol`
- `double val`
- `double maxViol`
- `int cardModSol`
- `double * modSol`
- `double modSolVal`
- `double modSolMaxViol`

7.37.1 Detailed Description

Definition at line 19 of file CouenneRecordBestSol.hpp.

7.37.2 Constructor & Destructor Documentation

7.37.2.1 `Couenne::CouenneRecordBestSol::CouenneRecordBestSol()`

Constructor.

7.37.2.2 `Couenne::CouenneRecordBestSol::CouenneRecordBestSol(const CouenneRecordBestSol & other)`

Copy constructor.

7.37.2.3 `Couenne::CouenneRecordBestSol::~CouenneRecordBestSol()`

Destructor.

7.37.3 Member Function Documentation

7.37.3.1 `int Couenne::CouenneRecordBestSol::getCardInitDom() const [inline]`

Definition at line 61 of file CouenneRecordBestSol.hpp.

7.37.3.2 `bool* Couenne::CouenneRecordBestSol::getInitIsInt() const [inline]`

Definition at line 62 of file CouenneRecordBestSol.hpp.

7.37.3.3 `std::vector<int> Couenne::CouenneRecordBestSol::getListInt() const [inline]`

Definition at line 63 of file CouenneRecordBestSol.hpp.

7.37.3.4 `void Couenne::CouenneRecordBestSol::setInitIsInt(const bool * givenIsInt, const int givenCard)`

7.37.3.5 `CouNumber* Couenne::CouenneRecordBestSol::getInitDomLb() const [inline]`

Definition at line 66 of file CouenneRecordBestSol.hpp.

7.37.3.6 `void Couenne::CouenneRecordBestSol::setInitDomLb(const CouNumber * givenLb, const int givenCard)`

7.37.3.7 `CouNumber* Couenne::CouenneRecordBestSol::getInitDomUb() const [inline]`

Definition at line 68 of file CouenneRecordBestSol.hpp.

7.37.3.8 `void Couenne::CouenneRecordBestSol::setInitDomUb(const CouNumber * givenUb, const int givenCard)`

7.37.3.9 `void Couenne::CouenneRecordBestSol::setHasSol(const bool givenHasSol)`

7.37.3.10 `bool Couenne::CouenneRecordBestSol::getHasSol() const [inline]`

Definition at line 72 of file CouenneRecordBestSol.hpp.

7.37.3.11 `void Couenne::CouenneRecordBestSol::setSol(const double * givenSol, const int givenCard, const double givenMaxViol)`

7.37.3.12 `int Couenne::CouenneRecordBestSol::getCardSol() const [inline]`

Definition at line 75 of file CouenneRecordBestSol.hpp.

7.37.3.13 `void Couenne::CouenneRecordBestSol::setCardSol(const int givenCard)`

7.37.3.14 `double* Couenne::CouenneRecordBestSol::getSol() const [inline]`

Definition at line 77 of file CouenneRecordBestSol.hpp.

7.37.3.15 `double Couenne::CouenneRecordBestSol::getMaxViol() const [inline]`

Definition at line 78 of file CouenneRecordBestSol.hpp.

7.37.3.16 `void Couenne::CouenneRecordBestSol::setVal(const double givenVal)`

7.37.3.17 `double Couenne::CouenneRecordBestSol::getVal() [inline]`

Definition at line 80 of file CouenneRecordBestSol.hpp.

7.37.3.18 `void Couenne::CouenneRecordBestSol::update(const double * givenSol, const int givenCard, const double givenVal, const double givenMaxViol)`

7.37.3.19 `void Couenne::CouenneRecordBestSol::update()`

7.37.3.20 `int Couenne::CouenneRecordBestSol::compareAndSave(const double * solA, const double solAVal, const double solAMaxViol, const bool solAIsFeas, const double * solB, const double solBVal, const double solBMaxViol, const bool solBIsFeas, const int cardSol, const double precision)`

7.37.3.21 `int Couenne::CouenneRecordBestSol::getCardModSol() const [inline]`

Definition at line 103 of file CouenneRecordBestSol.hpp.

7.37.3.22 `double* Couenne::CouenneRecordBestSol::getModSol(const int expectedCard)`

7.37.3.23 `double Couenne::CouenneRecordBestSol::getModSolVal() const [inline]`

Definition at line 105 of file CouenneRecordBestSol.hpp.

7.37.3.24 `double Couenne::CouenneRecordBestSol::getModSolMaxViol() const [inline]`

Definition at line 106 of file CouenneRecordBestSol.hpp.

7.37.3.25 `void Couenne::CouenneRecordBestSol::setModSol(const double * givenModSol, const int givenModCard, const double givenModVal, const double givenModMaxViol)`

7.37.3.26 `void Couenne::CouenneRecordBestSol::printSol(FILE * fsol) const`

7.37.4 Member Data Documentation

7.37.4.1 `int Couenne::CouenneRecordBestSol::cardInitDom`

Definition at line 24 of file CouenneRecordBestSol.hpp.

7.37.4.2 `bool* Couenne::CouenneRecordBestSol::initLsInt`

Definition at line 26 of file CouenneRecordBestSol.hpp.

7.37.4.3 `std::vector<int> Couenne::CouenneRecordBestSol::listInt`

Definition at line 28 of file CouenneRecordBestSol.hpp.

7.37.4.4 CouNumber* Couenne::CouenneRecordBestSol::initDomLb

Definition at line 30 of file CouenneRecordBestSol.hpp.

7.37.4.5 CouNumber* Couenne::CouenneRecordBestSol::initDomUb

Definition at line 32 of file CouenneRecordBestSol.hpp.

7.37.4.6 bool Couenne::CouenneRecordBestSol::hasSol

Definition at line 35 of file CouenneRecordBestSol.hpp.

7.37.4.7 int Couenne::CouenneRecordBestSol::cardSol

Definition at line 37 of file CouenneRecordBestSol.hpp.

7.37.4.8 double* Couenne::CouenneRecordBestSol::sol

Definition at line 39 of file CouenneRecordBestSol.hpp.

7.37.4.9 double Couenne::CouenneRecordBestSol::val

Definition at line 41 of file CouenneRecordBestSol.hpp.

7.37.4.10 double Couenne::CouenneRecordBestSol::maxViol

Definition at line 43 of file CouenneRecordBestSol.hpp.

7.37.4.11 int Couenne::CouenneRecordBestSol::cardModSol

Definition at line 46 of file CouenneRecordBestSol.hpp.

7.37.4.12 double* Couenne::CouenneRecordBestSol::modSol

Definition at line 47 of file CouenneRecordBestSol.hpp.

7.37.4.13 double Couenne::CouenneRecordBestSol::modSolVal

Definition at line 48 of file CouenneRecordBestSol.hpp.

7.37.4.14 double Couenne::CouenneRecordBestSol::modSolMaxViol

Definition at line 49 of file CouenneRecordBestSol.hpp.

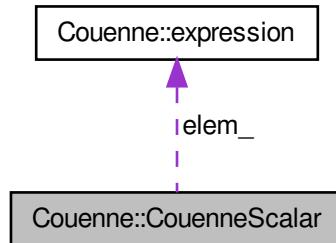
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/problem/[CouenneRecordBestSol.hpp](#)

7.38 Couenne::CouenneScalar Class Reference

```
#include <CouenneMatrix.hpp>
```

Collaboration diagram for Couenne::CouenneScalar:



Public Member Functions

- `CouenneScalar (int index, expression *elem)`
- `~CouenneScalar ()`
- `CouenneScalar (const CouenneScalar &rhs)`
- `CouenneScalar & operator= (const CouenneScalar &rhs)`
- `CouenneScalar * clone ()`
- `int getIndex () const`
- `expression * getElem () const`
- `bool operator< (const CouenneScalar &rhs) const`
- `void print () const`

Protected Attributes

- `int index_`
index of element in vector
- `expression * elem_`
element

Friends

- `bool operator< (const CouenneScalar &first, const CouenneScalar &second)`

7.38.1 Detailed Description

Definition at line 25 of file CouenneMatrix.hpp.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 Couenne::CouenneScalar::CouenneScalar (int *index*, expression * *elem*) [inline]

Definition at line 34 of file CouenneMatrix.hpp.

7.38.2.2 `Couenne::CouenneScalar::~CouenneScalar()`

7.38.2.3 `Couenne::CouenneScalar::CouenneScalar(const CouenneScalar & rhs)` [inline]

Definition at line 40 of file CouenneMatrix.hpp.

7.38.3 Member Function Documentation

7.38.3.1 `CouenneScalar& Couenne::CouenneScalar::operator=(const CouenneScalar & rhs)` [inline]

Definition at line 44 of file CouenneMatrix.hpp.

7.38.3.2 `CouenneScalar* Couenne::CouenneScalar::clone()` [inline]

Definition at line 50 of file CouenneMatrix.hpp.

7.38.3.3 `int Couenne::CouenneScalar::getIndex()` const [inline]

Definition at line 52 of file CouenneMatrix.hpp.

7.38.3.4 `expression* Couenne::CouenneScalar::getElem()` const [inline]

Definition at line 53 of file CouenneMatrix.hpp.

7.38.3.5 `bool Couenne::CouenneScalar::operator<(const CouenneScalar & rhs)` const [inline]

Definition at line 55 of file CouenneMatrix.hpp.

7.38.3.6 `void Couenne::CouenneScalar::print()` const

7.38.4 Friends And Related Function Documentation

7.38.4.1 `bool operator<(const CouenneScalar & first, const CouenneScalar & second)` [friend]

Definition at line 62 of file CouenneMatrix.hpp.

7.38.5 Member Data Documentation

7.38.5.1 `int Couenne::CouenneScalar::index_` [protected]

index of element in vector

Definition at line 29 of file CouenneMatrix.hpp.

7.38.5.2 `expression* Couenne::CouenneScalar::elem_` [protected]

element

Definition at line 30 of file CouenneMatrix.hpp.

The documentation for this class was generated from the following file:

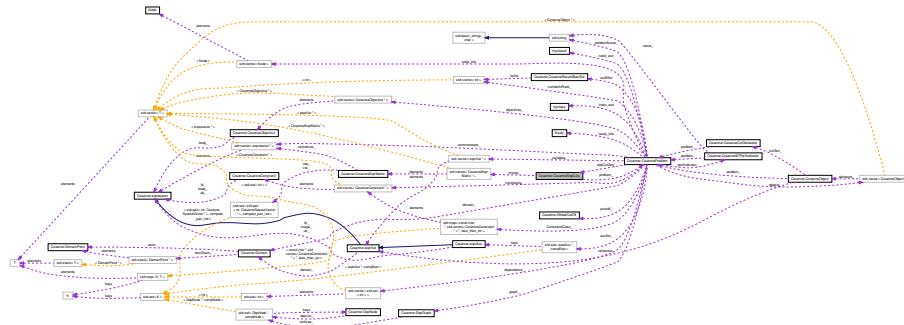
- /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneMatrix.hpp

7.39 Couenne::CouenneSdpCuts Class Reference

These are cuts of the form.

```
#include <CouenneSdpCuts.hpp>
```

Collaboration diagram for Couenne::CouenneSdpCuts:



Public Member Functions

- **CouenneSdpCuts (CouenneProblem *, JnlstPtr, const Ipopt::SmartPtr< Ipopt::OptionsList >)**
Constructor.
- **~CouenneSdpCuts ()**
Destructor.
- **CouenneSdpCuts & operator= (const CouenneSdpCuts &)**
Assignment.
- **CouenneSdpCuts (const CouenneSdpCuts &)**
Copy constructor.
- **virtual CglCutGenerator * clone () const**
Cloning constructor.
- **const bool doNotUse () const**
- **virtual void generateCuts (const OsiSolverInterface &, OsiCuts &, const CglTreeInfo=CglTreeInfo()) const**
The main CglCutGenerator.
- **void updateSol ()**

Static Public Member Functions

- **static void registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions)**
Add list of options to be read from file.

Protected Attributes

- **CouenneProblem * problem_**
pointer to problem info
- **bool doNotUse_**
after construction, true if there are enough product terms to justify application.
- **std::vector< CouenneExprMatrix * > minors_**

- int `numEigVec_`
minors on which to apply cuts
number of eigenvectors to be used (default: n)
- bool `onlyNegEV_`
only use negative eigenvalues (default: yes)
- bool `useSparsity_`
Sparsify eigenvalues before writing inequality (default: no)
- bool `fillMissingTerms_`
If minor not fully dense, create fictitious auxiliary variables that will be used in sdp cuts only (tighter than sdp cuts without)

7.39.1 Detailed Description

These are cuts of the form.

$$a' X a \geq 0$$

where X is a matrix constrained to be PSD.

Typical application is in problems with products forming a matrix of auxiliary variables $X_0 = (x_{ij})_{\{i,j \in N\}}$, and x_{ij} is the auxiliary variable for $x_i * x_j$. After reformulation, matrices like X_0 arise naturally and can be used to separate cuts that help strengthen the lower bound. See Sherali and Fraticelli for the base idea, and Qualizza, Belotti and Margot for an efficient rework and its implementation. Andrea Qualizza's code has been made open source and is used here (thanks Andrea!).

Definition at line 43 of file CouenneSdpCuts.hpp.

7.39.2 Constructor & Destructor Documentation

7.39.2.1 Couenne::CouenneSdpCuts::CouenneSdpCuts (CouenneProblem *, JnlstPtr , const Ipopt::SmartPtr< Ipopt::OptionsList >)

Constructor.

7.39.2.2 Couenne::CouenneSdpCuts::~CouenneSdpCuts ()

Destructor.

7.39.2.3 Couenne::CouenneSdpCuts::CouenneSdpCuts (const CouenneSdpCuts &)

Copy constructor.

7.39.3 Member Function Documentation

7.39.3.1 CouenneSdpCuts& Couenne::CouenneSdpCuts::operator= (const CouenneSdpCuts &)

Assignment.

7.39.3.2 virtual CglCutGenerator* Couenne::CouenneSdpCuts::clone () const [virtual]

Cloning constructor.

7.39.3.3 const bool Couenne::CouenneSdpCuts::doNotUse () const [inline]

Definition at line 76 of file CouenneSdpCuts.hpp.

7.39.3.4 virtual void Couenne::CouenneSdpCuts::generateCuts (const OsiSolverInterface & , OsiCuts & , const CglTreeInfo = CglTreeInfo ()) const [virtual]

The main CglCutGenerator.

7.39.3.5 static void Couenne::CouenneSdpCuts::registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > *options*) [static]

Add list of options to be read from file.

7.39.3.6 void Couenne::CouenneSdpCuts::updateSol ()

7.39.4 Member Data Documentation

7.39.4.1 CouenneProblem* Couenne::CouenneSdpCuts::problem_ [protected]

pointer to problem info

Definition at line 47 of file CouenneSdpCuts.hpp.

7.39.4.2 bool Couenne::CouenneSdpCuts::doNotUse_ [protected]

after construction, true if there are enough product terms to justify application.

If not, do not add this cut generator

Definition at line 49 of file CouenneSdpCuts.hpp.

7.39.4.3 std::vector<CouenneExprMatrix *> Couenne::CouenneSdpCuts::minors_ [protected]

minors on which to apply cuts

Definition at line 53 of file CouenneSdpCuts.hpp.

7.39.4.4 int Couenne::CouenneSdpCuts::numEigVec_ [protected]

number of eigenvectors to be used (default: n)

Definition at line 55 of file CouenneSdpCuts.hpp.

7.39.4.5 bool Couenne::CouenneSdpCuts::onlyNegEV_ [protected]

only use negative eigenvalues (default: yes)

Definition at line 57 of file CouenneSdpCuts.hpp.

7.39.4.6 bool Couenne::CouenneSdpCuts::useSparsity_ [protected]

Sparsify eigenvalues before writing inequality (default: no)

Definition at line 59 of file CouenneSdpCuts.hpp.

7.39.4.7 bool Couenne::CouenneSdpCuts::fillMissingTerms_ [protected]

If minor not fully dense, create fictitious auxiliary variables that will be used in sdp cuts only (tighter than sdp cuts without)

Definition at line 61 of file CouenneSdpCuts.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/[CouenneSdpCuts.hpp](#)

7.40 Couenne::CouenneSetup Class Reference

```
#include <BonCouenneSetup.hpp>
```

Public Member Functions

- [CouenneSetup \(\)](#)
Default constructor.
- [CouenneSetup \(const CouenneSetup &other\)](#)
Copy constructor.
- [virtual Bonmin::BabSetupBase * clone \(\) const](#)
virtual copy constructor.
- [virtual ~CouenneSetup \(\)](#)
destructor
- [bool InitializeCouenne \(char **argv=NULL, CouenneProblem *couenneProb=NULL, Ipopt::SmartPtr< Bonmin::TMINLP > tminlp=NULL, CouenneInterface *ci=NULL, Bonmin::Bab *bb=NULL\)](#)
Initialize from command line arguments.
- [virtual void registerOptions \(\)](#)
register the options
- [virtual void readOptionsFile \(\)](#)
Get the basic options if don't already have them.
- [CouenneCutGenerator * couennePtr \(\) const](#)
return pointer to cut generator (used to get pointer to problem)
- [bool displayStats \(\)](#)
true if one wants to display statistics at the end of program
- [void addMilpCutGenerators \(\)](#)
add cut generators
- [void setDoubleParameter \(const DoubleParameter &p, const double val\)](#)
modify parameter (used for MaxTime)
- [double getDoubleParameter \(const DoubleParameter &p\) const](#)
modify parameter (used for MaxTime)
- [void setNodeComparisonMethod \(Bonmin::BabSetupBase::NodeComparison c\)](#)

Static Public Member Functions

- [static void registerAllOptions \(Ipopt::SmartPtr< Bonmin::RegisteredOptions > roptions\)](#)
Register all Couenne options.

7.40.1 Detailed Description

Definition at line 43 of file BonCouenneSetup.hpp.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 Couenne::CouenneSetup::CouenneSetup() [inline]

Default constructor.

Definition at line 46 of file BonCouenneSetup.hpp.

7.40.2.2 Couenne::CouenneSetup::CouenneSetup (const CouenneSetup & other) [inline]

Copy constructor.

Definition at line 55 of file BonCouenneSetup.hpp.

7.40.2.3 virtual Couenne::CouenneSetup::~CouenneSetup () [virtual]

destructor

7.40.3 Member Function Documentation**7.40.3.1 virtual Bonmin::BabSetupBase* Couenne::CouenneSetup::clone () const [inline], [virtual]**

virtual copy constructor.

Definition at line 62 of file BonCouenneSetup.hpp.

7.40.3.2 bool Couenne::CouenneSetup::InitializeCouenne (char ** argv = NULL, CouenneProblem * couenneProb = NULL, Iopt::SmartPtr< Bonmin::TMINLP > tminlp = NULL, CouenneInterface * ci = NULL, Bonmin::Bab * bb = NULL)

Initialize from command line arguments.

7.40.3.3 virtual void Couenne::CouenneSetup::registerOptions () [virtual]

register the options

7.40.3.4 static void Couenne::CouenneSetup::registerAllOptions (Iopt::SmartPtr< Bonmin::RegisteredOptions > roptions) [static]

Register all [Couenne](#) options.

7.40.3.5 virtual void Couenne::CouenneSetup::readOptionsFile () [inline], [virtual]

Get the basic options if don't already have them.

Definition at line 81 of file BonCouenneSetup.hpp.

7.40.3.6 CouenneCutGenerator* Couenne::CouenneSetup::couennePtr () const [inline]

return pointer to cut generator (used to get pointer to problem)

Definition at line 87 of file BonCouenneSetup.hpp.

7.40.3.7 bool Couenne::CouenneSetup::displayStats () [inline]

true if one wants to display statistics at the end of program

Definition at line 91 of file BonCouenneSetup.hpp.

7.40.3.8 void Couenne::CouenneSetup::addMilpCutGenerators ()

add cut generators

7.40.3.9 void Couenne::CouenneSetup::setDoubleParameter (const DoubleParameter & p, const double val) [inline]

modify parameter (used for MaxTime)

Definition at line 98 of file BonCouenneSetup.hpp.

7.40.3.10 double Couenne::CouenneSetup::getDoubleParameter (const DoubleParameter & p) const [inline]

modify parameter (used for MaxTime)

Definition at line 102 of file BonCouenneSetup.hpp.

7.40.3.11 void Couenne::CouenneSetup::setNodeComparisonMethod (Bonmin::BabSetupBase::NodeComparison c) [inline]

Definition at line 105 of file BonCouenneSetup.hpp.

The documentation for this class was generated from the following file:

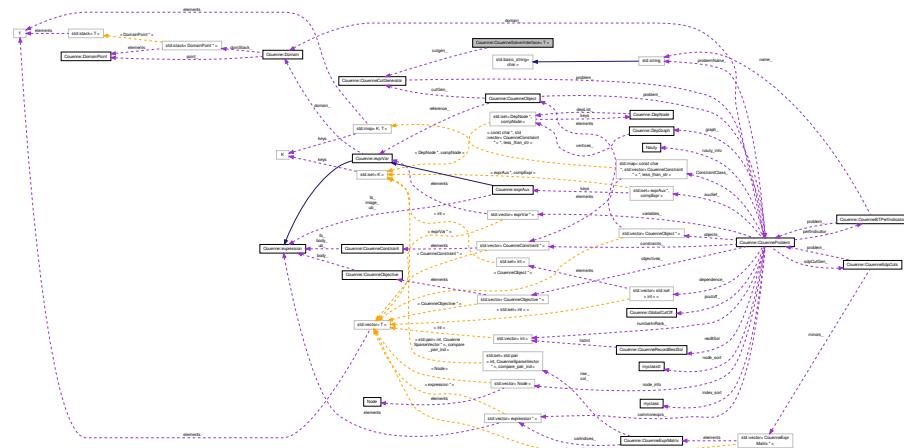
- /home/ted/COIN/trunk/Couenne/src/main/BonCouenneSetup.hpp

7.41 Couenne::CouenneSolverInterface< T > Class Template Reference

Solver interface class with a pointer to a [Couenne](#) cut generator.

```
#include <CouenneSolverInterface.hpp>
```

Collaboration diagram for Couenne::CouenneSolverInterface< T >:



Public Member Functions

- [CouenneSolverInterface \(CouenneCutGenerator *cg=NULL\)](#)
Constructor.
- [CouenneSolverInterface \(const CouenneSolverInterface &src\)](#)
Copy constructor.
- [~CouenneSolverInterface \(\)](#)
Destructor.
- virtual OsiSolverInterface * [clone](#) (bool copyData=true) const
Clone.
- virtual bool [isProvenPrimallyInfeasible](#) () const
we need to overwrite this since we might have internal knowledge
- virtual bool [isProvenOptimal](#) () const

- we need to overwrite this since we might have internal knowledge
- **CouenneCutGenerator * CutGen ()**
Return cut generator pointer.
- **void setCutGenPtr (CouenneCutGenerator *cg)**
Set cut generator pointer after setup, to avoid changes in the pointer due to cut generator cloning (it happens twice in the algorithm)
- **virtual void initialSolve ()**
Solve initial LP relaxation.
- **virtual void resolve ()**
Resolve an LP relaxation after problem modification.
- **virtual void resolve_nobt ()**
Resolve an LP without applying bound tightening beforehand.
- **virtual int tightenBounds (int lightweight)**
Tighten bounds on all variables (including continuous).
- **bool isProvenDualInfeasible () const**
set doingResolve_
- **virtual double getObjValue () const**
Get the objective function value.

Methods for strong branching.

- **virtual void markHotStart ()**
Create a hot start snapshot of the optimization process.
- **virtual void solveFromHotStart ()**
Optimize starting from the hot start snapshot.
- **virtual void unmarkHotStart ()**
Delete the hot start snapshot.

Protected Member Functions

- **virtual int tightenBoundsCLP (int lightweight)**
Copy of the Clp version — not light version.
- **virtual int tightenBoundsCLP_Light (int lightweight)**
Copy of the Clp version — light version.

Protected Attributes

- **CouenneCutGenerator * cutgen_**
The pointer to the Couenne cut generator.
- **bool knowInfeasible_**
Flag indicating that infeasibility was detected during solveFromHotStart.
- **bool knowOptimal_**
Flag indicating that optimality was detected during solveFromHotStart.
- **bool knowDualInfeasible_**
Flag indicating this problem's continuous relaxation is unbounded.

7.41.1 Detailed Description

```
template<class T>class Couenne::CouenneSolverInterface< T >
```

Solver interface class with a pointer to a [Couenne](#) cut generator.

Its main purposes are:

- 1) to apply bound tightening before re-solving
- 2) to replace OsiSolverInterface::isInteger () with problem_->[expression]->isInteger ()
- 3) to use NLP solution at branching

Definition at line 27 of file CouenneSolverInterface.hpp.

7.41.2 Constructor & Destructor Documentation

```
7.41.2.1 template<class T > Couenne::CouenneSolverInterface< T >::CouenneSolverInterface ( CouenneCutGenerator * cg = NULL )
```

Constructor.

```
7.41.2.2 template<class T > Couenne::CouenneSolverInterface< T >::CouenneSolverInterface ( const CouenneSolverInterface< T > & src )
```

Copy constructor.

```
7.41.2.3 template<class T > Couenne::CouenneSolverInterface< T >::~CouenneSolverInterface ( )
```

Destructor.

7.41.3 Member Function Documentation

```
7.41.3.1 template<class T > virtual OsiSolverInterface* Couenne::CouenneSolverInterface< T >::clone ( bool copyData = true ) const [inline], [virtual]
```

Clone.

Definition at line 41 of file CouenneSolverInterface.hpp.

```
7.41.3.2 template<class T > virtual bool Couenne::CouenneSolverInterface< T >::isProvenPrimalInfeasible ( ) const [virtual]
```

we need to overwrite this since we might have internal knowledge

```
7.41.3.3 template<class T > virtual bool Couenne::CouenneSolverInterface< T >::isProvenOptimal ( ) const [virtual]
```

we need to overwrite this since we might have internal knowledge

```
7.41.3.4 template<class T > CouenneCutGenerator* Couenne::CouenneSolverInterface< T >::CutGen ( ) [inline]
```

Return cut generator pointer.

Definition at line 51 of file CouenneSolverInterface.hpp.

7.41.3.5 template<class T> void Couenne::CouenneSolverInterface< T >::setCutGenPtr (CouenneCutGenerator * *cg*) [inline]

Set cut generator pointer after setup, to avoid changes in the pointer due to cut generator cloning (it happens twice in the algorithm)

Definition at line 57 of file CouenneSolverInterface.hpp.

7.41.3.6 template<class T> virtual void Couenne::CouenneSolverInterface< T >::initialSolve () [virtual]

Solve initial LP relaxation.

7.41.3.7 template<class T> virtual void Couenne::CouenneSolverInterface< T >::resolve () [virtual]

Resolve an LP relaxation after problem modification.

7.41.3.8 template<class T> virtual void Couenne::CouenneSolverInterface< T >::resolve_nobt () [inline], [virtual]

Resolve an LP without applying bound tightening beforehand.

Definition at line 70 of file CouenneSolverInterface.hpp.

7.41.3.9 template<class T> virtual void Couenne::CouenneSolverInterface< T >::markHotStart () [virtual]

Create a hot start snapshot of the optimization process.

7.41.3.10 template<class T> virtual void Couenne::CouenneSolverInterface< T >::solveFromHotStart () [virtual]

Optimize starting from the hot start snapshot.

7.41.3.11 template<class T> virtual void Couenne::CouenneSolverInterface< T >::unmarkHotStart () [virtual]

Delete the hot start snapshot.

7.41.3.12 template<class T> virtual int Couenne::CouenneSolverInterface< T >::tightenBounds (int *lightweight*) [virtual]

Tighten bounds on all variables (including continuous).

7.41.3.13 template<class T> bool Couenne::CouenneSolverInterface< T >::isProvenDualInfeasible () const

set doingResolve_

is this problem unbounded?

7.41.3.14 template<class T> virtual double Couenne::CouenneSolverInterface< T >::getObjValue () const [virtual]

Get the objective function value.

Modified due to possible constant objectives passed to [Couenne](#)

7.41.3.15 template<class T> virtual int Couenne::CouenneSolverInterface< T >::tightenBoundsCLP (int *lightweight*) [protected], [virtual]

Copy of the Clp version — not light version.

7.41.3.16 `template<class T> virtual int Couenne::CouenneSolverInterface< T >::tightenBoundsCLP_Light (int lightweight) [protected], [virtual]`

Copy of the Clp version — light version.

7.41.4 Member Data Documentation

7.41.4.1 `template<class T> CouenneCutGenerator* Couenne::CouenneSolverInterface< T >::cutgen_ [protected]`

The pointer to the [Couenne](#) cut generator.

Gives us a lot of information, for instance the nlp solver pointer, and the chance to do bound tightening before resolve ().

Definition at line 116 of file CouenneSolverInterface.hpp.

7.41.4.2 `template<class T> bool Couenne::CouenneSolverInterface< T >::knowInfeasible_ [protected]`

Flag indicating that infeasibility was detected during solveFromHotStart.

Definition at line 119 of file CouenneSolverInterface.hpp.

7.41.4.3 `template<class T> bool Couenne::CouenneSolverInterface< T >::knowOptimal_ [protected]`

Flag indicating that optimality was detected during solveFromHotStart.

Definition at line 122 of file CouenneSolverInterface.hpp.

7.41.4.4 `template<class T> bool Couenne::CouenneSolverInterface< T >::knowDualInfeasible_ [protected]`

Flag indicating this problem's continuous relaxation is unbounded.

Definition at line 125 of file CouenneSolverInterface.hpp.

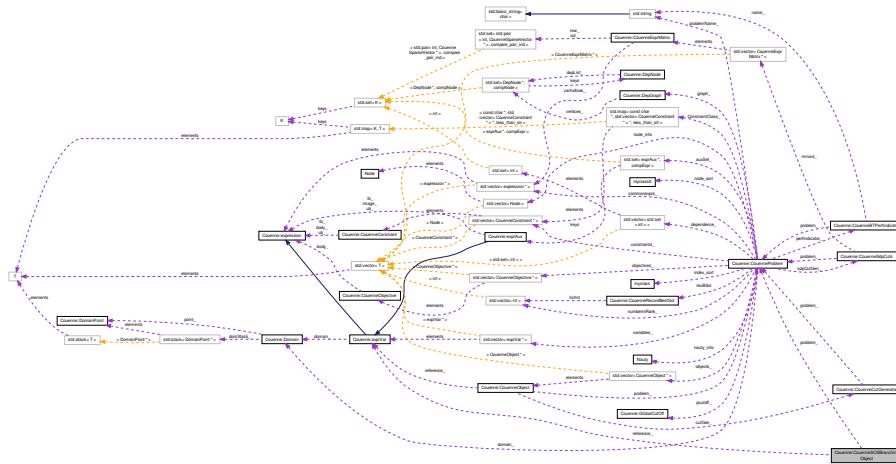
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/problem/[CouenneSolverInterface.hpp](#)

7.42 Couenne::CouenneSOSBranchingObject Class Reference

```
#include <CouenneSOSObject.hpp>
```

Collaboration diagram for Couenne::CouenneSOSBranchingObject:



Public Member Functions

- `CouenneSOSBranchingObject ()`
 - `CouenneSOSBranchingObject (CouenneProblem *p, exprVar *ref, OsiSolverInterface *solver, const OsiSOS *originalObject, int way, double separator, JnlstPtr jnlst, bool doFBBT, bool doConvCuts)`
 - `CouenneSOSBranchingObject (const CouenneSOSBranchingObject &src)`
 - virtual OsiBranchingObject * `clone () const`

Clone.
 - virtual double `branch (OsiSolverInterface *solver)`

Does next branch and updates state.

Protected Attributes

- **CouenneProblem * problem_**
pointer to Couenne problem
 - **exprVar * reference_**
The (auxiliary) variable this branching object refers to.
 - **JnlstPtr jnlst_**
SmartPointer to the Journalist.
 - **bool doFBBT_**
shall we do Feasibility based Bound Tightening (FBBT) at branching?
 - **bool doConvCuts_**
shall we add convexification cuts at branching?

7.42.1 Detailed Description

Definition at line 27 of file CouenneSOSObject.hpp.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 `Couenne::CouenneSOSBranchingObject::CouenneSOSBranchingObject() [inline]`

Definition at line 51 of file CouenneSOSObject.hpp.

7.42.2.2 `Couenne::CouenneSOSBranchingObject::CouenneSOSBranchingObject(CouenneProblem * p, exprVar * ref, OsiSolverInterface * solver, const OsiSOS * originalObject, int way, double separator, JnlstPtr jnlst, bool doFBBT, bool doConvCuts) [inline]`

Definition at line 54 of file CouenneSOSObject.hpp.

7.42.2.3 `Couenne::CouenneSOSBranchingObject::CouenneSOSBranchingObject(const CouenneSOSBranchingObject & src) [inline]`

Definition at line 73 of file CouenneSOSObject.hpp.

7.42.3 Member Function Documentation

7.42.3.1 `virtual OsiBranchingObject* Couenne::CouenneSOSBranchingObject::clone() const [inline], [virtual]`

Clone.

Definition at line 83 of file CouenneSOSObject.hpp.

7.42.3.2 `virtual double Couenne::CouenneSOSBranchingObject::branch(OsiSolverInterface * solver) [virtual]`

Does next branch and updates state.

7.42.4 Member Data Documentation

7.42.4.1 `CouenneProblem* Couenne::CouenneSOSBranchingObject::problem_ [protected]`

pointer to [Couenne](#) problem

Definition at line 32 of file CouenneSOSObject.hpp.

7.42.4.2 `exprVar* Couenne::CouenneSOSBranchingObject::reference_ [protected]`

The (auxiliary) variable this branching object refers to.

If the expression is $w=f(x,y)$, this is w , as opposed to [CouenneBranchingObject](#), where it would be either x or y .

Definition at line 37 of file CouenneSOSObject.hpp.

7.42.4.3 `JnlstPtr Couenne::CouenneSOSBranchingObject::jnlst_ [protected]`

SmartPointer to the Journalist.

Definition at line 40 of file CouenneSOSObject.hpp.

7.42.4.4 `bool Couenne::CouenneSOSBranchingObject::doFBBT_ [protected]`

shall we do Feasibility based Bound Tightening (FBBT) at branching?

Definition at line 43 of file CouenneSOSObject.hpp.

7.42.4.5 bool Couenne::CouenneSOSBranchingObject::doConvCuts_ [protected]

shall we add convexification cuts at branching?

Definition at line 46 of file CouenneSOSObject.hpp.

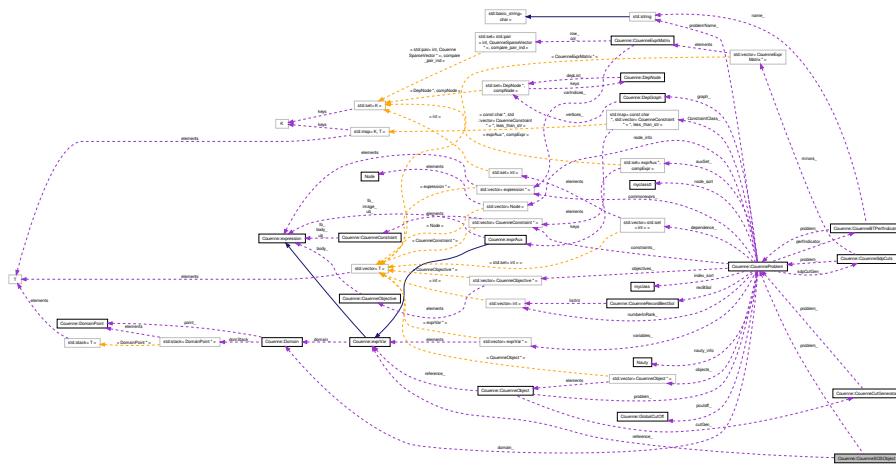
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/branch/CouenneSOSObject.hpp

7.43 Couenne::CouenneSOSObject Class Reference

```
#include <CouenneSOSObject.hpp>
```

Collaboration diagram for Couenne::CouenneSOSObject:



Public Member Functions

- [CouenneSOSObject](#) (OsiSolverInterface *solver, int nelem, int *indices, double *weights, int type, [CouenneProblem](#) *problem, [exprVar](#) *ref, [JnlstPtr](#) jnlst, bool doFBBT, bool doConvCuts)
- [CouenneSOSObject](#) (const [CouenneSOSObject](#) &src)

Copy constructor.

- virtual OsiObject * [clone](#) () const

Cloning method.

- OsiBranchingObject * [createBranch](#) (OsiSolverInterface *si, const OsiBranchingInformation *info, int way) const

create branching objects

Protected Attributes

- [CouenneProblem](#) * [problem_](#)
pointer to [Couenne](#) problem
- [exprVar](#) * [reference_](#)
The (auxiliary) variable this branching object refers to.
- [JnlstPtr](#) [jnlst_](#)

SmartPointer to the Journalist.

- bool `doFBBT_`
shall we do Feasibility based Bound Tightening (FBBT) at branching?
- bool `doConvCuts_`
shall we add convexification cuts at branching?

7.43.1 Detailed Description

Definition at line 95 of file CouenneSOSObject.hpp.

7.43.2 Constructor & Destructor Documentation

7.43.2.1 `Couenne::CouenneSOSObject::CouenneSOSObject (OsiSolverInterface * solver, int nelem, int * indices, double * weights, int type, CouenneProblem * problem, exprVar * ref, JnlstPtr jnlst, bool doFBBT, bool doConvCuts) [inline]`

Definition at line 118 of file CouenneSOSObject.hpp.

7.43.2.2 `Couenne::CouenneSOSObject::CouenneSOSObject (const CouenneSOSObject & src) [inline]`

Copy constructor.

Definition at line 134 of file CouenneSOSObject.hpp.

7.43.3 Member Function Documentation

7.43.3.1 `virtual OsiObject* Couenne::CouenneSOSObject::clone () const [inline], [virtual]`

Cloning method.

Definition at line 143 of file CouenneSOSObject.hpp.

7.43.3.2 `OsiBranchingObject* Couenne::CouenneSOSObject::createBranch (OsiSolverInterface * si, const OsiBranchingInformation * info, int way) const`

create branching objects

7.43.4 Member Data Documentation

7.43.4.1 `CouenneProblem* Couenne::CouenneSOSObject::problem_ [protected]`

pointer to `Couenne` problem

Definition at line 100 of file CouenneSOSObject.hpp.

7.43.4.2 `exprVar* Couenne::CouenneSOSObject::reference_ [protected]`

The (auxiliary) variable this branching object refers to.

If the expression is $w=f(x,y)$, this is w , as opposed to `CouenneBranchingObject`, where it would be either x or y .

Definition at line 105 of file CouenneSOSObject.hpp.

7.43.4.3 **JnlstPtr Couenne::CouenneSOSObject::jnlst_** [protected]

SmartPointer to the Journalist.

Definition at line 108 of file CouenneSOSObject.hpp.

7.43.4.4 **bool Couenne::CouenneSOSObject::doFBBT_** [protected]

shall we do Feasibility based Bound Tightening (FBBT) at branching?

Definition at line 111 of file CouenneSOSObject.hpp.

7.43.4.5 **bool Couenne::CouenneSOSObject::doConvCuts_** [protected]

shall we add convexification cuts at branching?

Definition at line 114 of file CouenneSOSObject.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/branch/CouenneSOSObject.hpp

7.44 Couenne::CouenneSparseBndVec< T > Class Template Reference

```
#include <CouenneSparseBndVec.hpp>
```

Public Member Functions

- **CouenneSparseBndVec** (unsigned int size)
Constructor.
- **CouenneSparseBndVec** (CouenneSparseBndVec &src)
Copy constructor.
- **~CouenneSparseBndVec** ()
Destructor.
- **void reset** ()
Reset (eeeeeasy!)
- **T & operator[]** (register unsigned int index)
Access – the only chance for garbage to be returned (and for valgrind to complain) is when object[ind] is READ without making sure it has been written.
- **T * data** ()
Return data in DENSE format – use with care.
- **unsigned int * indices** ()
Return indices in DENSE format – for use with data()
- **unsigned int nElements** ()
Return current size.
- **void resize** (unsigned int newsize)
Resize.

7.44.1 Detailed Description

```
template<class T>class Couenne::CouenneSparseBndVec< T >
```

Definition at line 16 of file CouenneSparseBndVec.hpp.

7.44.2 Constructor & Destructor Documentation

7.44.2.1 `template<class T > Couenne::CouenneSparseBndVec< T >::CouenneSparseBndVec(unsigned int size) [inline]`

Constructor.

Definition at line 51 of file CouenneSparseBndVec.hpp.

7.44.2.2 `template<class T > Couenne::CouenneSparseBndVec< T >::CouenneSparseBndVec(CouenneSparseBndVec< T > & src) [inline]`

Copy constructor.

`assert: src.sInd[ind] == i`

Definition at line 62 of file CouenneSparseBndVec.hpp.

7.44.2.3 `template<class T > Couenne::CouenneSparseBndVec< T >::~CouenneSparseBndVec() [inline]`

Destructor.

Definition at line 76 of file CouenneSparseBndVec.hpp.

7.44.3 Member Function Documentation

7.44.3.1 `template<class T > void Couenne::CouenneSparseBndVec< T >::reset() [inline]`

Reset (eeeeeasy!)

Definition at line 83 of file CouenneSparseBndVec.hpp.

7.44.3.2 `template<class T > T& Couenne::CouenneSparseBndVec< T >::operator[](register unsigned int index) [inline]`

Access – the only chance for garbage to be returned (and for valgrind to complain) is when object[ind] is READ without making sure it has been written.

This should not happen to the end user as read operations are only performed on the dense structure, after this object has been populated.

Definition at line 91 of file CouenneSparseBndVec.hpp.

7.44.3.3 `template<class T > T* Couenne::CouenneSparseBndVec< T >::data() [inline]`

Return data in DENSE format – use with care.

Definition at line 103 of file CouenneSparseBndVec.hpp.

7.44.3.4 `template<class T > unsigned int* Couenne::CouenneSparseBndVec< T >::indices() [inline]`

Return indices in DENSE format – for use with [data\(\)](#)

Definition at line 107 of file CouenneSparseBndVec.hpp.

7.44.3.5 `template<class T > unsigned int Couenne::CouenneSparseBndVec< T >::nElements() [inline]`

Return current size.

Definition at line 111 of file CouenneSparseBndVec.hpp.

7.44.3.6 template<class T> void Couenne::CouenneSparseBndVec< T >::resize(unsigned int *newsize*) [inline]

Resize.

Definition at line 115 of file CouenneSparseBndVec.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneSparseBndVec.hpp

7.45 Couenne::CouenneSparseMatrix Class Reference

Class for sparse Matrixs (used in modifying distances in FP)

```
#include <CouenneSparseMatrix.hpp>
```

Public Member Functions

- **CouenneSparseMatrix()**
Constructor.
- **CouenneSparseMatrix(const CouenneSparseMatrix &)**
Copy constructor.
- **CouenneSparseMatrix & operator=(const CouenneSparseMatrix &rhs)**
Assignment.
- **CouenneSparseMatrix * clone()**
Clone.
- **virtual ~CouenneSparseMatrix()**
Destructor.
- **int & num()**
Get methods.
- **double *& val()**
values
- **int *& col()**
column indices
- **int *& row()**
row indices

7.45.1 Detailed Description

Class for sparse Matrixs (used in modifying distances in FP)

Definition at line 17 of file CouenneSparseMatrix.hpp.

7.45.2 Constructor & Destructor Documentation

7.45.2.1 Couenne::CouenneSparseMatrix::CouenneSparseMatrix()

Constructor.

7.45.2.2 Couenne::CouenneSparseMatrix::CouenneSparseMatrix (const CouenneSparseMatrix &)

Copy constructor.

7.45.2.3 virtual Couenne::CouenneSparseMatrix::~CouenneSparseMatrix () [virtual]

Destructor.

7.45.3 Member Function Documentation

7.45.3.1 CouenneSparseMatrix& Couenne::CouenneSparseMatrix::operator= (const CouenneSparseMatrix & rhs)

Assignment.

7.45.3.2 CouenneSparseMatrix* Couenne::CouenneSparseMatrix::clone ()

Clone.

7.45.3.3 int& Couenne::CouenneSparseMatrix::num () [inline]

Get methods.

number of elements

Definition at line 37 of file CouenneSparseMatrix.hpp.

7.45.3.4 double*& Couenne::CouenneSparseMatrix::val () [inline]

values

Definition at line 38 of file CouenneSparseMatrix.hpp.

7.45.3.5 int*& Couenne::CouenneSparseMatrix::col () [inline]

column indices

Definition at line 39 of file CouenneSparseMatrix.hpp.

7.45.3.6 int*& Couenne::CouenneSparseMatrix::row () [inline]

row indices

Definition at line 40 of file CouenneSparseMatrix.hpp.

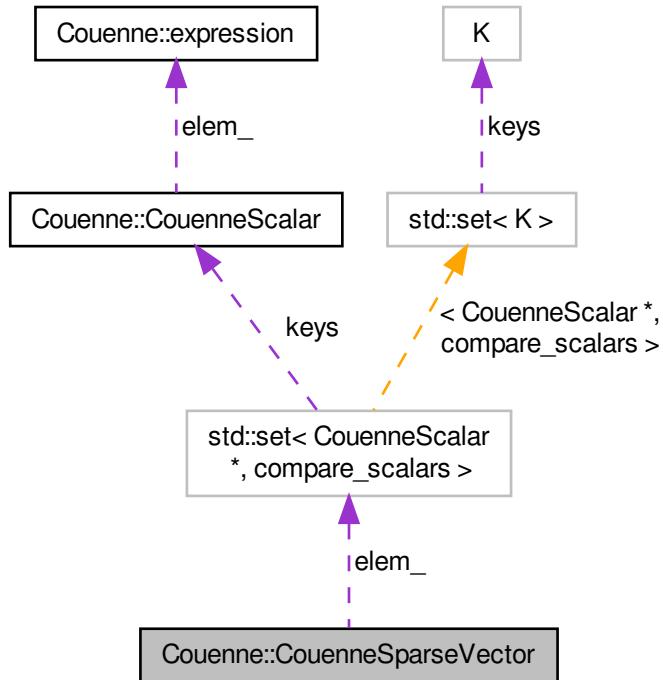
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/util/CouenneSparseMatrix.hpp

7.46 Couenne::CouenneSparseVector Class Reference

```
#include <CouenneMatrix.hpp>
```

Collaboration diagram for Couenne::CouenneSparseVector:



Classes

- struct `compare_scalars`

Public Member Functions

- `CouenneSparseVector ()`
- `~CouenneSparseVector ()`
- `CouenneSparseVector (const CouenneSparseVector &rhs)`
- `CouenneSparseVector & operator= (const CouenneSparseVector &rhs)`
- `CouenneSparseVector * clone ()`
- `void add_element (int index, expression *elem)`
- `void print () const`
- `const std::set< CouenneScalar *, compare_scalars > & getElements ()`
 - returns elements of vector as (ordered) set*
- `double operator* (const CouenneSparseVector &factor) const`
 - vector * vector (dot product)*
- `CouenneSparseVector & operator* (const CouenneExprMatrix &post) const`
 - vector * matrix*

- double `multiply_thres` (const CouenneSparseVector &*v2*, double *thres*) const
stops multiplication if above threshold

Protected Attributes

- std::set< CouenneScalar *, compare_scalars > *elem*

7.46.1 Detailed Description

Definition at line 66 of file CouenneMatrix.hpp.

7.46.2 Constructor & Destructor Documentation

7.46.2.1 Couenne::CouenneSparseVector::CouenneSparseVector() [inline]

Definition at line 82 of file CouenneMatrix.hpp.

7.46.2.2 Couenne::CouenneSparseVector::~CouenneSparseVector()

7.46.2.3 Couenne::CouenneSparseVector::CouenneSparseVector(const CouenneSparseVector & *rhs*)

7.46.3 Member Function Documentation

7.46.3.1 CouenneSparseVector& Couenne::CouenneSparseVector::operator=(const CouenneSparseVector & *rhs*)

7.46.3.2 CouenneSparseVector* Couenne::CouenneSparseVector::clone() [inline]

Definition at line 87 of file CouenneMatrix.hpp.

7.46.3.3 void Couenne::CouenneSparseVector::add_element(int *index*, expression * *elem*)

7.46.3.4 void Couenne::CouenneSparseVector::print() const

7.46.3.5 const std::set<CouenneScalar *, compare_scalars>& Couenne::CouenneSparseVector::getElements() [inline]

returns elements of vector as (ordered) set

Definition at line 93 of file CouenneMatrix.hpp.

7.46.3.6 double Couenne::CouenneSparseVector::operator*(const CouenneSparseVector & *factor*) const

vector * vector (dot product)

7.46.3.7 CouenneSparseVector& Couenne::CouenneSparseVector::operator*(const CouenneExprMatrix & *post*) const

vector * matrix

7.46.3.8 double Couenne::CouenneSparseVector::multiply_thres(const CouenneSparseVector & *v2*, double *thres*) const

stops multiplication if above threshold

7.46.4 Member Data Documentation

7.46.4.1 `std::set<CouenneScalar *, compare_scalars> Couenne::CouenneSparseVector::elem_` [protected]

Definition at line 78 of file CouenneMatrix.hpp.

The documentation for this class was generated from the following file:

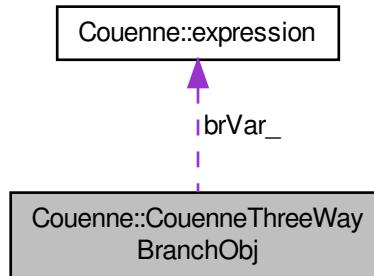
- /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneMatrix.hpp

7.47 Couenne::CouenneThreeWayBranchObj Class Reference

Spatial, three-way branching object.

```
#include <CouenneThreeWayBranchObj.hpp>
```

Collaboration diagram for Couenne::CouenneThreeWayBranchObj:



Public Member Functions

- `CouenneThreeWayBranchObj (JnlstPtr jnlst, expression *, CouNumber, CouNumber, int=THREE_CENTER)`
Constructor.
- `CouenneThreeWayBranchObj (const CouenneThreeWayBranchObj &src)`
Copy constructor.
- `virtual OsiBranchingObject * clone () const`
Cloning method.
- `virtual double branch (OsiSolverInterface *solver=NULL)`
Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.

Protected Attributes

- `expression * brVar_`
The variable this branching object refers to.
- `CouNumber lcrop_`

- *left divider*
- `CouNumber rcrop_`
- right divider*
- `int firstBranch_`
First branch to be performed: 0 is left, 1 is central, 2 is right.
- `JnlstPtr jnlst_`
True if the associated variable is integer.

7.47.1 Detailed Description

Spatial, three-way branching object.

Branching is performed on continuous variables but a better convexification is sought around the current point by dividing the interval in three parts

Definition at line 28 of file CouenneThreeWayBranchObj.hpp.

7.47.2 Constructor & Destructor Documentation

7.47.2.1 Couenne::CouenneThreeWayBranchObj::CouenneThreeWayBranchObj (`JnlstPtr jnlst, expression * , CouNumber , CouNumber , int = THREE_CENTER`)

Constructor.

7.47.2.2 Couenne::CouenneThreeWayBranchObj::CouenneThreeWayBranchObj (`const CouenneThreeWayBranchObj & src`) [inline]

Copy constructor.

Definition at line 42 of file CouenneThreeWayBranchObj.hpp.

7.47.3 Member Function Documentation

7.47.3.1 `virtual OsiBranchingObject* Couenne::CouenneThreeWayBranchObj::clone () const [inline], [virtual]`

Cloning method.

Definition at line 51 of file CouenneThreeWayBranchObj.hpp.

7.47.3.2 `virtual double Couenne::CouenneThreeWayBranchObj::branch (OsiSolverInterface * solver = NULL) [virtual]`

Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.

Returns change in guessed objective on next (what does "next" mean here?) branch

7.47.4 Member Data Documentation

7.47.4.1 `expression* Couenne::CouenneThreeWayBranchObj::brVar_ [protected]`

The variable this branching object refers to.

If the corresponding `CouenneObject` was created on $w=f(x,y)$, it is either x or y .

Definition at line 67 of file CouenneThreeWayBranchObj.hpp.

7.47.4.2 CouNumber Couenne::CouenneThreeWayBranchObj::lcrop_ [protected]

left divider

Definition at line 69 of file CouenneThreeWayBranchObj.hpp.

7.47.4.3 CouNumber Couenne::CouenneThreeWayBranchObj::rcrop_ [protected]

right divider

Definition at line 70 of file CouenneThreeWayBranchObj.hpp.

7.47.4.4 int Couenne::CouenneThreeWayBranchObj::firstBranch_ [protected]

First branch to be performed: 0 is left, 1 is central, 2 is right.

Definition at line 73 of file CouenneThreeWayBranchObj.hpp.

7.47.4.5 JnlstPtr Couenne::CouenneThreeWayBranchObj::jnlst_ [protected]

True if the associated variable is integer.

SmartPointer to the Journalist

Definition at line 79 of file CouenneThreeWayBranchObj.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/branch/CouenneThreeWayBranchObj.hpp

7.48 Couenne::CouenneTNLP Class Reference

Class for handling NLPs using [CouenneProblem](#).

```
#include <CouenneTNLP.hpp>
```

Public Member Functions

- [CouenneTNLP \(\)](#)
Empty constructor.
- [CouenneTNLP \(CouenneProblem *\)](#)
Constructor.
- [CouenneTNLP \(const CouenneTNLP &\)](#)
Copy constructor.
- [CouenneTNLP & operator= \(const CouenneTNLP &rhs\)](#)
Assignment.
- [CouenneTNLP * clone \(\)](#)
Clone.
- virtual [~CouenneTNLP \(\)](#)
Destructor.
- void [setInitSol \(const double *sol\)](#)
set initial solution
- [CouNumber * getSolution \(\)](#)
returns best solution (if it exists)
- [CouNumber getSolValue \(\)](#)

- virtual bool `get_nlp_info` (Ipopt::Index &n, Ipopt::Index &m, Ipopt::Index &nnz_jac_g, Ipopt::Index &nnz_h_lag, enum Ipopt::TNLP::IndexStyleEnum &index_style)

returns value of the best solution
- virtual bool `get_bounds_info` (Ipopt::Index n, Ipopt::Number *x_l, Ipopt::Number *x_u, Ipopt::Index m, Ipopt::Number *g_l, Ipopt::Number *g_u)

return the number of variables and constraints, and the number of non-zeros in the jacobian and the hessian.

return the information about the bound on the variables and constraints.
- virtual bool `get_variables_linearity` (Ipopt::Index n, Ipopt::TNLP::LinearityType *var_types)

return the variables linearity (TNLP::Linear or TNLP::NonLinear).
- virtual bool `get_constraints_linearity` (Ipopt::Index m, Ipopt::TNLP::LinearityType *const_types)

return the constraint linearity.
- virtual bool `get_starting_point` (Ipopt::Index n, bool init_x, Ipopt::Number *x, bool init_z, Ipopt::Number *z_L, Ipopt::Number *z_U, Ipopt::Index m, bool init_lambda, Ipopt::Number *lambda)

return the starting point.
- virtual bool `eval_f` (Ipopt::Index n, const Ipopt::Number *x, bool new_x, Ipopt::Number &obj_value)

return the value of the objective function
- virtual bool `eval_grad_f` (Ipopt::Index n, const Ipopt::Number *x, bool new_x, Ipopt::Number *grad_f)

return the vector of the gradient of the objective w.r.t. x
- virtual bool `eval_g` (Ipopt::Index n, const Ipopt::Number *x, bool new_x, Ipopt::Index m, Ipopt::Number *g)

return the vector of constraint values
- virtual bool `eval_jac_g` (Ipopt::Index n, const Ipopt::Number *x, bool new_x, Ipopt::Index m, Ipopt::Index nele_jac, Ipopt::Index *iRow, Ipopt::Index *jCol, Ipopt::Number *values)

return the jacobian of the constraints.
- virtual bool `eval_h` (Ipopt::Index n, const Ipopt::Number *x, bool new_x, Ipopt::Number obj_factor, Ipopt::Index m, const Ipopt::Number *lambda, bool new_lambda, Ipopt::Index nele_hess, Ipopt::Index *iRow, Ipopt::Index *jCol, Ipopt::Number *values)

return the hessian of the lagrangian.
- virtual void `finalize_solution` (Ipopt::SolverReturn status, Ipopt::Index n, const Ipopt::Number *x, const Ipopt::Number *z_L, const Ipopt::Number *z_U, Ipopt::Index m, const Ipopt::Number *g, const Ipopt::Number *lambda, Ipopt::Number obj_value, const Ipopt::IpoptData *ip_data, Ipopt::IpoptCalculatedQuantities *ip_cq)

This method is called when the algorithm is complete so the TNLP can store/write the solution.
- virtual bool `intermediate_callback` (Ipopt::AlgorithmMode mode, Ipopt::Index iter, Ipopt::Number obj_value, Ipopt::Number inf_pr, Ipopt::Number inf_du, Ipopt::Number mu, Ipopt::Number d_norm, Ipopt::Number regularization_size, Ipopt::Number alpha_du, Ipopt::Number alpha_pr, Ipopt::Index ls_trials, const Ipopt::IpoptData *ip_data, Ipopt::IpoptCalculatedQuantities *ip_cq)

Intermediate Callback method for the user.

Methods for quasi-Newton approximation. If the second

derivatives are approximated by `Ipopt`, it is better to do this only in the space of nonlinear variables.

The following methods are call by `Ipopt` if the quasi-Newton approximation is selected. If -1 is returned as number of nonlinear variables, `Ipopt` assumes that all variables are nonlinear. Otherwise, it calls `get_list_of_nonlinear_variables` with an array into which the indices of the nonlinear variables should be written - the array has the lengths `num_nonlin_vars`, which is identical with the return value of `get_number_of_nonlinear_variables ()`. It is assumed that the indices are counted starting with 1 in the FORTRAN_STYLE, and 0 for the C_STYLE.

- virtual Ipopt::Index `get_number_of_nonlinear_variables` ()
- virtual bool `get_list_of_nonlinear_variables` (Ipopt::Index num_nonlin_vars, Ipopt::Index *pos_nonlin_vars)

get real list

- virtual void [setObjective \(expression *newObj\)](#)
Change objective function and modify gradient expressions accordingly.
- [CouenneSparseMatrix *& optHessian \(\)](#)
Get methods.
- bool & [getSaveOptHessian \(\)](#)
set and get saveOptHessian_

7.48.1 Detailed Description

Class for handling NLPs using [CouenneProblem](#).

Definition at line 27 of file CouenneTNLP.hpp.

7.48.2 Constructor & Destructor Documentation

7.48.2.1 Couenne::CouenneTNLP::CouenneTNLP ()

Empty constructor.

7.48.2.2 Couenne::CouenneTNLP::CouenneTNLP (CouenneProblem *)

Constructor.

7.48.2.3 Couenne::CouenneTNLP::CouenneTNLP (const CouenneTNLP &)

Copy constructor.

7.48.2.4 virtual Couenne::CouenneTNLP::~CouenneTNLP () [virtual]

Destructor.

7.48.3 Member Function Documentation

7.48.3.1 CouenneTNLP& Couenne::CouenneTNLP::operator= (const CouenneTNLP & rhs)

Assignment.

7.48.3.2 CouenneTNLP* Couenne::CouenneTNLP::clone ()

Clone.

7.48.3.3 void Couenne::CouenneTNLP::setInitSol (const double * sol)

set initial solution

7.48.3.4 CouNumber* Couenne::CouenneTNLP::getSolution () [inline]

returns best solution (if it exists)

Definition at line 53 of file CouenneTNLP.hpp.

7.48.3.5 CouNumber Couenne::CouenneTNLP::getSolValue () [inline]

returns value of the best solution

Definition at line 57 of file CouenneTNLP.hpp.

7.48.3.6 virtual bool Couenne::CouenneTNLP::get_nlp_info (Ipopt::Index & n, Ipopt::Index & m, Ipopt::Index & nnz_jac_g, Ipopt::Index & nnz_h_lag, enum Ipopt::TNLP::IndexStyleEnum & index_style) [virtual]

return the number of variables and constraints, and the number of non-zeros in the jacobian and the hessian.

The index_style parameter lets you specify C or Fortran style indexing for the sparse matrix iRow and jCol parameters. C_STYLE is 0-based, and FORTRAN_STYLE is 1-based.

7.48.3.7 virtual bool Couenne::CouenneTNLP::get_bounds_info (Ipopt::Index n, Ipopt::Number * x_l, Ipopt::Number * x_u, Ipopt::Index m, Ipopt::Number * g_l, Ipopt::Number * g_u) [virtual]

return the information about the bound on the variables and constraints.

The value that indicates that a bound does not exist is specified in the parameters nlp_lower_bound_inf and nlp_upper_bound_inf. By default, nlp_lower_bound_inf is -1e19 and nlp_upper_bound_inf is 1e19. (see TNLPAdapter)

7.48.3.8 virtual bool Couenne::CouenneTNLP::get_variables_linearity (Ipopt::Index n, Ipopt::TNLP::LinearityType * var_types) [virtual]

return the variables linearity (TNLP::Linear or TNLP::NonLinear).

The var_types array should be allocated with length at least n. (default implementation just return false and does not fill the array).

7.48.3.9 virtual bool Couenne::CouenneTNLP::get_constraints_linearity (Ipopt::Index m, Ipopt::TNLP::LinearityType * const_types) [virtual]

return the constraint linearity.

array should be alocated with length at least n. (default implementation just return false and does not fill the array).

7.48.3.10 virtual bool Couenne::CouenneTNLP::get_starting_point (Ipopt::Index n, bool init_x, Ipopt::Number * x, bool init_z, Ipopt::Number * z_L, Ipopt::Number * z_U, Ipopt::Index m, bool init_lambda, Ipopt::Number * lambda) [virtual]

return the starting point.

The bool variables indicate whether the algorithm wants you to initialize x, z_L/z_U, and lambda, respectively. If, for some reason, the algorithm wants you to initialize these and you cannot, return false, which will cause [Ipopt](#) to stop. You will have to run [Ipopt](#) with different options then.

7.48.3.11 virtual bool Couenne::CouenneTNLP::eval_f (Ipopt::Index n, const Ipopt::Number * x, bool new_x, Ipopt::Number & obj_value) [virtual]

return the value of the objective function

7.48.3.12 virtual bool Couenne::CouenneTNLP::eval_grad_f (Ipopt::Index n, const Ipopt::Number * x, bool new_x, Ipopt::Number * grad_f) [virtual]

return the vector of the gradient of the objective w.r.t. x

7.48.3.13 virtual bool Couenne::CouenneTNLP::eval_g (Ipopt::Index n, const Ipopt::Number * x, bool new_x, Ipopt::Index m, Ipopt::Number * g) [virtual]

return the vector of constraint values

7.48.3.14 virtual bool Couenne::CouenneTNLP::eval_jac_g (Ipopt::Index *n*, const Ipopt::Number * *x*, bool *new_x*, Ipopt::Index *m*, Ipopt::Index *nele_jac*, Ipopt::Index * *iRow*, Ipopt::Index * *jCol*, Ipopt::Number * *values*) [virtual]

return the jacobian of the constraints.

The vectors *iRow* and *jCol* only need to be set once. The first call is used to set the structure only (*iRow* and *jCol* will be non-NULL, and *values* will be NULL) For subsequent calls, *iRow* and *jCol* will be NULL.

7.48.3.15 virtual bool Couenne::CouenneTNLP::eval_h (Ipopt::Index *n*, const Ipopt::Number * *x*, bool *new_x*, Ipopt::Number *obj_factor*, Ipopt::Index *m*, const Ipopt::Number * *lambda*, bool *new_lambda*, Ipopt::Index *nele_hess*, Ipopt::Index * *iRow*, Ipopt::Index * *jCol*, Ipopt::Number * *values*) [virtual]

return the hessian of the lagrangian.

The vectors *iRow* and *jCol* only need to be set once (during the first call). The first call is used to set the structure only (*iRow* and *jCol* will be non-NULL, and *values* will be NULL) For subsequent calls, *iRow* and *jCol* will be NULL. This matrix is symmetric - specify the lower diagonal only. A default implementation is provided, in case the user wants to se quasi-Newton approximations to estimate the second derivatives and doesn't not neet to implement this method.

7.48.3.16 virtual void Couenne::CouenneTNLP::finalize_solution (Ipopt::SolverReturn *status*, Ipopt::Index *n*, const Ipopt::Number * *x*, const Ipopt::Number * *z_L*, const Ipopt::Number * *z_U*, Ipopt::Index *m*, const Ipopt::Number * *g*, const Ipopt::Number * *lambda*, Ipopt::Number *obj_value*, const Ipopt::IpoptData * *ip_data*, Ipopt::IpoptCalculatedQuantities * *ip_cq*) [virtual]

This method is called when the algorithm is complete so the TNLP can store/write the solution.

7.48.3.17 virtual bool Couenne::CouenneTNLP::intermediate_callback (Ipopt::AlgorithmMode *mode*, Ipopt::Index *iter*, Ipopt::Number *obj_value*, Ipopt::Number *inf_pr*, Ipopt::Number *inf_du*, Ipopt::Number *mu*, Ipopt::Number *d_norm*, Ipopt::Number *regularization_size*, Ipopt::Number *alpha_du*, Ipopt::Number *alpha_pr*, Ipopt::Index *ls_trials*, const Ipopt::IpoptData * *ip_data*, Ipopt::IpoptCalculatedQuantities * *ip_cq*) [virtual]

Intermediate Callback method for the user.

Providing dummy default implementation. For details see IntermediateCallBack in IpNLP.hpp.

7.48.3.18 virtual Ipopt::Index Couenne::CouenneTNLP::get_number_of_nonlinear_variables () [virtual]

7.48.3.19 virtual bool Couenne::CouenneTNLP::get_list_of_nonlinear_variables (Ipopt::Index *num_nonlin_vars*, Ipopt::Index * *pos_nonlin_vars*) [virtual]

get real list

7.48.3.20 virtual void Couenne::CouenneTNLP::setObjective (expression * *newObj*) [virtual]

Change objective function and modify gradient expressions accordingly.

7.48.3.21 CouenneSparseMatrix*& Couenne::CouenneTNLP::optHessian () [inline]

Get methods.

Definition at line 182 of file CouenneTNLP.hpp.

7.48.3.22 bool& Couenne::CouenneTNLP::getSaveOptHessian () [inline]

set and get saveOptHessian_

Definition at line 186 of file CouenneTNLP.hpp.

The documentation for this class was generated from the following file:

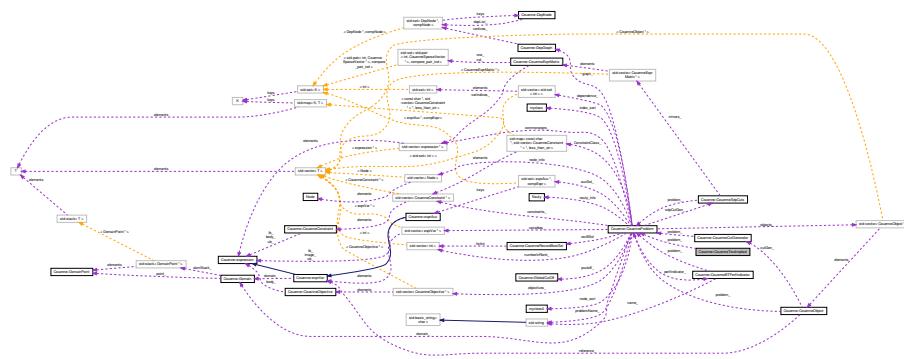
- `/home/ted/COIN/trunk/Couenne/src/interfaces/CouenneTNLP.hpp`

7.49 Couenne::CouenneTwoImplied Class Reference

Cut Generator for implied bounds derived from pairs of linear (in)equalities.

```
#include <CouenneTwoImplied.hpp>
```

Collaboration diagram for Couenne::CouenneTwoImplied:



Public Member Functions

- **CouenneTwoImplied** (*CouenneProblem* *, *JnlstPtr*, const *Ipopt::SmartPtr<Ipopt::OptionsList>*)
constructor
 - **CouenneTwoImplied** (const **CouenneTwoImplied** &)
copy constructor
 - **~CouenneTwoImplied** ()
destructor
 - **CouenneTwoImplied** * **clone** () const
clone method (necessary for the abstract CglCutGenerator class)
 - void **generateCuts** (const *OsiSolverInterface* &, *OsiCuts* &, const *CglTreeInfo*=*CglTreeInfo()*) const
the main CglCutGenerator

Static Public Member Functions

- static void **registerOptions** (Ioppt::SmartPtr< Bonmin::RegisteredOptions > options)
Add list of options to be read from file.

Protected Attributes

- `CouenneProblem * problem_`
pointer to problem data structure (used for post-BT)
 - `JnlstPtr jnlst_`
Journalist.
 - `int nMaxTrials_`
maximum number of trials in every call

- double `totalTime_`
Total CPU time spent separating cuts.
- double `totalInitTime_`
CPU time spent columning the row formulation.
- bool `firstCall_`
first call indicator
- int `depthLevelling_`
Depth of the BB tree where to start decreasing chance of running this.
- int `depthStopSeparate_`
Depth of the BB tree where stop separation.

7.49.1 Detailed Description

Cut Generator for implied bounds derived from pairs of linear (in)equalities.

Implied bounds usually work on a SINGLE inequality of the form

$$\ell_j \leq \sum_{i \in N_+} a_{ji}x_i + \sum_{i \in N_-} a_{ji}x_i \leq u_j$$

where $a_{ji} > 0$ for $i \in N_+$ and $a_{ji} < 0$ for $i \in N_-$, and allow one to infer better bounds $[x_i^L, x_i^U]$ on all variables with nonzero coefficients:

- (1) $x_i^L \geq (\ell_j - \sum_{i \in N_+} a_{ji}x_i^U - \sum_{i \in N_-} a_{ji}x_i^L)/a_{ji} \quad \forall i \in N_+$
- (2) $x_i^U \leq (u_j - \sum_{i \in N_+} a_{ji}x_i^L - \sum_{i \in N_-} a_{ji}x_i^U)/a_{ji} \quad \forall i \in N_+$
- (3) $x_i^L \geq (u_j - \sum_{i \in N_+} a_{ji}x_i^L - \sum_{i \in N_-} a_{ji}x_i^U)/a_{ji} \quad \forall i \in N_-$
- (4) $x_i^U \leq (\ell_j - \sum_{i \in N_+} a_{ji}x_i^U - \sum_{i \in N_-} a_{ji}x_i^L)/a_{ji} \quad \forall i \in N_+$

Consider now two inequalities:

$$\ell_h \leq \sum_{i \in N_+^1} a_{hi}x_i + \sum_{i \in N_-^1} a_{hi}x_i \leq u_h$$

$$\ell_k \leq \sum_{i \in N_+^2} a_{ki}x_i + \sum_{i \in N_-^2} a_{ki}x_i \leq u_k$$

and their CONVEX combination using α and $1 - \alpha$, where $\alpha \in [0, 1]$:

$$\ell' \leq \sum_{i \in N} b_i x_i \leq u'$$

with $N = N_+^1 \cup N_-^1 \cup N_+^2 \cup N_-^2$, $\ell' = \alpha\ell_h + (1 - \alpha)\ell_k$, and $u' = \alpha u_h + (1 - \alpha)u_k$. As an example where this might be useful, consider

$$x + y \geq 2$$

$$x - y \geq 1$$

with $x \in [0, 4]$ and $y \in [0, 1]$. (This is similar to an example given in Tawarmalani and Sahinidis to explain FBBT != OBBT, I believe.) The sum of the two above inequalities gives $x \geq 1.5$, while using only the implied bounds on the single inequalities gives $x \geq 1$.

The key consideration here is that the b_i coefficients, ℓ' , and u' are functions of α , which determines which, among (1)-(4), to apply. In general,

if $b_i > 0$ then

$$x_i^L \geq (l' - \sum_{j \in N'_+} b_j x_j^U - \sum_{j \in N'_-} b_j x_j^L)/b_i,$$

$$x_i^U \leq (u' - \sum_{j \in N'_+} b_j x_j^L - \sum_{j \in N'_-} b_j x_j^U)/b_i;$$

if $b_i < 0$ then

$$x_i^L \geq (l' - \sum_{j \in N'_+} b_j x_j^U - \sum_{j \in N'_-} b_j x_j^L)/b_i,$$

$$x_i^U \leq (u' - \sum_{j \in N'_+} b_j x_j^L - \sum_{j \in N'_-} b_j x_j^U) / b_i.$$

Each lower/upper bound is therefore a piecewise rational function of α , given that b_i and the content of N'_+ and N'_- depend on α . These functions are continuous (easy to prove) but not differentiable at some points of $[0, 1]$.

The purpose of this procedure is to find the maximum of the lower bounding function and the minimum of the upper bounding function.

Divide the interval $[0, 1]$ into at most $m + 1$ intervals (where m is the number of coefficients not identically zero, or the number of b_i that are nonzero for at least one value of α). The limits c_i of the subintervals are the zeros of each coefficient, i.e. the values of α such that $\alpha a_{ki} + (1 - \alpha)a_{hi} = 0$, or $c_i = \frac{-a_{hi}}{a_{ki}-a_{hi}}$.

Sorting these values gives us something to do on every interval $[c_j, c_{j+1}]$ when computing a new value of x_i^L and x_i^U , which I'll denote L_i and U_i in the following.

0) if $c_j = c_i$ then

- compute $VL = \lim_{\alpha \rightarrow c_j} L_i(\alpha)$
- if $= +\infty$, infeasible else compute derivative DL (should be $+\infty$)

1) else

- compute $VL = \lim_{\alpha \rightarrow c_j} L_i(\alpha)$ (can be retrieved from previous interval as $L_i(\alpha)$ is continuous)
- compute $DL = \lim_{\alpha \rightarrow c_j} dL_i(\alpha)$

update x^L with VL if necessary.

2) if $c_{j+1} = c_i$ then

- compute $VR = \lim_{\alpha \rightarrow c_{j+1}} L_i(\alpha)$
- if $= +\infty$, infeasible else compute derivative DR (should be $-\infty$)

3) else

- compute $VR = \lim_{\alpha \rightarrow c_{j+1}} L_i(\alpha)$
- compute $DR = \lim_{\alpha \rightarrow c_{j+1}} dL_i(\alpha)$

update x^L with VR if necessary.

if $DL > 0$ and $DR < 0$, there might be a maximum in between, otherwise continue to next interval

compute internal maximum VI , update x^L with VI if necessary.

Apply a similar procedure for the upper bound

This should be applied for any h, k, i , therefore we might have a lot to do. First, select possible pairs (h, k) among those for which there exists at least one variable that satisfies neither of the following conditions:

- a) same sign coefficient, constraints (h, k) both \geq or both \leq
- b) opposite sign coefficient, constraints (h, k) (\leq, \geq) or (\geq, \leq)

as in those cases, no c_i would be in $[0, 1]$

Definition at line 174 of file CouenneTwoImplied.hpp.

7.49.2 Constructor & Destructor Documentation

7.49.2.1 **Couenne::CouenneTwoImplied::CouenneTwoImplied (CouenneProblem *, JnlstPtr , const Ipopt::SmartPtr< Ipopt::OptionsList >)**

constructor

7.49.2.2 **Couenne::CouenneTwoImplied::CouenneTwoImplied (const CouenneTwoImplied &)**

copy constructor

7.49.2.3 **Couenne::CouenneTwoImplied::~CouenneTwoImplied ()**

destructor

7.49.3 Member Function Documentation

7.49.3.1 **CouenneTwoImplied* Couenne::CouenneTwoImplied::clone () const [inline]**

clone method (necessary for the abstract CglCutGenerator class)

Definition at line 190 of file CouenneTwoImplied.hpp.

7.49.3.2 **void Couenne::CouenneTwoImplied::generateCuts (const OsiSolverInterface & , OsiCuts & , const CglTreeInfo = CglTreeInfo ()) const**

the main CglCutGenerator

7.49.3.3 **static void Couenne::CouenneTwoImplied::registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions > options) [static]**

Add list of options to be read from file.

7.49.4 Member Data Documentation

7.49.4.1 **CouenneProblem* Couenne::CouenneTwoImplied::problem_ [protected]**

pointer to problem data structure (used for post-BT)

Definition at line 208 of file CouenneTwoImplied.hpp.

7.49.4.2 **JnlstPtr Couenne::CouenneTwoImplied::jnlst_ [protected]**

Journalist.

Definition at line 211 of file CouenneTwoImplied.hpp.

7.49.4.3 **int Couenne::CouenneTwoImplied::nMaxTrials_ [protected]**

maximum number of trials in every call

Definition at line 214 of file CouenneTwoImplied.hpp.

7.49.4.4 **double Couenne::CouenneTwoImplied::totalTime_ [mutable], [protected]**

Total CPU time spent separating cuts.

Definition at line 217 of file CouenneTwoImplied.hpp.

7.49.4.5 double Couenne::CouenneTwoImplied::totalInitTime_ [mutable], [protected]

CPU time spent columning the row formulation.

Definition at line 220 of file CouenneTwoImplied.hpp.

7.49.4.6 bool Couenne::CouenneTwoImplied::firstCall_ [mutable], [protected]

first call indicator

Definition at line 223 of file CouenneTwoImplied.hpp.

7.49.4.7 int Couenne::CouenneTwoImplied::depthLevelling_ [protected]

Depth of the BB tree where to start decreasing chance of running this.

Definition at line 226 of file CouenneTwoImplied.hpp.

7.49.4.8 int Couenne::CouenneTwoImplied::depthStopSeparate_ [protected]

Depth of the BB tree where stop separation.

Definition at line 229 of file CouenneTwoImplied.hpp.

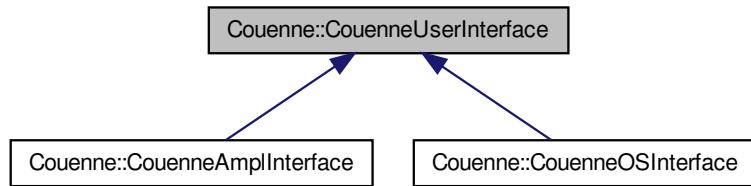
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/bound_tightening/twoImpliedBT/CouenneTwoImplied.hpp

7.50 Couenne::CouenneUserInterface Class Reference

```
#include <CouenneUserInterface.hpp>
```

Inheritance diagram for Couenne::CouenneUserInterface:



Public Member Functions

- [CouenneUserInterface](#) (Ipopt::SmartPtr< Ipopt::OptionsList > options_, Ipopt::SmartPtr< Ipopt::Journalist > jnlst_)
- virtual [~CouenneUserInterface](#) ()
- virtual bool [setupJournals](#) ()

Setup journals for printing.

- virtual `CouenneProblem * getCouenneProblem ()=0`
Should return the problem to solve in algebraic form.
- virtual `Iopt::SmartPtr< Bonmin::TMINLP > getTMINLP ()=0`
Should return the problem to solve as TMINLP.
- virtual bool `addBabPlugins (Bonmin::Bab &bab)`
Called after B&B object is setup.
- virtual bool `writeSolution (Bonmin::Bab &bab)`
Called after B&B finished.

Protected Attributes

- `Iopt::SmartPtr< Iopt::OptionsList > options`
- `Iopt::SmartPtr< Iopt::Journalist > jnlst`

7.50.1 Detailed Description

Definition at line 32 of file CouenneUserInterface.hpp.

7.50.2 Constructor & Destructor Documentation

7.50.2.1 `Couenne::CouenneUserInterface::CouenneUserInterface (Iopt::SmartPtr< Iopt::OptionsList > options_, Iopt::SmartPtr< Iopt::Journalist > jnlst_) [inline]`

Definition at line 38 of file CouenneUserInterface.hpp.

7.50.2.2 `virtual Couenne::CouenneUserInterface::~CouenneUserInterface () [inline], [virtual]`

Definition at line 42 of file CouenneUserInterface.hpp.

7.50.3 Member Function Documentation

7.50.3.1 `virtual bool Couenne::CouenneUserInterface::setupJournals () [inline], [virtual]`

Setup journals for printing.

Default is to have one journal that prints to stdout.

Definition at line 47 of file CouenneUserInterface.hpp.

7.50.3.2 `virtual CouenneProblem* Couenne::CouenneUserInterface::getCouenneProblem () [pure virtual]`

Should return the problem to solve in algebraic form.

NOTE: `Couenne` is (currently) going to modify this problem!

Implemented in `Couenne::CouenneOSInterface`, and `Couenne::CouenneAmplInterface`.

7.50.3.3 virtual `Iopt::SmartPtr<Bonmin::TMINLP>` `Couenne::CouenneUserInterface::getTMINLP()` [pure virtual]

Should return the problem to solve as TMINLP.

Implemented in `Couenne::CouenneOSInterface`, and `Couenne::CouenneAmplInterface`.

7.50.3.4 virtual bool `Couenne::CouenneUserInterface::addBabPlugins(Bonmin::Bab & bab)` [inline], [virtual]

Called after B&B object is setup.

User should add plugins like cut generators, bound tighteners, or heuristics here.

Definition at line 65 of file `CouenneUserInterface.hpp`.

7.50.3.5 virtual bool `Couenne::CouenneUserInterface::writeSolution(Bonmin::Bab & bab)` [inline], [virtual]

Called after B&B finished.

Should write solution information.

Reimplemented in `Couenne::CouenneOSInterface`, and `Couenne::CouenneAmplInterface`.

Definition at line 79 of file `CouenneUserInterface.hpp`.

7.50.4 Member Data Documentation

7.50.4.1 `Iopt::SmartPtr<Iopt::OptionsList>` `Couenne::CouenneUserInterface::options` [protected]

Definition at line 34 of file `CouenneUserInterface.hpp`.

7.50.4.2 `Iopt::SmartPtr<Iopt::Journalist>` `Couenne::CouenneUserInterface::jnlst` [protected]

Definition at line 35 of file `CouenneUserInterface.hpp`.

The documentation for this class was generated from the following file:

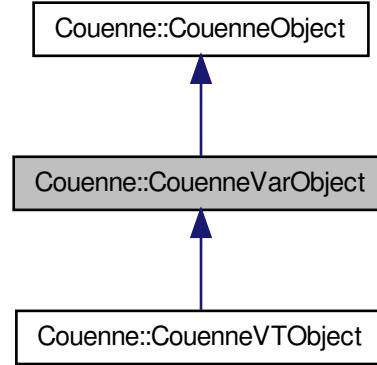
- /home/ted/COIN/trunk/Couenne/src/interfaces/[CouenneUserInterface.hpp](#)

7.51 Couenne::CouenneVarObject Class Reference

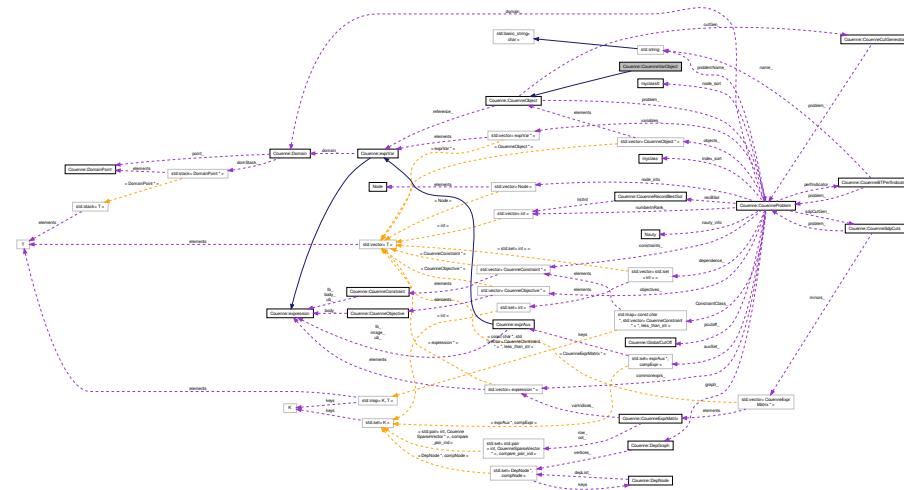
OsiObject for variables in a MINLP.

```
#include <CouenneVarObject.hpp>
```

Inheritance diagram for Couenne::CouenneVarObject:



Collaboration diagram for Couenne::CouenneVarObject:



Public Member Functions

- **CouenneVarObject** (*CouenneCutGenerator* *c, *CouenneProblem* *p, *exprVar* *ref, *Bonmin::BabSetupBase* *base, *JnlstPtr* jnlst, int varSelection)

Constructor with information for branching point selection strategy.
 - **CouenneVarObject** (const *CouenneVarObject* &src)

Copy constructor.
 - **~CouenneVarObject** ()

Destructor.
 - virtual *CouenneObject* * **clone** () const

Cloning method.

- virtual double [infeasibility](#) (const OsiBranchingInformation *info, int &way) const
compute infeasibility of this variable x as the sum/min/max of all infeasibilities of auxiliaries w whose defining function depends on x $|w - f(x)|$
- virtual double [checkInfeasibility](#) (const OsiBranchingInformation *info) const
compute infeasibility of this variable, $|w - f(x)|$, where w is the auxiliary variable defined as $w = f(x)$
- virtual OsiBranchingObject * [createBranch](#) (OsiSolverInterface *, const OsiBranchingInformation *, int) const
create CouenneBranchingObject or CouenneThreeWayBranchObj based on this object
- virtual double [feasibleRegion](#) (OsiSolverInterface *, const OsiBranchingInformation *) const
fix nonlinear coordinates of current integer-nonlinear feasible solution
- virtual bool [isCuttable](#) () const
are we on the bad or good side of the expression?

Protected Member Functions

- [CouNumber computeBranchingPoint](#) (const OsiBranchingInformation *info, int &bestWay, const [CouenneObject](#) *&criticalObject) const
Method computing the branching point.

Protected Attributes

- int [varSelection_](#)
branching scheme used.

Additional Inherited Members

7.51.1 Detailed Description

OsiObject for variables in a MINLP.

Definition at line 22 of file CouenneVarObject.hpp.

7.51.2 Constructor & Destructor Documentation

7.51.2.1 Couenne::CouenneVarObject::CouenneVarObject (CouenneCutGenerator * c, CouenneProblem * p, exprVar * ref, Bonmin::BabSetupBase * base, JnlstPtr jnlst, int varSelection)

Constructor with information for branching point selection strategy.

7.51.2.2 Couenne::CouenneVarObject::CouenneVarObject (const CouenneVarObject & src) [inline]

Copy constructor.

Definition at line 35 of file CouenneVarObject.hpp.

7.51.2.3 Couenne::CouenneVarObject::~CouenneVarObject () [inline]

Destructor.

Definition at line 40 of file CouenneVarObject.hpp.

7.51.3 Member Function Documentation

7.51.3.1 `virtual CouenneObject* Couenne::CouenneVarObject::clone() const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::CouenneObject](#).

Reimplemented in [Couenne::CouenneVTOObject](#).

Definition at line 43 of file [CouenneVarObject.hpp](#).

7.51.3.2 `virtual double Couenne::CouenneVarObject::infeasibility(const OsiBranchingInformation * info, int & way) const [virtual]`

compute infeasibility of this variable x as the sum/min/max of all infeasibilities of auxiliaries w whose defining function depends on $x |w - f(x)|$

TODO: suggest way

Reimplemented from [Couenne::CouenneObject](#).

Reimplemented in [Couenne::CouenneVTOObject](#).

7.51.3.3 `virtual double Couenne::CouenneVarObject::checkInfeasibility(const OsiBranchingInformation * info) const [virtual]`

compute infeasibility of this variable, $|w - f(x)|$, where w is the auxiliary variable defined as $w = f(x)$

Reimplemented from [Couenne::CouenneObject](#).

7.51.3.4 `virtual OsiBranchingObject* Couenne::CouenneVarObject::createBranch(OsiSolverInterface *, const OsiBranchingInformation *, int) const [virtual]`

create [CouenneBranchingObject](#) or [CouenneThreeWayBranchObj](#) based on this object

Reimplemented from [Couenne::CouenneObject](#).

7.51.3.5 `virtual double Couenne::CouenneVarObject::feasibleRegion(OsiSolverInterface *, const OsiBranchingInformation *) const [virtual]`

fix nonlinear coordinates of current integer-nonlinear feasible solution

Reimplemented from [Couenne::CouenneObject](#).

7.51.3.6 `virtual bool Couenne::CouenneVarObject::isCuttable() const [virtual]`

are we on the bad or good side of the expression?

Reimplemented from [Couenne::CouenneObject](#).

7.51.3.7 `CouNumber Couenne::CouenneVarObject::computeBranchingPoint(const OsiBranchingInformation * info, int & bestWay, const CouenneObject *& criticalObject) const [protected]`

Method computing the branching point.

7.51.4 Member Data Documentation

7.51.4.1 int Couenne::CouenneVarObject::varSelection_ [protected]

branching scheme used.

Experimental: still figuring out why plain LP branching doesn't work with strong/reliability branching

Definition at line 73 of file CouenneVarObject.hpp.

The documentation for this class was generated from the following file:

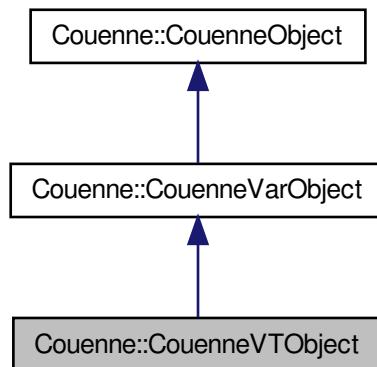
- /home/ted/COIN/trunk/Couenne/src/branch/CouenneVarObject.hpp

7.52 Couenne::CouenneVTOObject Class Reference

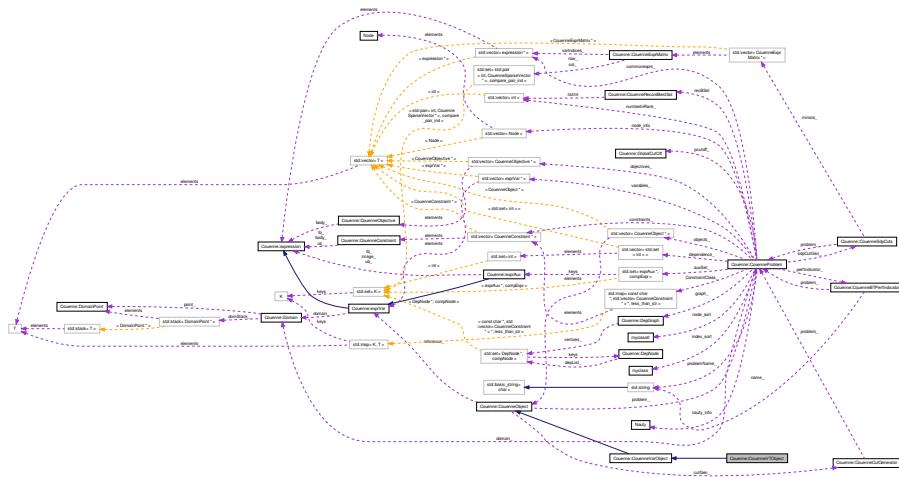
OsiObject for violation transfer on variables in a MINLP.

```
#include <CouenneVTOObject.hpp>
```

Inheritance diagram for Couenne::CouenneVTOObject:



Collaboration diagram for Couenne::CouenneVTOObject:



Public Member Functions

- [CouenneVTOObject \(CouenneCutGenerator *c, CouenneProblem *p, exprVar *ref, Bonmin::BabSetupBase *base, JnlstPtr jnlst, int varSelection\)](#)
Constructor with information for branching point selection strategy.
- [CouenneVTOObject \(const CouenneVTOObject &src\)](#)
Copy constructor.
- [~CouenneVTOObject \(\)](#)
Destructor.
- virtual [CouenneObject * clone \(\) const](#)
Cloning method.
- virtual double [infeasibility \(const OsiBranchingInformation *info, int &way\) const](#)
compute infeasibility of this variable x as the sum/min/max of all infeasibilities of auxiliaries w whose defining function depends on x |w - f(x)|

Additional Inherited Members

7.52.1 Detailed Description

OsiObject for violation transfer on variables in a MINLP.

Definition at line 19 of file CouenneVTOObject.hpp.

7.52.2 Constructor & Destructor Documentation

7.52.2.1 Couenne::CouenneVTOObject::CouenneVTOObject (CouenneCutGenerator * c, CouenneProblem * p, exprVar * ref, Bonmin::BabSetupBase * base, JnlstPtr jnlst, int varSelection) [inline]

Constructor with information for branching point selection strategy.

Definition at line 24 of file CouenneVTOObject.hpp.

7.52.2.2 Couenne::CouenneVTOObject::CouenneVTOObject (const CouenneVTOObject & src) [inline]

Copy constructor.

Definition at line 35 of file CouenneVTOObject.hpp.

7.52.2.3 Couenne::CouenneVTOObject::~CouenneVTOObject () [inline]

Destructor.

Definition at line 39 of file CouenneVTOObject.hpp.

7.52.3 Member Function Documentation

7.52.3.1 virtual CouenneObject* Couenne::CouenneVTOObject::clone () const [inline], [virtual]

Cloning method.

Reimplemented from [Couenne::CouenneVarObject](#).

Definition at line 42 of file CouenneVTOObject.hpp.

7.52.3.2 virtual double Couenne::CouenneVTOObject::infeasibility (const OsiBranchingInformation * info, int & way) const [virtual]

compute infeasibility of this variable x as the sum/min/max of all infeasibilities of auxiliaries w whose defining function depends on $|w - f(x)|$

Reimplemented from [Couenne::CouenneVarObject](#).

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/branch/[CouenneVTOObject.hpp](#)

7.53 Couenne::CouExpr Class Reference

```
#include <CouExpr.hpp>
```

Public Member Functions

- [CouExpr \(expression *e\)](#)
- [CouExpr \(const CouExpr &e\)](#)
- [CouExpr & operator= \(CouExpr &e\)](#)
- [expression * Expression \(\) const](#)

7.53.1 Detailed Description

Definition at line 17 of file CouExpr.hpp.

7.53.2 Constructor & Destructor Documentation

7.53.2.1 Couenne::CouExpr::CouExpr (expression * e) [inline]

Definition at line 25 of file CouExpr.hpp.

7.53.2.2 Couenne::CouExpr::CouExpr (const CouExpr & e) [inline]

Definition at line 28 of file CouExpr.hpp.

7.53.3 Member Function Documentation

7.53.3.1 CouExpr& Couenne::CouExpr::operator= (CouExpr & e) [inline]

Definition at line 32 of file CouExpr.hpp.

7.53.3.2 expression* Couenne::CouExpr::Expression () const [inline]

Definition at line 37 of file CouExpr.hpp.

The documentation for this class was generated from the following file:

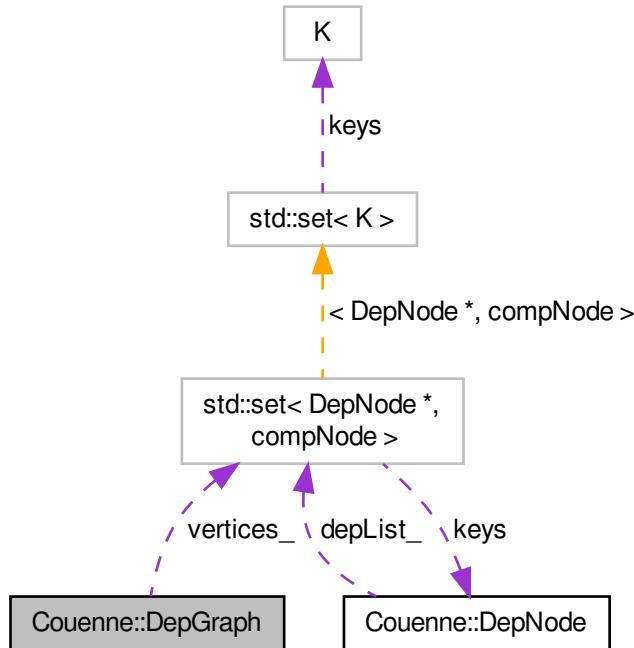
- /home/ted/COIN/trunk/Couenne/src/expression/CouExpr.hpp

7.54 Couenne::DepGraph Class Reference

Dependence graph.

```
#include <CouenneDepGraph.hpp>
```

Collaboration diagram for Couenne::DepGraph:



Public Member Functions

- **DepGraph ()**
constructor
- **~DepGraph ()**
destructor
- **std::set< DepNode *, compNode > & Vertices ()**
return vertex set
- **int & Counter ()**
node index counter
- **void insert (exprVar *)**
insert new variable if new
- **void insert (exprAux *)**
insert new auxiliary if new
- **void erase (exprVar *)**
delete element
- **bool depends (int, int, bool=false)**
does w depend on x?
- **void createOrder ()**
assign numbering to all nodes of graph
- **void print (bool descend=false)**
debugging procedure
- **DepNode * lookup (int index)**
search for node in vertex set
- **bool checkCycles ()**
check for dependence cycles in graph
- **void replaceIndex (int oldVar, int newVar)**
replace, throughout the whole graph, the index of a variable with another in the entire graph.

Protected Attributes

- **std::set< DepNode *, compNode > vertices_**
set of variable nodes
- **int counter_**
counter to assign numbering to all nodes

7.54.1 Detailed Description

Dependence graph.

Shows dependence of auxiliary variable on other (auxiliary and/or original) variables

Definition at line 115 of file CouenneDepGraph.hpp.

7.54.2 Constructor & Destructor Documentation

7.54.2.1 Couenne::DepGraph::DepGraph() [inline]

constructor

Definition at line 128 of file CouenneDepGraph.hpp.

7.54.2.2 Couenne::DepGraph::~DepGraph() [inline]

destructor

Definition at line 131 of file CouenneDepGraph.hpp.

7.54.3 Member Function Documentation**7.54.3.1 std::set<DepNode *, compNode>& Couenne::DepGraph::Vertices() [inline]**

return vertex set

Definition at line 138 of file CouenneDepGraph.hpp.

7.54.3.2 int& Couenne::DepGraph::Counter() [inline]

node index counter

Definition at line 142 of file CouenneDepGraph.hpp.

7.54.3.3 void Couenne::DepGraph::insert(exprVar *)

insert new variable if new

7.54.3.4 void Couenne::DepGraph::insert(exprAux *)

insert new auxiliary if new

7.54.3.5 void Couenne::DepGraph::erase(exprVar *)

delete element

7.54.3.6 bool Couenne::DepGraph::depends(int , int , bool =false)

does w depend on x?

7.54.3.7 void Couenne::DepGraph::createOrder()

assign numbering to all nodes of graph

7.54.3.8 void Couenne::DepGraph::print(bool descend=false)

debugging procedure

7.54.3.9 DepNode* Couenne::DepGraph::lookup(int index)

search for node in vertex set

7.54.3.10 bool Couenne::DepGraph::checkCycles()

check for dependence cycles in graph

7.54.3.11 void Couenne::DepGraph::replaceIndex(int oldVar, int newVar)

replace, throughout the whole graph, the index of a variable with another in the entire graph.

Used when redundant constraints w := x are discovered

7.54.4 Member Data Documentation

7.54.4.1 `std::set<DepNode *, compNode> Couenne::DepGraph::vertices_` [protected]

set of variable nodes

Definition at line 120 of file CouenneDepGraph.hpp.

7.54.4.2 `int Couenne::DepGraph::counter_` [protected]

counter to assign numbering to all nodes

Definition at line 123 of file CouenneDepGraph.hpp.

The documentation for this class was generated from the following file:

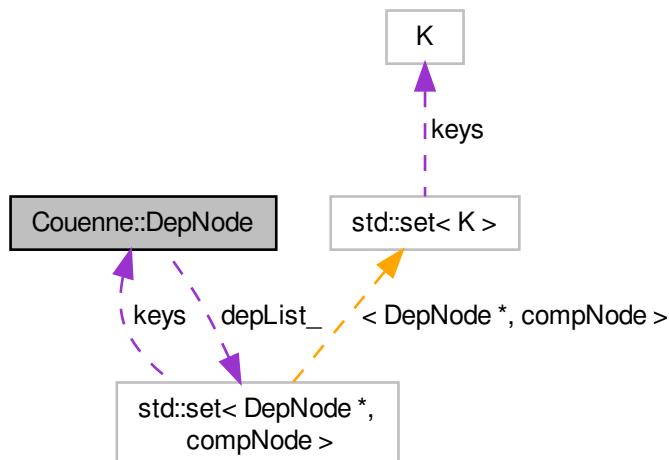
- /home/ted/COIN/trunk/Couenne/src/problem/depGraph/CouenneDepGraph.hpp

7.55 Couenne::DepNode Class Reference

vertex of a dependence graph.

```
#include <CouenneDepGraph.hpp>
```

Collaboration diagram for Couenne::DepNode:



Public Types

- enum `dep_color` { `DEP_WHITE`, `DEP_GRAY`, `DEP_BLACK` }
- color used in DFS for checking cycles*

Public Member Functions

- `DepNode (int ind)`
fictitious constructor: only fill in index (such object is used in find() and then discarded)
- `~DepNode ()`
destructor
- `int Index () const`
return index of this variable
- `int Order () const`
return index of this variable
- `std::set< DepNode *, compNode > * DepList () const`
return all variables it depends on
- `bool depends (int xi, bool=false, std::set< DepNode *, compNode > *already_visited=NULL) const`
does this variable depend on variable with index xi?
- `void createOrder (DepGraph *)`
assign numbering to all nodes of graph
- `void print (int=0, bool descend=false) const`
debugging procedure
- `enum dep_color & color ()`
return or set color of a node
- `std::set< DepNode *, compNode > * depList ()`
index nodes on which this one depends (forward star in dependence graph)
- `void replaceIndex (DepNode *oldVarNode, DepNode *newVarNode)`
replace the index of a variable with another in the entire graph.

Protected Attributes

- `int index_`
index of variable associated with node
- `std::set< DepNode *, compNode > * depList_`
index nodes on which this one depends (forward star in dependence graph)
- `int order_`
order in which this variable should be updated, evaluated, etc.
- `enum dep_color color_`
color used in DFS for checking cycles

7.55.1 Detailed Description

vertex of a dependence graph.

Contains variable and its forward star (all variables it depends on)

Definition at line 33 of file CouenneDepGraph.hpp.

7.55.2 Member Enumeration Documentation

7.55.2.1 enum Couenne::DepNode::dep_color

color used in DFS for checking cycles

Enumerator:

DEP_WHITE

DEP_GRAY

DEP_BLACK

Definition at line 38 of file CouenneDepGraph.hpp.

7.55.3 Constructor & Destructor Documentation

7.55.3.1 Couenne::DepNode::DepNode (int *ind*) [inline]

fictitious constructor: only fill in index (such object is used in find() and then discarded)

Definition at line 59 of file CouenneDepGraph.hpp.

7.55.3.2 Couenne::DepNode::~DepNode () [inline]

destructor

Definition at line 66 of file CouenneDepGraph.hpp.

7.55.4 Member Function Documentation

7.55.4.1 int Couenne::DepNode::Index () const [inline]

return index of this variable

Definition at line 70 of file CouenneDepGraph.hpp.

7.55.4.2 int Couenne::DepNode::Order () const [inline]

return index of this variable

Definition at line 74 of file CouenneDepGraph.hpp.

7.55.4.3 std::set<DepNode *, compNode>* Couenne::DepNode::DepList () const [inline]

return all variables it depends on

Definition at line 78 of file CouenneDepGraph.hpp.

7.55.4.4 bool Couenne::DepNode::depends (int *xi*, bool *= false*, std::set<DepNode *, compNode> * *already_visited* = NULL) const

does this variable depend on variable with index xi?

7.55.4.5 void Couenne::DepNode::createOrder (DepGraph *)

assign numbering to all nodes of graph

7.55.4.6 void Couenne::DepNode::print(int = 0, bool descend = false) const
debugging procedure

7.55.4.7 enum dep_color& Couenne::DepNode::color() [inline]

return or set color of a node

Definition at line 92 of file CouenneDepGraph.hpp.

7.55.4.8 std::set<DepNode *, compNode>*& Couenne::DepNode::depList() [inline]

index nodes on which this one depends (forward star in dependence graph)

Definition at line 97 of file CouenneDepGraph.hpp.

7.55.4.9 void Couenne::DepNode::replaceIndex(DepNode * oldVarNode, DepNode * newVarNode)

replace the index of a variable with another in the entire graph.

Used when redundant constraints w := x are discovered

7.55.5 Member Data Documentation

7.55.5.1 int Couenne::DepNode::index_ [protected]

index of variable associated with node

Definition at line 43 of file CouenneDepGraph.hpp.

7.55.5.2 std::set<DepNode *, compNode>*& Couenne::DepNode::depList_ [protected]

index nodes on which this one depends (forward star in dependence graph)

Definition at line 47 of file CouenneDepGraph.hpp.

7.55.5.3 int Couenne::DepNode::order_ [protected]

order in which this variable should be updated, evaluated, etc.

Definition at line 50 of file CouenneDepGraph.hpp.

7.55.5.4 enum dep_color Couenne::DepNode::color_ [protected]

color used in DFS for checking cycles

Definition at line 53 of file CouenneDepGraph.hpp.

The documentation for this class was generated from the following file:

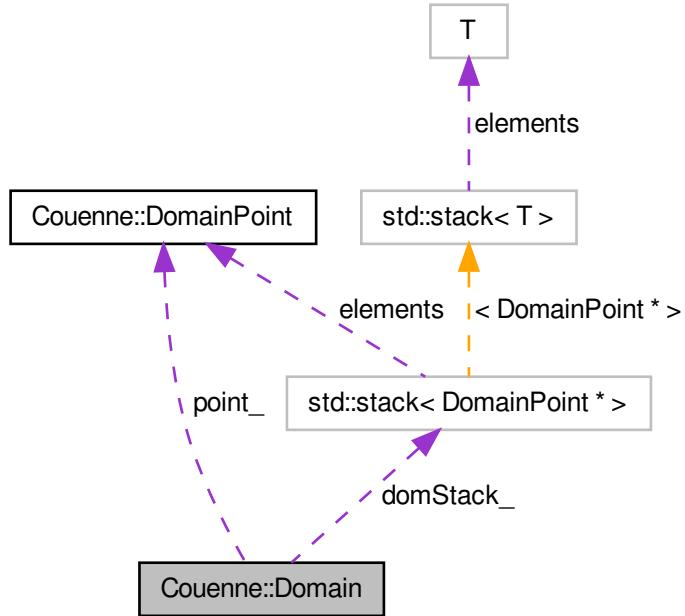
- /home/ted/COIN/trunk/Couenne/src/problem/depGraph/CouenneDepGraph.hpp

7.56 Couenne::Domain Class Reference

Define a dynamic point+bounds, with a way to save and restore previous points+bounds through a LIFO structure.

```
#include <CouenneDomain.hpp>
```

Collaboration diagram for Couenne::Domain:



Public Member Functions

- **Domain ()**
basic constructor
- **Domain (const Domain &src)**
copy constructor
- **~Domain ()**
destructor
- **void push (int dim, CouNumber *x, CouNumber *lb, CouNumber *ub, bool copy=true)**
save current point and start using another
- **void push (int dim, const CouNumber *x, const CouNumber *lb, const CouNumber *ub, bool copy=true)**
save current point and start using another
- **void push (const OsiSolverInterface *si, OsiCuts *cs=NULL, bool copy=true)**
save current point and start using another – retrieve information from solver interface and from previous column cuts
- **void push (const DomainPoint &dp, bool copy=true)**
save current point and start using another
- **void pop ()**
restore previous point
- **DomainPoint * current ()**
return current point
- **CouNumber & x (register int index)**

- **CouNumber & lb** (register int index)
 - current variable*
 - current lower bound*
- **CouNumber & ub** (register int index)
 - current upper bound*
- **CouNumber * x ()**
 - return current variable vector*
- **CouNumber * lb ()**
 - return current lower bound vector*
- **CouNumber * ub ()**
 - return current upper bound vector*

Protected Attributes

- **DomainPoint * point_**
 - current point*
- **std::stack< DomainPoint * > domStack_**
 - stack of saved points*

7.56.1 Detailed Description

Define a dynamic point+bounds, with a way to save and restore previous points+bounds through a LIFO structure.

Definition at line 104 of file CouenneDomain.hpp.

7.56.2 Constructor & Destructor Documentation

7.56.2.1 Couenne::Domain::Domain () [inline]

basic constructor

Definition at line 114 of file CouenneDomain.hpp.

7.56.2.2 Couenne::Domain::Domain (const Domain & src) [inline]

copy constructor

Definition at line 117 of file CouenneDomain.hpp.

7.56.2.3 Couenne::Domain::~Domain ()

destructor

7.56.3 Member Function Documentation

7.56.3.1 void Couenne::Domain::push (int dim, CouNumber * x, CouNumber * lb, CouNumber * ub, bool copy = true)

save current point and start using another

7.56.3.2 void Couenne::Domain::push (int dim, const CouNumber * x, const CouNumber * lb, const CouNumber * ub, bool copy = true)

save current point and start using another

7.56.3.3 void Couenne::Domain::push (const OsiSolverInterface * *si*, OsiCuts * *cs* = NULL, bool *copy* = true)

save current point and start using another – retrieve information from solver interface and from previous column cuts

7.56.3.4 void Couenne::Domain::push (const DomainPoint & *dp*, bool *copy* = true)

save current point and start using another

7.56.3.5 void Couenne::Domain::pop ()

restore previous point

7.56.3.6 DomainPoint* Couenne::Domain::current () [inline]

return current point

Definition at line 154 of file CouenneDomain.hpp.

7.56.3.7 CouNumber& Couenne::Domain::x (register int *index*) [inline]

current variable

Definition at line 156 of file CouenneDomain.hpp.

7.56.3.8 CouNumber& Couenne::Domain::lb (register int *index*) [inline]

current lower bound

Definition at line 157 of file CouenneDomain.hpp.

7.56.3.9 CouNumber& Couenne::Domain::ub (register int *index*) [inline]

current upper bound

Definition at line 158 of file CouenneDomain.hpp.

7.56.3.10 CouNumber* Couenne::Domain::x () [inline]

return current variable vector

Definition at line 160 of file CouenneDomain.hpp.

7.56.3.11 CouNumber* Couenne::Domain::lb () [inline]

return current lower bound vector

Definition at line 161 of file CouenneDomain.hpp.

7.56.3.12 CouNumber* Couenne::Domain::ub () [inline]

return current upper bound vector

Definition at line 162 of file CouenneDomain.hpp.

7.56.4 Member Data Documentation

7.56.4.1 DomainPoint* Couenne::Domain::point_ [protected]

current point

Definition at line 108 of file CouenneDomain.hpp.

7.56.4.2 `std::stack<DomainPoint *> Couenne::Domain::domStack_` [protected]

stack of saved points

Definition at line 109 of file CouenneDomain.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/CouenneDomain.hpp

7.57 Couenne::DomainPoint Class Reference

Define a point in the solution space and the bounds around it.

```
#include <CouenneDomain.hpp>
```

Public Member Functions

- `DomainPoint (int dim, CouNumber *x, CouNumber *lb, CouNumber *ub, bool copy=true)`
constructor
- `DomainPoint (int dim=0, const CouNumber *x=NULL, const CouNumber *lb=NULL, const CouNumber *ub=NULL, bool copy=true)`
constructor
- `~DomainPoint ()`
destructor
- `DomainPoint (const DomainPoint &src)`
copy constructor
- `void resize (int newdim)`
resize domain point (for extending into higher space)
- `int size () const`
return current size
- `int Dimension ()`
return dimension_
- `CouNumber & x (register int index)`
return current variable
- `CouNumber & lb (register int index)`
return current lower bound
- `CouNumber & ub (register int index)`
return current upper bound
- `CouNumber * x ()`
return current variable vector
- `CouNumber * lb ()`
return current lower bound vector
- `CouNumber * ub ()`
return current upper bound vector
- `DomainPoint & operator= (const DomainPoint &src)`
assignment operator
- `bool & isNlp ()`
true if this point is the nlp solution

Protected Attributes

- int `dimension_`
dimension of point
- `CouNumber * x_`
current value of variables
- `CouNumber * lb_`
lower bound
- `CouNumber * ub_`
upper bound
- bool `copied_`
true if data has been copied (so we own it, and have to delete it upon destruction)
- bool `isNlp_`
true if this point comes from an NLP solver (and is thus nlp feasible)

Friends

- class `Domain`

7.57.1 Detailed Description

Define a point in the solution space and the bounds around it.

Definition at line 30 of file CouenneDomain.hpp.

7.57.2 Constructor & Destructor Documentation

7.57.2.1 `Couenne::DomainPoint::DomainPoint (int dim, CouNumber * x, CouNumber * lb, CouNumber * ub, bool copy = true)`

constructor

7.57.2.2 `Couenne::DomainPoint::DomainPoint (int dim = 0, const CouNumber * x = NULL, const CouNumber * lb = NULL, const CouNumber * ub = NULL, bool copy = true)`

constructor

7.57.2.3 `Couenne::DomainPoint::~DomainPoint () [inline]`

destructor

Definition at line 64 of file CouenneDomain.hpp.

7.57.2.4 `Couenne::DomainPoint::DomainPoint (const DomainPoint & src)`

copy constructor

7.57.3 Member Function Documentation

7.57.3.1 `void Couenne::DomainPoint::resize (int newdim)`

resize domain point (for extending into higher space)

7.57.3.2 `int Couenne::DomainPoint::size() const [inline]`

return current size

Definition at line 79 of file CouenneDomain.hpp.

7.57.3.3 `int Couenne::DomainPoint::Dimension() [inline]`

return dimension_

Definition at line 82 of file CouenneDomain.hpp.

7.57.3.4 `CouNumber& Couenne::DomainPoint::x(register int index) [inline]`

return current variable

Definition at line 84 of file CouenneDomain.hpp.

7.57.3.5 `CouNumber& Couenne::DomainPoint::lb(register int index) [inline]`

return current lower bound

Definition at line 85 of file CouenneDomain.hpp.

7.57.3.6 `CouNumber& Couenne::DomainPoint::ub(register int index) [inline]`

return current upper bound

Definition at line 86 of file CouenneDomain.hpp.

7.57.3.7 `CouNumber* Couenne::DomainPoint::x() [inline]`

return current variable vector

Definition at line 88 of file CouenneDomain.hpp.

7.57.3.8 `CouNumber* Couenne::DomainPoint::lb() [inline]`

return current lower bound vector

Definition at line 89 of file CouenneDomain.hpp.

7.57.3.9 `CouNumber* Couenne::DomainPoint::ub() [inline]`

return current upper bound vector

Definition at line 90 of file CouenneDomain.hpp.

7.57.3.10 `DomainPoint& Couenne::DomainPoint::operator=(const DomainPoint & src)`

assignment operator

7.57.3.11 `bool& Couenne::DomainPoint::isNlp() [inline]`

true if this point is the nlp solution

Definition at line 96 of file CouenneDomain.hpp.

7.57.4 Friends And Related Function Documentation

7.57.4.1 friend class Domain [friend]

Definition at line 32 of file CouenneDomain.hpp.

7.57.5 Member Data Documentation

7.57.5.1 int Couenne::DomainPoint::dimension_ [protected]

dimension of point

Definition at line 36 of file CouenneDomain.hpp.

7.57.5.2 CouNumber* Couenne::DomainPoint::x_ [protected]

current value of variables

Definition at line 38 of file CouenneDomain.hpp.

7.57.5.3 CouNumber* Couenne::DomainPoint::lb_ [protected]

lower bound

Definition at line 39 of file CouenneDomain.hpp.

7.57.5.4 CouNumber* Couenne::DomainPoint::ub_ [protected]

upper bound

Definition at line 40 of file CouenneDomain.hpp.

7.57.5.5 bool Couenne::DomainPoint::copied_ [protected]

true if data has been copied (so we own it, and have to delete it upon destruction)

Definition at line 42 of file CouenneDomain.hpp.

7.57.5.6 bool Couenne::DomainPoint::isNlp_ [protected]

true if this point comes from an NLP solver (and is thus nlp feasible)

Definition at line 45 of file CouenneDomain.hpp.

The documentation for this class was generated from the following file:

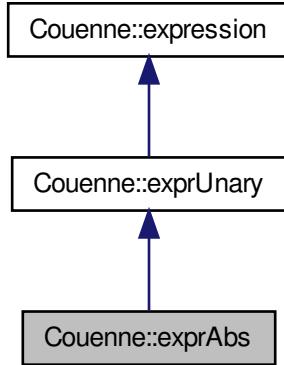
- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneDomain.hpp](#)

7.58 Couenne::exprAbs Class Reference

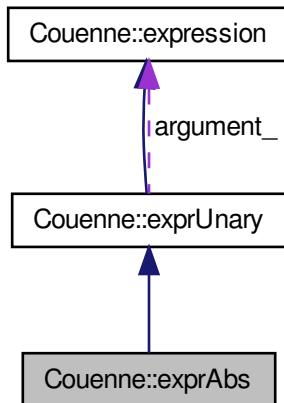
class for $|f(x)|$

```
#include <CouenneExprAbs.hpp>
```

Inheritance diagram for Couenne::exprAbs:



Collaboration diagram for Couenne::exprAbs:



Public Member Functions

- [exprAbs \(expression *al\)](#)
Constructor.
- [unary_function F \(\)](#)
The operator's function.
- [expression * clone \(Domain *d=NULL\) const](#)

- *cloning method*
- std::string **printOp () const**
output
- CouNumber **gradientNorm (const double *x)**
return L_2 norm of gradient at given point
- expression * **differentiate (int index)**
differentiation
- virtual void **getBounds (expression *&, expression *&)**
Get lower and upper bound of an expression (if any)
- virtual void **getBounds (CouNumber &lb, CouNumber &ub)**
Get value of lower and upper bound of an expression (if any)
- void **generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *t_chg_bounds =NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)**
*generate equality between *this and *w*
- enum **expr_type code ()**
code for comparisons
- bool **isInteger ()**
is this expression integer?
- bool **impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)**
implied bound processing
- virtual CouNumber **selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)**
set up branching object by evaluating many branching points for each expression's arguments
- virtual void **closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right)**
const
closest feasible points in function in both directions
- virtual bool **isCuttable (CouenneProblem *problem, int index) const**
can this expression be further linearized or are we on its concave ("bad") side

Additional Inherited Members

7.58.1 Detailed Description

class for $|f(x)|$

Definition at line 23 of file CouenneExprAbs.hpp.

7.58.2 Constructor & Destructor Documentation

7.58.2.1 Couenne::exprAbs::exprAbs (expression * al) [inline]

Constructor.

Definition at line 28 of file CouenneExprAbs.hpp.

7.58.3 Member Function Documentation

7.58.3.1 `unary_function Couenne::exprAbs::F() [inline], [virtual]`

The operator's function.

Reimplemented from [Couenne::exprUnary](#).

Definition at line 32 of file CouenneExprAbs.hpp.

7.58.3.2 `expression* Couenne::exprAbs::clone(Domain * d = NULL) const [inline], [virtual]`

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 35 of file CouenneExprAbs.hpp.

7.58.3.3 `std::string Couenne::exprAbs::printOp() const [inline], [virtual]`

output

Reimplemented from [Couenne::exprUnary](#).

Definition at line 39 of file CouenneExprAbs.hpp.

7.58.3.4 `CouNumber Couenne::exprAbs::gradientNorm(const double * x) [inline], [virtual]`

return L2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Definition at line 43 of file CouenneExprAbs.hpp.

7.58.3.5 `expression* Couenne::exprAbs::differentiate(int index) [virtual]`

differentiation

Reimplemented from [Couenne::expression](#).

7.58.3.6 `virtual void Couenne::exprAbs::getBounds(expression *&, expression *&) [virtual]`

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.58.3.7 `virtual void Couenne::exprAbs::getBounds(CouNumber & lb, CouNumber & ub) [virtual]`

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.58.3.8 `void Couenne::exprAbs::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]`

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

7.58.3.9 `enum expr_type Couenne::exprAbs::code() [inline], [virtual]`

code for comparisons

Reimplemented from [Couenne::exprUnary](#).

Definition at line 63 of file CouenneExprAbs.hpp.

7.58.3.10 `bool Couenne::exprAbs::isInteger() [inline], [virtual]`

is this expression integer?

Reimplemented from [Couenne::exprUnary](#).

Definition at line 66 of file CouenneExprAbs.hpp.

7.58.3.11 `bool Couenne::exprAbs::impliedBound(int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [virtual]`

implied bound processing

Reimplemented from [Couenne::expression](#).

7.58.3.12 `virtual CouNumber Couenne::exprAbs::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]`

set up branching object by evaluating many branching points for each expression's arguments

Reimplemented from [Couenne::expression](#).

7.58.3.13 `virtual void Couenne::exprAbs::closestFeasible(expression * varind, expression * vardep, CouNumber & left, CouNumber & right) const [virtual]`

closest feasible points in function in both directions

Reimplemented from [Couenne::expression](#).

7.58.3.14 `virtual bool Couenne::exprAbs::isCuttable(CouenneProblem * problem, int index) const [virtual]`

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

The documentation for this class was generated from the following file:

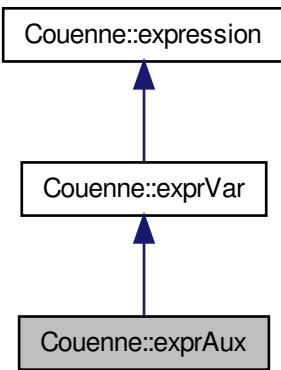
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprAbs.hpp](#)

7.59 Couenne::exprAux Class Reference

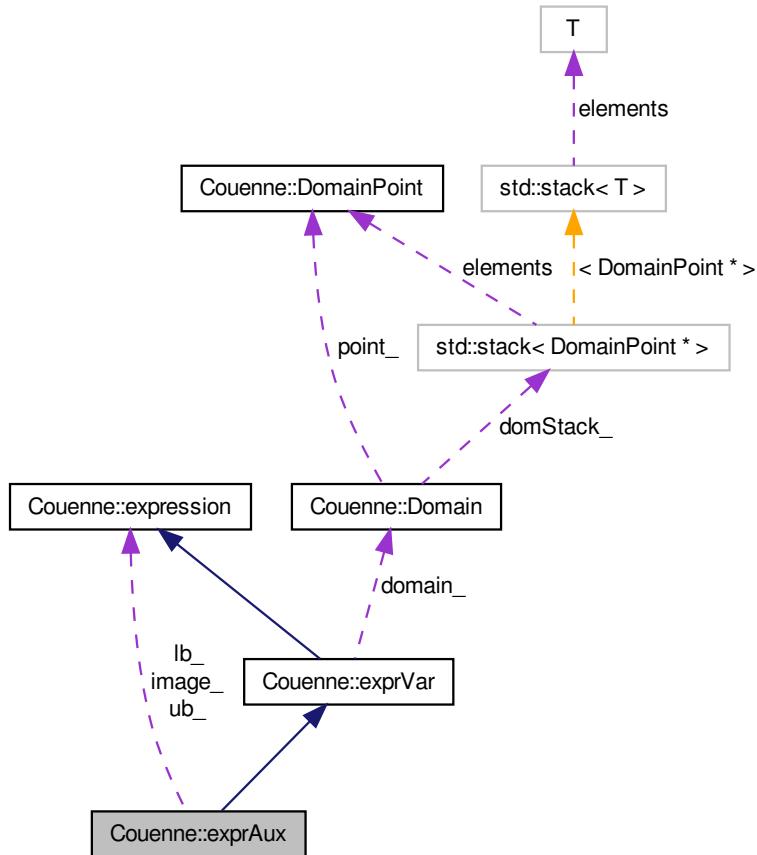
Auxiliary variable.

```
#include <CouenneExprAux.hpp>
```

Inheritance diagram for Couenne::exprAux:



Collaboration diagram for Couenne::exprAux:



Public Types

- enum `intType` { `Unset` = -1, `Continuous`, `Integer` }
integrality type of an auxiliary variable: unset, continuous, integer

Public Member Functions

- enum `nodeType` `Type` () const
Node type.
- `exprAux` (`expression` *, int, int, `intType`=`Unset`, `Domain` *=`NULL`, enum `auxSign`=`expression::AUX_EQ`)
Constructor.
- `exprAux` (`expression` *, `Domain` *=`NULL`, enum `auxSign`=`expression::AUX_EQ`)
Constructor to be used with standardize ([...], false)
- virtual `~exprAux` ()
Destructor.

- `exprAux (const exprAux &, Domain *d=NULL)`
`Copy constructor.`
- `virtual exprVar * clone (Domain *d=NULL) const`
`Cloning method.`
- `expression * Lb ()`
`get lower bound expression`
- `expression * Ub ()`
`get upper bound expression`
- `virtual void print (std::ostream &=std::cout, bool=false) const`
`Print expression.`
- `expression * Image () const`
`The expression associated with this auxiliary variable.`
- `void Image (expression *image)`
`Sets expression associated with this auxiliary variable.`
- `CouNumber operator() ()`
`Null function for evaluating the expression.`
- `int DepList (std::set< int > &deplist, enum dig_type type=ORIG_ONLY)`
`fill in the set with all indices of variables appearing in the expression`
- `expression * simplify ()`
`simplify`
- `int Linearity ()`
`Get a measure of "how linear" the expression is (see CouenneTypes.h)`
- `void crossBounds ()`
`Get lower and upper bound of an expression (if any)`
- `void generateCuts (OsiCuts &, const CouenneCutGenerator *, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
`generate cuts for expression associated with this auxiliary`
- `virtual int rank ()`
`used in rank-based branching variable choice`
- `virtual bool isDefinedInteger ()`
`is this expression defined as integer?`
- `virtual bool isInteger ()`
`is this expression integer?`
- `virtual void setInteger (bool value)`
`Set this variable as integer.`
- `void increaseMult ()`
`Tell this variable appears once more.`
- `void decreaseMult ()`
`Tell this variable appears once less (standardized within exprSum, for instance)`
- `void zeroMult ()`
`Disable this auxiliary variable.`
- `int Multiplicity ()`
`How many times this variable appears.`
- `void linkDomain (Domain *d)`
`link this variable to a domain`
- `bool & top_level ()`
`return top_level_`

- `CouenneObject * properObject (CouenneCutGenerator *c, CouenneProblem *p, Bonmin::BabSetupBase *base, JnlstPtr jnlst)`
`return proper object to handle expression associated with this variable (NULL if this is not an auxiliary)`
- virtual enum `auxSign sign () const`
`return its sign in the definition constraint`

Protected Attributes

- `expression * image_`
`The expression associated with this auxiliary variable.`
- `expression * lb_`
`lower bound, a function of the associated expression and the bounds on the variables in the expression`
- `expression * ub_`
`upper bound, a function of the associated expression and the bounds on the variables in the expression`
- int `rank_`
`used in rank-based branching variable choice: original variables have rank 1; auxiliary $w=f(x)$ has rank $r(w) = r(x)+1$; finally, auxiliary $w=f(x_1,x_2,\dots,x_k)$ has rank $r(w) = 1+\max\{r(x_i):i=1..k\}$.`
- int `multiplicity_`
`number of appearances of this aux in the formulation.`
- enum `intType integer_`
`is this variable integer?`
- bool `top_level_`
`True if this variable replaces the lhs of a constraint, i.e., if it is a top level variable in the DAG of the problem.`
- enum `auxSign sign_`
`"sign" of the defining constraint`

7.59.1 Detailed Description

Auxiliary variable.

It is associated with an expression which depends, in general, on original and/or other auxiliary variables. It is used for AMPL's defined variables (aka common expressions) and to reformulate nonlinear constraints/objectives.

Definition at line 31 of file CouenneExprAux.hpp.

7.59.2 Member Enumeration Documentation

7.59.2.1 enum Couenne::exprAux::intType

integrality type of an auxiliary variable: unset, continuous, integer

Enumerator:

- `Unset`**
- `Continuous`**
- `Integer`**

Definition at line 36 of file CouenneExprAux.hpp.

7.59.3 Constructor & Destructor Documentation

7.59.3.1 `Couenne::exprAux::exprAux (expression * , int , int , intType = Unset, Domain * =NULL, enum auxSign = expression::AUX_EQ)`

Constructor.

7.59.3.2 `Couenne::exprAux::exprAux (expression * , Domain * =NULL, enum auxSign = expression::AUX_EQ)`

Constructor to be used with standardize ([...], false)

7.59.3.3 `virtual Couenne::exprAux::~exprAux () [virtual]`

Destructor.

7.59.3.4 `Couenne::exprAux::exprAux (const exprAux & , Domain * d=NULL)`

Copy constructor.

7.59.4 Member Function Documentation

7.59.4.1 `enum nodeType Couenne::exprAux::Type () const [inline], [virtual]`

`Node` type.

Reimplemented from [Couenne::exprVar](#).

Definition at line 74 of file CouenneExprAux.hpp.

7.59.4.2 `virtual exprVar* Couenne::exprAux::clone (Domain * d=NULL) const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::exprVar](#).

Definition at line 90 of file CouenneExprAux.hpp.

7.59.4.3 `expression* Couenne::exprAux::Lb () [inline], [virtual]`

get lower bound expression

Reimplemented from [Couenne::exprVar](#).

Definition at line 93 of file CouenneExprAux.hpp.

7.59.4.4 `expression* Couenne::exprAux::Ub () [inline], [virtual]`

get upper bound expression

Reimplemented from [Couenne::exprVar](#).

Definition at line 94 of file CouenneExprAux.hpp.

7.59.4.5 `virtual void Couenne::exprAux::print (std::ostream & =std::cout, bool =false) const [virtual]`

Print expression.

Reimplemented from [Couenne::exprVar](#).

7.59.4.6 `expression* Couenne::exprAux::Image() const [inline], [virtual]`

The expression associated with this auxiliary variable.

Reimplemented from [Couenne::expression](#).

Definition at line 101 of file CouenneExprAux.hpp.

7.59.4.7 `void Couenne::exprAux::Image(expression * image) [inline], [virtual]`

Sets expression associated with this auxiliary variable.

Reimplemented from [Couenne::expression](#).

Definition at line 105 of file CouenneExprAux.hpp.

7.59.4.8 `CouNumber Couenne::exprAux::operator()() [inline], [virtual]`

Null function for evaluating the expression.

Reimplemented from [Couenne::exprVar](#).

Definition at line 109 of file CouenneExprAux.hpp.

7.59.4.9 `int Couenne::exprAux::DepList(std::set< int > & depList, enum dig_type type = ORIG_ONLY) [virtual]`

fill in the set with all indices of variables appearing in the expression

Reimplemented from [Couenne::exprVar](#).

7.59.4.10 `expression* Couenne::exprAux::simplify() [virtual]`

simplify

Reimplemented from [Couenne::exprVar](#).

7.59.4.11 `int Couenne::exprAux::Linearity() [inline], [virtual]`

Get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::exprVar](#).

Definition at line 121 of file CouenneExprAux.hpp.

7.59.4.12 `void Couenne::exprAux::crossBounds() [virtual]`

Get lower and upper bound of an expression (if any)

set bounds depending on both branching rules and propagated bounds. To be used after standardization

Reimplemented from [Couenne::exprVar](#).

7.59.4.13 `void Couenne::exprAux::generateCuts(OsiCuts &, const CouenneCutGenerator * , t_chg_bounds * =NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]`

generate cuts for expression associated with this auxiliary

Reimplemented from [Couenne::exprVar](#).

7.59.4.14 `virtual int Couenne::exprAux::rank() [inline], [virtual]`

used in rank-based branching variable choice

Reimplemented from [Couenne::exprVar](#).

Definition at line 140 of file CouenneExprAux.hpp.

7.59.4.15 virtual bool Couenne::exprAux::isDefinedInteger() [inline], [virtual]

is this expression defined as integer?

Reimplemented from [Couenne::exprVar](#).

Definition at line 144 of file CouenneExprAux.hpp.

7.59.4.16 virtual bool Couenne::exprAux::isInteger() [inline], [virtual]

is this expression integer?

Reimplemented from [Couenne::exprVar](#).

Definition at line 153 of file CouenneExprAux.hpp.

7.59.4.17 virtual void Couenne::exprAux::setInteger(bool value) [inline], [virtual]

Set this variable as integer.

Reimplemented from [Couenne::exprVar](#).

Definition at line 165 of file CouenneExprAux.hpp.

7.59.4.18 void Couenne::exprAux::increaseMult() [inline]

Tell this variable appears once more.

Definition at line 169 of file CouenneExprAux.hpp.

7.59.4.19 void Couenne::exprAux::decreaseMult() [inline], [virtual]

Tell this variable appears once less (standardized within [exprSum](#), for instance)

Reimplemented from [Couenne::exprVar](#).

Definition at line 173 of file CouenneExprAux.hpp.

7.59.4.20 void Couenne::exprAux::zeroMult() [inline], [virtual]

Disable this auxiliary variable.

Reimplemented from [Couenne::exprVar](#).

Definition at line 176 of file CouenneExprAux.hpp.

7.59.4.21 int Couenne::exprAux::Multiplicity() [inline], [virtual]

How many times this variable appears.

Reimplemented from [Couenne::expression](#).

Definition at line 179 of file CouenneExprAux.hpp.

7.59.4.22 void Couenne::exprAux::linkDomain(Domain * d) [inline], [virtual]

link this variable to a domain

Reimplemented from [Couenne::exprVar](#).

Definition at line 182 of file CouenneExprAux.hpp.

7.59.4.23 `bool& Couenne::exprAux::top_level() [inline]`

return top_level_

Definition at line 189 of file CouenneExprAux.hpp.

7.59.4.24 `CouenneObject* Couenne::exprAux::properObject(CouenneCutGenerator * c, CouenneProblem * p, Bonmin::BabSetupBase * base, JnlstPtr jnlst) [virtual]`

return proper object to handle expression associated with this variable (NULL if this is not an auxiliary)

Reimplemented from [Couenne::exprVar](#).

7.59.4.25 `virtual enum auxSign Couenne::exprAux::sign() const [inline], [virtual]`

return its sign in the definition constraint

Reimplemented from [Couenne::exprVar](#).

Definition at line 200 of file CouenneExprAux.hpp.

7.59.5 Member Data Documentation

7.59.5.1 `expression* Couenne::exprAux::image_ [protected]`

The expression associated with this auxiliary variable.

Definition at line 41 of file CouenneExprAux.hpp.

7.59.5.2 `expression* Couenne::exprAux::lb_ [protected]`

lower bound, a function of the associated expression and the bounds on the variables in the expression

Definition at line 45 of file CouenneExprAux.hpp.

7.59.5.3 `expression* Couenne::exprAux::ub_ [protected]`

upper bound, a function of the associated expression and the bounds on the variables in the expression

Definition at line 49 of file CouenneExprAux.hpp.

7.59.5.4 `int Couenne::exprAux::rank_ [protected]`

used in rank-based branching variable choice: original variables have rank 1; auxiliary $w=f(x)$ has rank $r(w) = r(x)+1$; finally, auxiliary $w=f(x_1,x_2,\dots,x_k)$ has rank $r(w) = 1+\max\{r(x_i):i=1..k\}$.

Definition at line 54 of file CouenneExprAux.hpp.

7.59.5.5 `int Couenne::exprAux::multiplicity_ [protected]`

number of appearances of this aux in the formulation.

The more times it occurs in the formulation, the more implication its branching has on other variables

Definition at line 59 of file CouenneExprAux.hpp.

7.59.5.6 `enum intType Couenne::exprAux::integer_ [protected]`

is this variable integer?

Definition at line 62 of file CouenneExprAux.hpp.

7.59.5.7 bool Couenne::exprAux::top_level_ [protected]

True if this variable replaces the lhs of a constraint, i.e., if it is a top level variable in the DAG of the problem.

Definition at line 66 of file CouenneExprAux.hpp.

7.59.5.8 enum auxSign Couenne::exprAux::sign_ [protected]

"sign" of the defining constraint

Definition at line 69 of file CouenneExprAux.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprAux.hpp

7.60 Couenne::exprBinProd Class Reference

class for $\prod_{i=1}^n f_i(x)$ with $f_i(x)$ all binary

```
#include <CouenneExprBinProd.hpp>
```

Public Member Functions

- **exprBinProd (expression **, int)**
Constructor.
- **exprBinProd (expression *, expression *)**
Constructor with two arguments.
- **CouNumber gradientNorm (const double *x)**
return l-2 norm of gradient at given point
- **expression * differentiate (int index)**
differentiation
- **expression * simplify ()**
simplification
- **virtual int Linearity ()**
get a measure of "how linear" the expression is:
- **virtual void getBounds (expression *&, expression *&)**
Get lower and upper bound of an expression (if any)
- **virtual void getBounds (CouNumber &lb, CouNumber &ub)**
Get value of lower and upper bound of an expression (if any)
- **virtual exprAux * standardize (CouenneProblem *p, bool addAux=true)**
reduce expression in standard form, creating additional aux variables (and constraints)
- **void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)**
*generate equality between *this and *w*
- **virtual enum expr_type code ()**
code for comparison
- **bool impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum Couenne::expression::auxSign=Couenne::expression::AUX_EQ)**
implied bound processing

- virtual `CouNumber selectBranch` (const `CouenneObject` *obj, const `OsiBranchingInformation` *info, `expression` *&var, double *&brpts, double *&brDist, int &way)
set up branching object by evaluating many branching points for each expression's arguments
- virtual void `closestFeasible` (`expression` *varind, `expression` *vardep, `CouNumber` &left, `CouNumber` &right) const
compute y^{lv} and y^{uv} for Violation Transfer algorithm

Protected Member Functions

- `CouNumber balancedMul` (const `OsiBranchingInformation` *info, int index, int wind)
balanced strategy for branching point selection in products
- virtual bool `isCuttable` (`CouenneProblem` *problem, int index) const
can this expression be further linearized or are we on its concave ("bad") side

7.60.1 Detailed Description

class for $\prod_{i=1}^n f_i(x)$ with $f_i(x)$ all binary

Definition at line 21 of file CouenneExprBinProd.hpp.

7.60.2 Constructor & Destructor Documentation

7.60.2.1 Couenne::exprBinProd::exprBinProd (`expression` ** , int)

Constructor.

7.60.2.2 Couenne::exprBinProd::exprBinProd (`expression` * , `expression` *)

Constructor with two arguments.

7.60.3 Member Function Documentation

7.60.3.1 Couenne::exprBinProd::gradientNorm (const `double` * `x`)

return l-2 norm of gradient at given point

7.60.3.2 `expression`* Couenne::exprBinProd::differentiate (int `index`)

differentiation

7.60.3.3 `expression`* Couenne::exprBinProd::simplify ()

simplification

7.60.3.4 virtual int Couenne::exprBinProd::Linearity () [virtual]

get a measure of "how linear" the expression is:

7.60.3.5 virtual void Couenne::exprBinProd::getBounds (`expression` *& , `expression` *&) [virtual]

Get lower and upper bound of an expression (if any)

7.60.3.6 virtual void Couenne::exprBinProd::getBounds (CouNumber & *lb*, CouNumber & *ub*) [virtual]

Get value of lower and upper bound of an expression (if any)

7.60.3.7 virtual exprAux* Couenne::exprBinProd::standardize (CouenneProblem * *p*, bool *addAux* = true) [virtual]

reduce expression in standard form, creating additional aux variables (and constraints)

7.60.3.8 void Couenne::exprBinProd::generateCuts (expression * *w*, OsiCuts & *cs*, const CouenneCutGenerator * *cg*, t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY)

generate equality between *this and *w

7.60.3.9 virtual enum expr_type Couenne::exprBinProd::code () [inline], [virtual]

code for comparison

Definition at line 61 of file CouenneExprBinProd.hpp.

7.60.3.10 bool Couenne::exprBinProd::impliedBound (int , CouNumber * , CouNumber * , t_chg_bounds * , enum Couenne::expression::auxSign = Couenne::expression::AUX_EQ)

implied bound processing

7.60.3.11 virtual CouNumber Couenne::exprBinProd::selectBranch (const CouenneObject * *obj*, const OsiBranchingInformation * *info*, expression *& *var*, double *& *brpts*, double *& *brDist*, int & *way*) [virtual]

set up branching object by evaluating many branching points for each expression's arguments

7.60.3.12 virtual void Couenne::exprBinProd::closestFeasible (expression * *varind*, expression * *vardep*, CouNumber & *left*, CouNumber & *right*) const [virtual]

compute y^{lv} and y^{uv} for Violation Transfer algorithm

7.60.3.13 CouNumber Couenne::exprBinProd::balancedMul (const OsiBranchingInformation * *info*, int *index*, int *wind*) [protected]

balanced strategy for branching point selection in products

7.60.3.14 virtual bool Couenne::exprBinProd::isCuttable (CouenneProblem * *problem*, int *index*) const [inline], [protected], [virtual]

can this expression be further linearized or are we on its concave ("bad") side

Definition at line 89 of file CouenneExprBinProd.hpp.

The documentation for this class was generated from the following file:

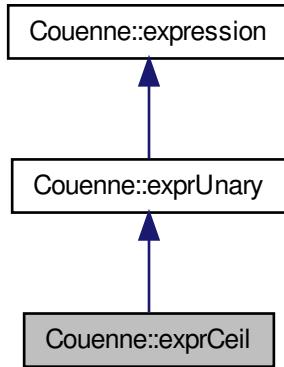
- /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprBinProd.hpp

7.61 Couenne::exprCeil Class Reference

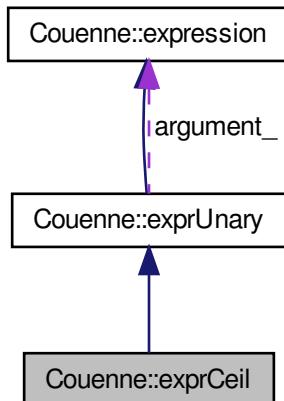
class ceiling, $\lceil f(x) \rceil$

```
#include <CouenneExprCeil.hpp>
```

Inheritance diagram for Couenne::exprCeil:



Collaboration diagram for Couenne::exprCeil:



Public Member Functions

- [exprCeil \(expression *arg\)](#)
constructor, destructor
- [expression * clone \(Domain *d=NULL\) const](#)
cloning method
- [unary_function F \(\)](#)

- `std::string printOp () const`
print operator
- `CouNumber gradientNorm (const double *x)`
return L2 norm of gradient at given point
- `expression * differentiate (int index)`
obtain derivative of expression
- `void getBounds (expression *&, expression *&)`
Get lower and upper bound of an expression (if any)
- `void getBounds (CouNumber &lb, CouNumber &ub)`
Get value of lower and upper bound of an expression.
- `void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *chgs = NULL, int = -1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
*generate equality between *this and *w*
- `virtual enum expr_type code ()`
code for comparisons
- `bool impliedBound (int index, CouNumber *l, CouNumber *u, t_chg_bounds *chgs, enum auxSign=expression::AUX_EQ)`
implied bound processing
- `virtual CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
Set up branching object by evaluating many branching points for each expression's arguments.
- `virtual void closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right) const`
closest feasible points in function in both directions
- `virtual bool isCuttable (CouenneProblem *problem, int index) const`
can this expression be further linearized or are we on its concave ("bad") side?

Additional Inherited Members

7.61.1 Detailed Description

class ceiling, $\lceil f(x) \rceil$

Definition at line 20 of file CouenneExprCeil.hpp.

7.61.2 Constructor & Destructor Documentation

7.61.2.1 Couenne::exprCeil::exprCeil (`expression * arg`) [inline]

constructor, destructor

Definition at line 25 of file CouenneExprCeil.hpp.

7.61.3 Member Function Documentation

7.61.3.1 `expression* Couenne::exprCeil::clone (Domain * d = NULL) const` [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 29 of file CouenneExprCeil.hpp.

7.61.3.2 unary_function Couenne::exprCeil::F() [inline], [virtual]

the operator itself (e.g. sin, log...)

Reimplemented from [Couenne::exprUnary](#).

Definition at line 33 of file CouenneExprCeil.hpp.

7.61.3.3 std::string Couenne::exprCeil::printOp() const [inline], [virtual]

print operator

Reimplemented from [Couenne::exprUnary](#).

Definition at line 37 of file CouenneExprCeil.hpp.

7.61.3.4 CouNumber Couenne::exprCeil::gradientNorm(const double *x) [inline], [virtual]

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Definition at line 41 of file CouenneExprCeil.hpp.

7.61.3.5 expression* Couenne::exprCeil::differentiate(int index) [virtual]

obtain derivative of expression

Reimplemented from [Couenne::expression](#).

7.61.3.6 void Couenne::exprCeil::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.61.3.7 void Couenne::exprCeil::getBounds(CouNumber & lb, CouNumber & ub) [virtual]

Get value of lower and upper bound of an expression.

Reimplemented from [Couenne::expression](#).

7.61.3.8 void Couenne::exprCeil::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

7.61.3.9 virtual enum expr_type Couenne::exprCeil::code() [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::exprUnary](#).

Definition at line 63 of file CouenneExprCeil.hpp.

7.61.3.10 `bool Couenne::exprCeil::impliedBound (int index, CouNumber * l, CouNumber * u, t_chg_bounds * chg, enum auxSign = expression::AUX_EQ) [inline], [virtual]`

implied bound processing

Reimplemented from [Couenne::expression](#).

Definition at line 67 of file CouenneExprCeil.hpp.

7.61.3.11 `virtual CouNumber Couenne::exprCeil::selectBranch (const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [inline], [virtual]`

Set up branching object by evaluating many branching points for each expression's arguments.

Reimplemented from [Couenne::expression](#).

Definition at line 75 of file CouenneExprCeil.hpp.

7.61.3.12 `virtual void Couenne::exprCeil::closestFeasible (expression * varind, expression * vardep, CouNumber & left, CouNumber & right) const [virtual]`

closest feasible points in function in both directions

Reimplemented from [Couenne::expression](#).

7.61.3.13 `virtual bool Couenne::exprCeil::isCuttable (CouenneProblem * problem, int index) const [inline], [virtual]`

can this expression be further linearized or are we on its concave ("bad") side?

Reimplemented from [Couenne::expression](#).

Definition at line 90 of file CouenneExprCeil.hpp.

The documentation for this class was generated from the following file:

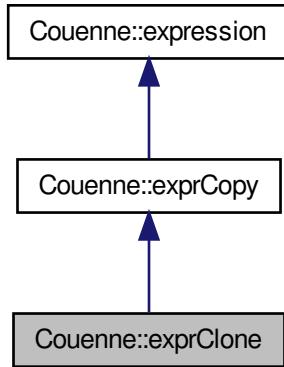
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprCeil.hpp](#)

7.62 Couenne::exprClone Class Reference

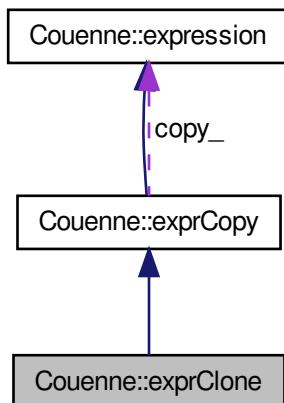
expression clone (points to another expression)

```
#include <CouenneExprClone.hpp>
```

Inheritance diagram for Couenne::exprClone:



Collaboration diagram for Couenne::exprClone:



Public Member Functions

- `exprClone (expression *copy)`
Constructor.
- `exprClone (const exprClone &e, Domain *d=NULL)`
copy constructor
- `expression * clone (Domain *d=NULL) const`

- cloning method*
- virtual [~exprClone \(\)](#)

Destructor.
 - virtual void [print \(std::ostream &out=std::cout, bool descend=false\) const](#)

Printing.
 - [CouNumber Value \(\) const](#)

value
 - [CouNumber operator\(\) \(\)](#)

null function for evaluating the expression

Additional Inherited Members

7.62.1 Detailed Description

expression clone (points to another expression)

Definition at line 24 of file CouenneExprClone.hpp.

7.62.2 Constructor & Destructor Documentation

7.62.2.1 Couenne::exprClone::exprClone(**expression * copy**) [inline]

Constructor.

Definition at line 29 of file CouenneExprClone.hpp.

7.62.2.2 Couenne::exprClone::exprClone(**const exprClone & e, Domain * d=NULL**) [inline]

copy constructor

Definition at line 33 of file CouenneExprClone.hpp.

7.62.2.3 virtual Couenne::exprClone::~exprClone() [inline], [virtual]

Destructor.

Definition at line 47 of file CouenneExprClone.hpp.

7.62.3 Member Function Documentation

7.62.3.1 **expression* Couenne::exprClone::clone(Domain * d=NULL) const** [inline], [virtual]

cloning method

Reimplemented from [Couenne::exprCopy](#).

Definition at line 38 of file CouenneExprClone.hpp.

7.62.3.2 virtual void Couenne::exprClone::print(**std::ostream & out = std::cout, bool descend = false**) const [virtual]

Printing.

Reimplemented from [Couenne::exprCopy](#).

7.62.3.3 CouNumber Couenne::exprClone::Value() const [inline], [virtual]

value

Reimplemented from [Couenne::exprCopy](#).

Definition at line 58 of file CouenneExprClone.hpp.

7.62.3.4 CouNumber Couenne::exprClone::operator()() [inline], [virtual]

null function for evaluating the expression

Reimplemented from [Couenne::exprCopy](#).

Definition at line 62 of file CouenneExprClone.hpp.

The documentation for this class was generated from the following file:

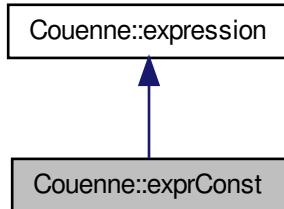
- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprClone.hpp](#)

7.63 Couenne::exprConst Class Reference

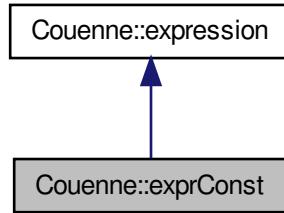
constant-type operator

```
#include <CouenneExprConst.hpp>
```

Inheritance diagram for Couenne::exprConst:



Collaboration diagram for Couenne::exprConst:



Public Member Functions

- enum `nodeType Type () const`
node type
- `CouNumber Value () const`
value of expression
- `exprConst (CouNumber value)`
Constructor.
- `exprConst (const exprConst &e, Domain *d=NULL)`
Copy constructor.
- virtual `expression * clone (Domain *d=NULL) const`
Cloning method.
- void `print (std::ostream &out=std::cout, bool=false) const`
I/O.
- `CouNumber operator() ()`
return constant's value
- `expression * differentiate (int)`
differentiation
- int `dependsOn (int *ind, int n, enum dig_type type=STOP_AT_AUX)`
dependence on variable set
- int `Linearity ()`
get a measure of "how linear" the expression is (see CouenneTypes.h)
- void `getBounds (expression *&lower, expression *&upper)`
Get lower and upper bound of an expression (if any)
- void `getBounds (CouNumber &lower, CouNumber &upper)`
Get value of lower and upper bound of an expression (if any)
- void `generateCuts (expression *, OsiCuts &, const CouenneCutGenerator *, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
generate convexification cut for constraint w = this
- virtual enum `expr_type code ()`
code for comparisons
- virtual bool `isInteger ()`

- *is this expression integer?*
- virtual int **rank** ()
 - used in rank-based branching variable choice*

Additional Inherited Members

7.63.1 Detailed Description

constant-type operator

Definition at line 23 of file CouenneExprConst.hpp.

7.63.2 Constructor & Destructor Documentation

7.63.2.1 Couenne::exprConst::exprConst(CouNumber value) [inline]

Constructor.

Definition at line 41 of file CouenneExprConst.hpp.

7.63.2.2 Couenne::exprConst::exprConst(const exprConst & e, Domain * d = NULL) [inline]

Copy constructor.

Definition at line 45 of file CouenneExprConst.hpp.

7.63.3 Member Function Documentation

7.63.3.1 enum nodeType Couenne::exprConst::Type() const [inline], [virtual]

node type

Reimplemented from [Couenne::expression](#).

Definition at line 33 of file CouenneExprConst.hpp.

7.63.3.2 CouNumber Couenne::exprConst::Value() const [inline], [virtual]

value of expression

Reimplemented from [Couenne::expression](#).

Definition at line 37 of file CouenneExprConst.hpp.

7.63.3.3 virtual expression* Couenne::exprConst::clone(Domain * d = NULL) const [inline], [virtual]

Cloning method.

Reimplemented from [Couenne::expression](#).

Definition at line 49 of file CouenneExprConst.hpp.

7.63.3.4 void Couenne::exprConst::print(std::ostream & out = std::cout, bool = false) const [inline], [virtual]

I/O.

Reimplemented from [Couenne::expression](#).

Definition at line 53 of file CouenneExprConst.hpp.

7.63.3.5 CouNumber Couenne::exprConst::operator() () [inline], [virtual]

return constant's value

Implements [Couenne::expression](#).

Definition at line 58 of file CouenneExprConst.hpp.

7.63.3.6 expression* Couenne::exprConst::differentiate(int) [inline], [virtual]

differentiation

Reimplemented from [Couenne::expression](#).

Definition at line 62 of file CouenneExprConst.hpp.

7.63.3.7 int Couenne::exprConst::dependsOn(int *ind, int n, enum dig_type type = STOP_AT_AUX) [inline], [virtual]

dependence on variable set

Reimplemented from [Couenne::expression](#).

Definition at line 66 of file CouenneExprConst.hpp.

7.63.3.8 int Couenne::exprConst::Linearity() [inline], [virtual]

get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::expression](#).

Definition at line 70 of file CouenneExprConst.hpp.

7.63.3.9 void Couenne::exprConst::getBounds(expression *& lower, expression *& upper) [inline], [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

Definition at line 74 of file CouenneExprConst.hpp.

7.63.3.10 void Couenne::exprConst::getBounds(CouNumber & lower, CouNumber & upper) [inline], [virtual]

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

Definition at line 80 of file CouenneExprConst.hpp.

7.63.3.11 void Couenne::exprConst::generateCuts(expression *, OsiCuts &, const CouenneCutGenerator * , t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

generate convexification cut for constraint w = this

Reimplemented from [Couenne::expression](#).

7.63.3.12 virtual enum expr_type Couenne::exprConst::code() [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::expression](#).

Definition at line 91 of file CouenneExprConst.hpp.

7.63.3.13 virtual bool Couenne::exprConst::isInteger() [inline], [virtual]

is this expression integer?

Reimplemented from [Couenne::expression](#).

Definition at line 95 of file CouenneExprConst.hpp.

7.63.3.14 virtual int Couenne::exprConst::rank() [inline], [virtual]

used in rank-based branching variable choice

Reimplemented from [Couenne::expression](#).

Definition at line 99 of file CouenneExprConst.hpp.

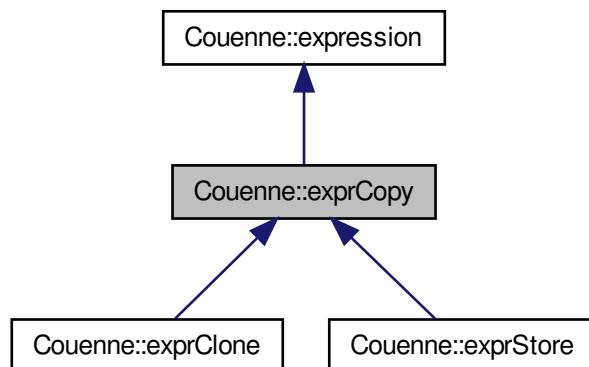
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprConst.hpp](#)

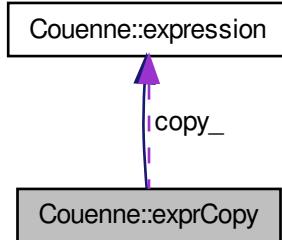
7.64 Couenne::exprCopy Class Reference

```
#include <CouenneExprCopy.hpp>
```

Inheritance diagram for Couenne::exprCopy:



Collaboration diagram for Couenne::exprCopy:



Public Member Functions

- enum `nodeType Type () const`
node type
- `exprCopy (expression *copy)`
Empty constructor - used in cloning method of `exprClone`.
- `exprCopy (const exprCopy &e, Domain *d=NULL)`
Copy constructor.
- virtual `~exprCopy ()`
Destructor – CAUTION: this is the only destructive destructor, `exprClone` and `exprStore` do not destroy anything.
- virtual `expression * clone (Domain *d=NULL) const`
Cloning method.
- const `expression * Original () const`
If this is an `exprClone` of a `exprClone` of an `expr???`, point to the original `expr???` instead of an `exprClone` – improves computing efficiency.
- bool `isACopy () const`
return true if this is a copy of something, i.e.
- `expression * Copy () const`
return copy of this expression (only makes sense in `exprCopy`)
- `expression * Image () const`
return pointer to corresponding expression (for auxiliary variables only)
- int `Index () const`
Get variable index in problem.
- int `nArgs () const`
Return number of arguments (when applicable, that is, with N-ary functions)
- `expression ** ArgList () const`
return arglist (when applicable, that is, with N-ary functions)
- void `ArgList (expression **al)`
set arglist (used in deleting nodes without deleting children)
- `expression * Argument () const`
return argument (when applicable, i.e., with univariate functions)

- `expression ** ArgPtr ()`
`return pointer to argument (when applicable, i.e., with univariate functions)`
- `virtual void print (std::ostream &out=std::cout, bool descend=false) const`
`I/O.`
- `virtual CouNumber Value () const`
`value`
- `virtual CouNumber operator() ()`
`null function for evaluating the expression`
- `CouNumber gradientNorm (const double *x)`
`return l-2 norm of gradient at given point`
- `expression * differentiate (int index)`
`differentiation`
- `int DepList (std::set< int > &deplist, enum dig_type type=ORIG_ONLY)`
`fill in the set with all indices of variables appearing in the expression`
- `expression * simplify ()`
`simplify expression (useful for derivatives)`
- `int Linearity ()`
`get a measure of "how linear" the expression is (see CouenneTypes.h)`
- `bool isInteger ()`
`is this expression integer?`
- `virtual bool isDefinedInteger ()`
`is this expression DEFINED as integer?`
- `void getBounds (expression *&lower, expression *&upper)`
`Get lower and upper bound of an expression (if any)`
- `void getBounds (CouNumber &lower, CouNumber &upper)`
`Get value of lower and upper bound of an expression (if any)`
- `exprAux * standardize (CouenneProblem *p, bool addAux=true)`
`Create standard formulation of this expression.`
- `void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *chg=NULL, int wind=-1, CouNumber lb=-COUENNE_INFINITY, CouNumber ub=COUENNE_INFINITY)`
`generate convexification cut for constraint w = this`
- `enum expr_type code ()`
`code for comparisons`
- `enum convexity convexity () const`
`either CONVEX, CONCAVE, AFFINE, or NONCONVEX`
- `int compare (expression &e)`
`compare this with other expression`
- `int rank ()`
`used in rank-based branching variable choice`
- `bool impliedBound (int wind, CouNumber *l, CouNumber *u, t_chg_bounds *chg)`
`implied bound processing`
- `int Multiplicity ()`
`multiplicity of a variable: how many times this variable occurs in expressions throughout the problem`
- `CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
`Set up branching object by evaluating many branching points for each expression's arguments.`
- `void replace (exprVar *, exprVar *)`

- `replace occurrence of a variable with another variable`
- void `fillDepSet (std::set< DepNode *, compNode > *dep, DepGraph *g)`
fill in dependence structure
- void `realign (const CouenneProblem *p)`
redirect variables to proper variable vector
- bool `isBijective () const`
indicating if function is monotonically increasing
- CouNumber `inverse (expression *vardep) const`
compute the inverse function
- void `closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right) const`
closest feasible points in function in both directions
- bool `isCuttable (CouenneProblem *problem, int index) const`
can this expression be further linearized or are we on its concave ("bad") side

Protected Attributes

- `expression * copy_`
the expression this object is a (reference) copy of
- `CouNumber value_`
saved value to be used by `exprStore` expressions

Additional Inherited Members

7.64.1 Detailed Description

Definition at line 25 of file CouenneExprCopy.hpp.

7.64.2 Constructor & Destructor Documentation

7.64.2.1 Couenne::exprCopy::exprCopy (`expression * copy`) [inline]

Empty constructor - used in cloning method of `exprClone`.

Constructor

Definition at line 45 of file CouenneExprCopy.hpp.

7.64.2.2 Couenne::exprCopy::exprCopy (`const exprCopy & e, Domain * d = NULL`)

Copy constructor.

7.64.2.3 virtual Couenne::exprCopy::~exprCopy () [inline], [virtual]

Destructor – CAUTION: this is the only destructive destructor, `exprClone` and `exprStore` do not destroy anything.

Definition at line 55 of file CouenneExprCopy.hpp.

7.64.3 Member Function Documentation

7.64.3.1 `enum nodeType Couenne::exprCopy::Type() const [inline], [virtual]`

node type

Reimplemented from [Couenne::expression](#).

Definition at line 38 of file CouenneExprCopy.hpp.

7.64.3.2 `virtual expression* Couenne::exprCopy::clone(Domain * d=NULL) const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprStore](#), and [Couenne::exprClone](#).

Definition at line 61 of file CouenneExprCopy.hpp.

7.64.3.3 `const expression* Couenne::exprCopy::Original() const [inline], [virtual]`

If this is an [exprClone](#) of a [exprClone](#) of an [expr???](#), point to the original [expr???](#) instead of an [exprClone](#) – improves computing efficiency.

Reimplemented from [Couenne::expression](#).

Definition at line 67 of file CouenneExprCopy.hpp.

7.64.3.4 `bool Couenne::exprCopy::isaCopy() const [inline], [virtual]`

return true if this is a copy of something, i.e.

if it is an [exprCopy](#) or derives

Reimplemented from [Couenne::expression](#).

Definition at line 72 of file CouenneExprCopy.hpp.

7.64.3.5 `expression* Couenne::exprCopy::Copy() const [inline], [virtual]`

return copy of this expression (only makes sense in [exprCopy](#))

Reimplemented from [Couenne::expression](#).

Definition at line 76 of file CouenneExprCopy.hpp.

7.64.3.6 `expression* Couenne::exprCopy::Image() const [inline], [virtual]`

return pointer to corresponding expression (for auxiliary variables only)

Reimplemented from [Couenne::expression](#).

Definition at line 80 of file CouenneExprCopy.hpp.

7.64.3.7 `int Couenne::exprCopy::Index() const [inline], [virtual]`

Get variable index in problem.

Reimplemented from [Couenne::expression](#).

Definition at line 84 of file CouenneExprCopy.hpp.

7.64.3.8 `int Couenne::exprCopy::nArgs() const [inline], [virtual]`

Return number of arguments (when applicable, that is, with N-ary functions)

Reimplemented from [Couenne::expression](#).

Definition at line 88 of file CouenneExprCopy.hpp.

7.64.3.9 `expression** Couenne::exprCopy::ArgList() const [inline], [virtual]`

return arglist (when applicable, that is, with N-ary functions)

Reimplemented from [Couenne::expression](#).

Definition at line 92 of file CouenneExprCopy.hpp.

7.64.3.10 `void Couenne::exprCopy::ArgList(expression ** al) [inline], [virtual]`

set arglist (used in deleting nodes without deleting children)

Reimplemented from [Couenne::expression](#).

Definition at line 96 of file CouenneExprCopy.hpp.

7.64.3.11 `expression* Couenne::exprCopy::Argument() const [inline], [virtual]`

return argument (when applicable, i.e., with univariate functions)

Reimplemented from [Couenne::expression](#).

Definition at line 100 of file CouenneExprCopy.hpp.

7.64.3.12 `expression** Couenne::exprCopy::ArgPtr() [inline], [virtual]`

return pointer to argument (when applicable, i.e., with univariate functions)

Reimplemented from [Couenne::expression](#).

Definition at line 104 of file CouenneExprCopy.hpp.

7.64.3.13 `virtual void Couenne::exprCopy::print(std::ostream & out = std::cout, bool descend = false) const [virtual]`

I/O.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprClone](#), and [Couenne::exprStore](#).

7.64.3.14 `virtual CouNumber Couenne::exprCopy::Value() const [inline], [virtual]`

value

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprClone](#).

Definition at line 112 of file CouenneExprCopy.hpp.

7.64.3.15 `virtual CouNumber Couenne::exprCopy::operator()() [inline], [virtual]`

null function for evaluating the expression

Implements [Couenne::expression](#).

Reimplemented in [Couenne::exprClone](#), and [Couenne::exprStore](#).

Definition at line 116 of file CouenneExprCopy.hpp.

7.64.3.16 CouNumber Couenne::exprCopy::gradientNorm (const double * x) [inline], [virtual]

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Definition at line 122 of file CouenneExprCopy.hpp.

7.64.3.17 expression* Couenne::exprCopy::differentiate (int index) [inline], [virtual]

differentiation

Reimplemented from [Couenne::expression](#).

Definition at line 126 of file CouenneExprCopy.hpp.

7.64.3.18 int Couenne::exprCopy::DepList (std::set< int > & depList, enum dig_type type = ORIG_ONLY) [inline], [virtual]

fill in the set with all indices of variables appearing in the expression

Reimplemented from [Couenne::expression](#).

Definition at line 131 of file CouenneExprCopy.hpp.

7.64.3.19 expression* Couenne::exprCopy::simplify () [inline], [virtual]

simplify expression (useful for derivatives)

Reimplemented from [Couenne::expression](#).

Definition at line 136 of file CouenneExprCopy.hpp.

7.64.3.20 int Couenne::exprCopy::Linearity () [inline], [virtual]

get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::expression](#).

Definition at line 140 of file CouenneExprCopy.hpp.

7.64.3.21 bool Couenne::exprCopy::isInteger () [inline], [virtual]

is this expression integer?

Reimplemented from [Couenne::expression](#).

Definition at line 143 of file CouenneExprCopy.hpp.

7.64.3.22 virtual bool Couenne::exprCopy::isDefinedInteger () [inline], [virtual]

is this expression DEFINED as integer?

Reimplemented from [Couenne::expression](#).

Definition at line 147 of file CouenneExprCopy.hpp.

7.64.3.23 void Couenne::exprCopy::getBounds (expression *& lower, expression *& upper) [inline], [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

Definition at line 151 of file CouenneExprCopy.hpp.

7.64.3.24 void Couenne::exprCopy::getBounds (CouNumber & lower, CouNumber & upper) [inline], [virtual]

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

Definition at line 155 of file CouenneExprCopy.hpp.

7.64.3.25 exprAux* Couenne::exprCopy::standardize (CouenneProblem * p, bool addAux = true) [inline], [virtual]

Create standard formulation of this expression.

Reimplemented from [Couenne::expression](#).

Definition at line 160 of file CouenneExprCopy.hpp.

7.64.3.26 void Couenne::exprCopy::generateCuts (expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * chg = NULL, int wind = -1, CouNumber lb = -COUENNE_INFINITY, CouNumber ub = COUENNE_INFINITY) [inline], [virtual]

generate convexification cut for constraint w = this

Reimplemented from [Couenne::expression](#).

Definition at line 164 of file CouenneExprCopy.hpp.

7.64.3.27 enum expr_type Couenne::exprCopy::code () [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::expression](#).

Definition at line 173 of file CouenneExprCopy.hpp.

7.64.3.28 enum convexity Couenne::exprCopy::convexity () const [inline], [virtual]

either CONVEX, CONCAVE, AFFINE, or NONCONVEX

Reimplemented from [Couenne::expression](#).

Definition at line 177 of file CouenneExprCopy.hpp.

7.64.3.29 int Couenne::exprCopy::compare (expression & e) [inline], [virtual]

compare this with other expression

Reimplemented from [Couenne::expression](#).

Definition at line 181 of file CouenneExprCopy.hpp.

7.64.3.30 int Couenne::exprCopy::rank () [inline], [virtual]

used in rank-based branching variable choice

Reimplemented from [Couenne::expression](#).

Definition at line 185 of file CouenneExprCopy.hpp.

```
7.64.3.31 bool Couenne::exprCopy::impliedBound( int wind, CouNumber * l, CouNumber * u, t_chg_bounds * chg )  
[inline]
```

implied bound processing

Definition at line 189 of file CouenneExprCopy.hpp.

```
7.64.3.32 int Couenne::exprCopy::Multiplicity( ) [inline], [virtual]
```

multiplicity of a variable: how many times this variable occurs in expressions throughout the problem

Reimplemented from [Couenne::expression](#).

Definition at line 194 of file CouenneExprCopy.hpp.

```
7.64.3.33 CouNumber Couenne::exprCopy::selectBranch( const CouenneObject * obj, const OsiBranchingInformation *  
info, expression *& var, double *& brpts, double *& brDist, int & way ) [inline], [virtual]
```

Set up branching object by evaluating many branching points for each expression's arguments.

Return estimated improvement in objective function

Reimplemented from [Couenne::expression](#).

Definition at line 199 of file CouenneExprCopy.hpp.

```
7.64.3.34 void Couenne::exprCopy::replace( exprVar *, exprVar * ) [virtual]
```

replace occurrence of a variable with another variable

Reimplemented from [Couenne::expression](#).

```
7.64.3.35 void Couenne::exprCopy::fillDepSet( std::set< DepNode *, compNode *> * dep, DepGraph * g ) [inline],  
[virtual]
```

fill in dependence structure

Reimplemented from [Couenne::expression](#).

Definition at line 213 of file CouenneExprCopy.hpp.

```
7.64.3.36 void Couenne::exprCopy::realign( const CouenneProblem * p ) [virtual]
```

redirect variables to proper variable vector

Reimplemented from [Couenne::expression](#).

```
7.64.3.37 bool Couenne::exprCopy::isBijective( ) const [inline], [virtual]
```

indicating if function is monotonically increasing

Reimplemented from [Couenne::expression](#).

Definition at line 221 of file CouenneExprCopy.hpp.

```
7.64.3.38 CouNumber Couenne::exprCopy::inverse( expression * vardep ) const [inline], [virtual]
```

compute the inverse function

Reimplemented from [Couenne::expression](#).

Definition at line 225 of file CouenneExprCopy.hpp.

```
7.64.3.39 void Couenne::exprCopy::closestFeasible( expression * varind, expression * vardep, CouNumber & left,
    CouNumber & right ) const [inline], [virtual]
```

closest feasible points in function in both directions

Reimplemented from [Couenne::expression](#).

Definition at line 229 of file CouenneExprCopy.hpp.

```
7.64.3.40 bool Couenne::exprCopy::isCuttable( CouenneProblem * problem, int index ) const [inline], [virtual]
```

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

Definition at line 235 of file CouenneExprCopy.hpp.

7.64.4 Member Data Documentation

```
7.64.4.1 expression* Couenne::exprCopy::copy_ [protected]
```

the expression this object is a (reference) copy of

Definition at line 30 of file CouenneExprCopy.hpp.

```
7.64.4.2 CouNumber Couenne::exprCopy::value_ [protected]
```

saved value to be used by [exprStore](#) expressions

Definition at line 33 of file CouenneExprCopy.hpp.

The documentation for this class was generated from the following file:

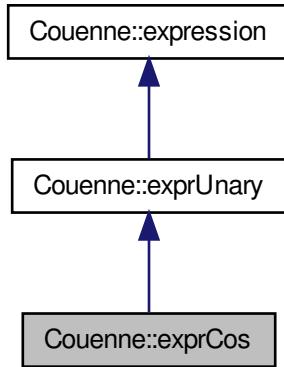
- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprCopy.hpp](#)

7.65 Couenne::exprCos Class Reference

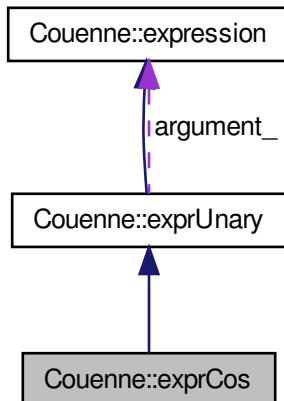
```
class cosine, cos f(x)
```

```
#include <CouenneExprCos.hpp>
```

Inheritance diagram for Couenne::exprCos:



Collaboration diagram for Couenne::exprCos:



Public Member Functions

- [exprCos \(expression *al\)](#)
constructor, destructor
- [expression * clone \(Domain *d=NULL\) const](#)
cloning method
- [unary_function F \(\)](#)

- `the operator itself (e.g. sin, log...)`
- `std::string printOp () const`
`print operator`
- `CouNumber gradientNorm (const double *x)`
`return L2 norm of gradient at given point`
- `expression * differentiate (int index)`
`obtain derivative of expression`
- `void getBounds (expression *&, expression *&)`
`Get lower and upper bound of an expression (if any)`
- `void getBounds (CouNumber &lb, CouNumber &ub)`
`Get value of lower and upper bound of an expression.`
- `void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *chgbounds = NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
`generate equality between *this and *w`
- `virtual enum expr_type code ()`
`code for comparisons`
- `bool impliedBound (int index, CouNumber *l, CouNumber *u, t_chg_bounds *chgbounds, enum auxSign=expression::AUX_EQ)`
`implied bound processing`
- `virtual CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
`Set up branching object by evaluating many branching points for each expression's arguments.`
- `virtual void closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right) const`
`closest feasible points in function in both directions`
- `virtual bool isCuttable (CouenneProblem *problem, int index) const`
`can this expression be further linearized or are we on its concave ("bad") side`

Additional Inherited Members

7.65.1 Detailed Description

class cosine, $\cos f(x)$

Definition at line 20 of file CouenneExprCos.hpp.

7.65.2 Constructor & Destructor Documentation

7.65.2.1 Couenne::exprCos::exprCos (`expression * al`) [inline]

constructor, destructor

Definition at line 25 of file CouenneExprCos.hpp.

7.65.3 Member Function Documentation

7.65.3.1 `expression* Couenne::exprCos::clone (Domain * d = NULL) const` [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 29 of file CouenneExprCos.hpp.

7.65.3.2 unary_function Couenne::exprCos::F() [inline], [virtual]

the operator itself (e.g. sin, log...)

Reimplemented from [Couenne::exprUnary](#).

Definition at line 33 of file CouenneExprCos.hpp.

7.65.3.3 std::string Couenne::exprCos::printOp() const [inline], [virtual]

print operator

Reimplemented from [Couenne::exprUnary](#).

Definition at line 37 of file CouenneExprCos.hpp.

7.65.3.4 CouNumber Couenne::exprCos::gradientNorm(const double *x) [inline], [virtual]

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Definition at line 41 of file CouenneExprCos.hpp.

7.65.3.5 expression* Couenne::exprCos::differentiate(int index) [virtual]

obtain derivative of expression

Reimplemented from [Couenne::expression](#).

7.65.3.6 void Couenne::exprCos::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.65.3.7 void Couenne::exprCos::getBounds(CouNumber & lb, CouNumber & ub) [virtual]

Get value of lower and upper bound of an expression.

Reimplemented from [Couenne::expression](#).

7.65.3.8 void Couenne::exprCos::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =COUENNE_INFINITY) [virtual]

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

7.65.3.9 virtual enum expr_type Couenne::exprCos::code() [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::exprUnary](#).

Definition at line 63 of file CouenneExprCos.hpp.

7.65.3.10 `bool Couenne::exprCos::impliedBound (int index, CouNumber * l, CouNumber * u, t_chg_bounds * chg, enum auxSign = expression::AUX_EQ) [inline], [virtual]`

implied bound processing

Reimplemented from [Couenne::expression](#).

Definition at line 67 of file CouenneExprCos.hpp.

7.65.3.11 `virtual CouNumber Couenne::exprCos::selectBranch (const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [inline], [virtual]`

Set up branching object by evaluating many branching points for each expression's arguments.

Reimplemented from [Couenne::expression](#).

Definition at line 84 of file CouenneExprCos.hpp.

7.65.3.12 `virtual void Couenne::exprCos::closestFeasible (expression * varind, expression * vardep, CouNumber & left, CouNumber & right) const [virtual]`

closest feasible points in function in both directions

Reimplemented from [Couenne::expression](#).

7.65.3.13 `virtual bool Couenne::exprCos::isCuttable (CouenneProblem * problem, int index) const [inline], [virtual]`

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

Definition at line 99 of file CouenneExprCos.hpp.

The documentation for this class was generated from the following file:

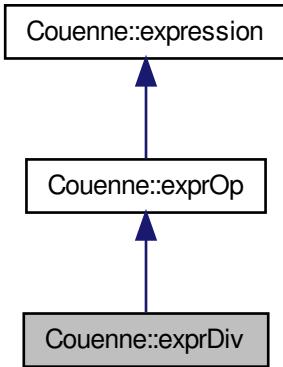
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprCos.hpp](#)

7.66 Couenne::exprDiv Class Reference

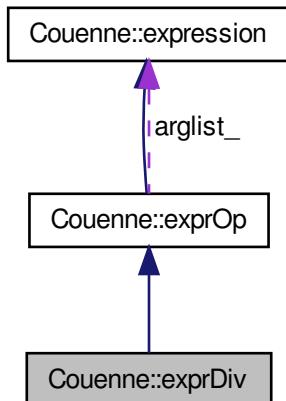
class for divisions, $\frac{f(x)}{g(x)}$

```
#include <CouenneExprDiv.hpp>
```

Inheritance diagram for Couenne::exprDiv:



Collaboration diagram for Couenne::exprDiv:



Public Member Functions

- [exprDiv \(expression **al, int n=2\)](#)
Constructor.
- [exprDiv \(expression *arg0, expression *arg1\)](#)
Constructor with two arguments given explicitly.
- [expression * clone \(Domain *d=NULL\) const](#)

- Cloning method.
- std::string **printOp () const**
Print operator.
- **CouNumber operator() ()**
Function for the evaluation of the expression.
- **CouNumber gradientNorm (const double *x)**
return l-2 norm of gradient at given point
- **expression * differentiate (int index)**
Differentiation.
- **expression * simplify ()**
Simplification.
- int **Linearity ()**
Get a measure of "how linear" the expression is (see CouenneTypes.h)
- void **getBounds (expression *&lb, expression *&ub)**
Get lower and upper bound of an expression (if any)
- void **getBounds (CouNumber &lb, CouNumber &ub)**
Get value of lower and upper bound of an expression (if any)
- **exprAux * standardize (CouenneProblem *p, bool addAux=true)**
Reduce expression in standard form, creating additional aux variables (and constraints)
- void **generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *t_chg_bounds=NULL, int=-1, CouNumber=-COUNNE_INFINITY, CouNumber=COUNNE_INFINITY)**
*Generate equality between *this and *w.*
- virtual enum **expr_type code ()**
Code for comparisons.
- bool **isInteger ()**
is this expression integer?
- bool **impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)**
Implied bound processing.
- virtual **CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)**
Set up branching object by evaluating many branching points for each expression's arguments.
- virtual void **closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right)**
const
compute $y^{\{l\}}$ and $y^{\{u\}}$ for Violation Transfer algorithm
- virtual bool **isCuttable (CouenneProblem *problem, int index)** const
can this expression be further linearized or are we on its concave ("bad") side

Additional Inherited Members

7.66.1 Detailed Description

class for divisions, $\frac{f(x)}{g(x)}$

Definition at line 24 of file CouenneExprDiv.hpp.

7.66.2 Constructor & Destructor Documentation

7.66.2.1 `Couenne::exprDiv::exprDiv (expression ** al, int n = 2) [inline]`

Constructor.

Definition at line 29 of file CouenneExprDiv.hpp.

7.66.2.2 `Couenne::exprDiv::exprDiv (expression * arg0, expression * arg1) [inline]`

Constructor with two arguments given explicitly.

Definition at line 33 of file CouenneExprDiv.hpp.

7.66.3 Member Function Documentation

7.66.3.1 `expression* Couenne::exprDiv::clone (Domain * d = NULL) const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::expression](#).

Definition at line 37 of file CouenneExprDiv.hpp.

7.66.3.2 `std::string Couenne::exprDiv::printOp () const [inline], [virtual]`

Print operator.

Reimplemented from [Couenne::exprOp](#).

Definition at line 41 of file CouenneExprDiv.hpp.

7.66.3.3 `CouNumber Couenne::exprDiv::operator() () [inline], [virtual]`

Function for the evaluation of the expression.

Compute division.

Implements [Couenne::expression](#).

Definition at line 115 of file CouenneExprDiv.hpp.

7.66.3.4 `CouNumber Couenne::exprDiv::gradientNorm (const double * x) [virtual]`

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

7.66.3.5 `expression* Couenne::exprDiv::differentiate (int index) [virtual]`

Differentiation.

Reimplemented from [Couenne::expression](#).

7.66.3.6 `expression* Couenne::exprDiv::simplify () [virtual]`

Simplification.

Reimplemented from [Couenne::exprOp](#).

7.66.3.7 int Couenne::exprDiv::Linearity() [inline], [virtual]

Get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::exprOp](#).

Definition at line 57 of file CouenneExprDiv.hpp.

7.66.3.8 void Couenne::exprDiv::getBounds(expression *& lb, expression *& ub) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.66.3.9 void Couenne::exprDiv::getBounds(CouNumber & lb, CouNumber & ub) [virtual]

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.66.3.10 exprAux* Couenne::exprDiv::standardize(CouenneProblem * p, bool addAux = true) [virtual]

Reduce expression in standard form, creating additional aux variables (and constraints)

Reimplemented from [Couenne::exprOp](#).

7.66.3.11 void Couenne::exprDiv::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

Generate equality between *this and *w.

Reimplemented from [Couenne::expression](#).

7.66.3.12 virtual enum expr_type Couenne::exprDiv::code() [inline], [virtual]

Code for comparisons.

Reimplemented from [Couenne::exprOp](#).

Definition at line 82 of file CouenneExprDiv.hpp.

7.66.3.13 bool Couenne::exprDiv::isInteger() [virtual]

is this expression integer?

Reimplemented from [Couenne::exprOp](#).

7.66.3.14 bool Couenne::exprDiv::impliedBound(int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [virtual]

Implied bound processing.

Reimplemented from [Couenne::expression](#).

7.66.3.15 virtual CouNumber Couenne::exprDiv::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]

Set up branching object by evaluating many branching points for each expression's arguments.

Reimplemented from [Couenne::expression](#).

7.66.3.16 virtual void Couenne::exprDiv::closestFeasible (**expression** * *varind*, **expression** * *vardep*, **CouNumber** & *left*, **CouNumber** & *right*) const [virtual]

compute $y^{\{v\}}$ and $y^{\{uv\}}$ for Violation Transfer algorithm

Reimplemented from [Couenne::expression](#).

7.66.3.17 virtual bool Couenne::exprDiv::isCuttable (**CouenneProblem** * *problem*, int *index*) const [inline], [virtual]

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

Definition at line 108 of file [CouenneExprDiv.hpp](#).

The documentation for this class was generated from the following file:

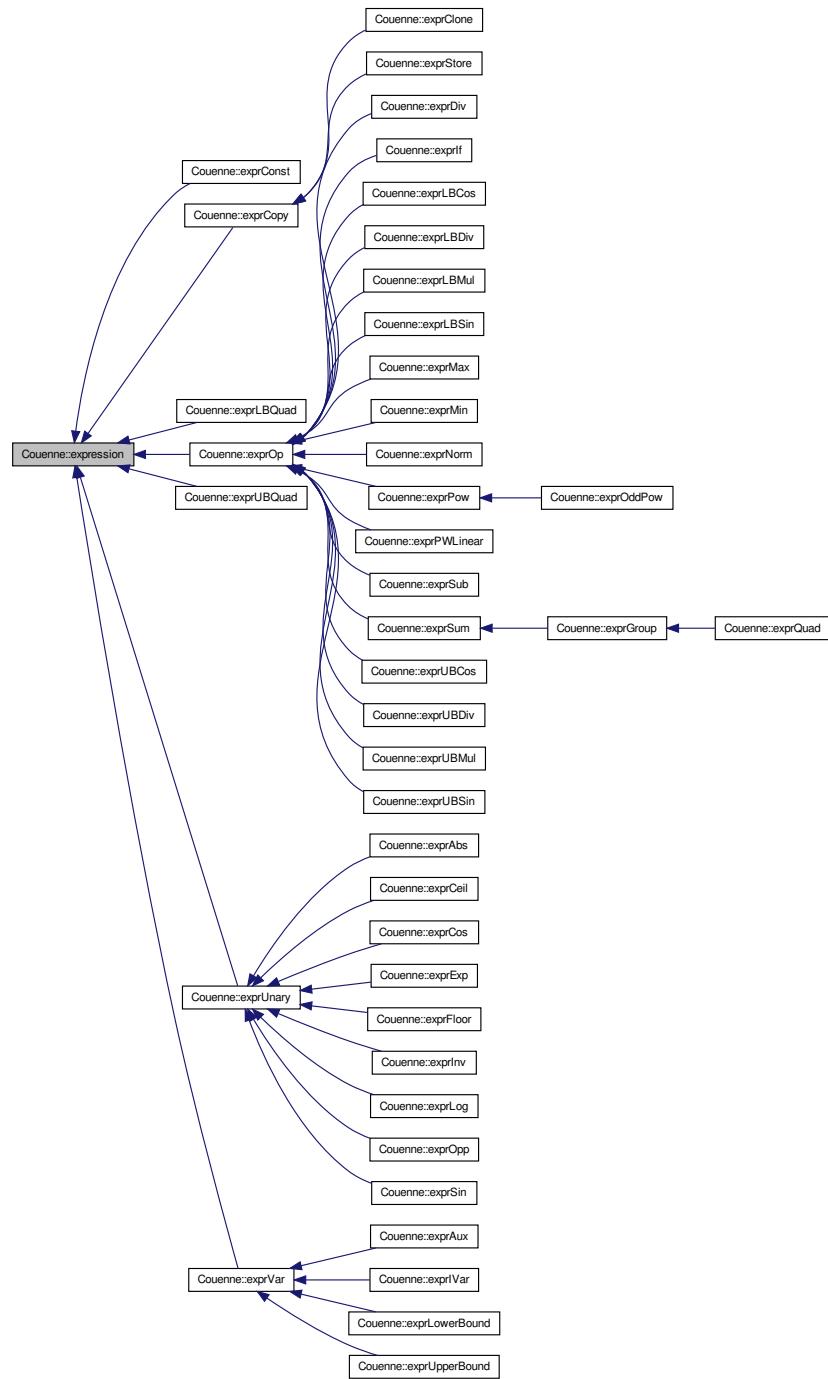
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprDiv.hpp](#)

7.67 Couenne::expression Class Reference

Expression base class.

```
#include <CouenneExpression.hpp>
```

Inheritance diagram for Couenne::expression:



Public Types

- enum `auxSign` { `AUX_UNDEF` = -2, `AUX_LEQ` = -1, `AUX_EQ`, `AUX_GEQ` }
"sign" of the constraint defining an auxiliary.

Public Member Functions

- **expression ()**
Constructor.
- **expression (const expression &e, Domain *d=NULL)**
Copy constructor.
- **virtual ~expression ()**
Destructor.
- **virtual expression * clone (Domain *d=NULL) const**
Cloning method.
- **virtual int Index () const**
Return index of variable (only valid for `exprVar` and `exprAux`)
- **virtual int nArgs () const**
return number of arguments (when applicable, that is, with N-ary functions)
- **virtual expression ** ArgList () const**
return arglist (when applicable, that is, with N-ary functions)
- **virtual void ArgList (expression **al)**
set arglist (used in deleting nodes without deleting children)
- **virtual expression * Argument () const**
return argument (when applicable, i.e., with univariate functions)
- **virtual expression ** ArgPtr ()**
return pointer to argument (when applicable, i.e., with univariate functions)
- **virtual enum nodeType Type () const**
node type
- **virtual expression * Image () const**
return pointer to corresponding expression (for auxiliary variables only)
- **virtual void Image (expression *image)**
set expression associated with this auxiliary variable (for compatibility with `exprAux`)
- **virtual CouNumber Value () const**
value (empty)
- **virtual const expression * Original () const**
If this is an `exprClone` of a `exprClone` of an `expr???`, point to the original `expr???` instead of an `exprClone` – improve computing efficiency.
- **virtual void print (std::ostream &s=std::cout, bool=false) const**
print expression to ostream
- **virtual CouNumber operator() ()=0**
null function for evaluating the expression
- **virtual CouNumber gradientNorm (const double *x)**
return l-2 norm of gradient at given point
- **virtual expression * differentiate (int)**
differentiation
- **virtual int dependsOn (int *ind, int n, enum dig_type type=STOP_AT_AUX)**
dependence on variable set: return cardinality of subset of the set of indices in first argument which occur in expression.
- **int dependsOn (int singleton, enum dig_type type=STOP_AT_AUX)**
version with one index only
- **virtual int DepList (std::set< int > &deplist, enum dig_type type=ORIG_ONLY)**
fill std::set with indices of variables on which this expression depends.
- **virtual expression * simplify ()**

- *simplify expression (useful for derivatives)*
- virtual int **Linearity** ()
 - get a measure of "how linear" the expression is (see CouenneTypes.h)*
- virtual bool **isDefinedInteger** ()
 - is this expression defined as an integer?*
- virtual bool **isInteger** ()
 - is this expression integer?*
- virtual void **getBounds** (**expression** *&, **expression** *&)
 - Get lower and upper bound of an expression (if any)*
- virtual void **getBounds** (**CouNumber** &, **CouNumber** &)
 - Get lower and upper bound of an expression (if any) – real values.*
- virtual **exprAux** * **standardize** (**CouenneProblem** *p, bool addAux=true)
 - Create standard form of this expression, by:*
- virtual void **generateCuts** (**expression** *w, OsiCuts &cs, const **CouenneCutGenerator** *cg, **t_chg_bounds** *chg=N-ULL, int wind=-1, **CouNumber** lb=-COUENNE_INFINITY, **CouNumber** ub=COUENNE_INFINITY)
 - generate convexification cut for constraint w = this*
- virtual enum **expr_type** **code** ()
 - return integer for comparing expressions (used to recognize common expression)*
- virtual enum **convexity** **convexity** () const
 - either CONVEX, CONCAVE, AFFINE, or NONCONVEX*
- virtual int **compare** (**expression** &)
 - compare expressions*
- virtual int **compare** (**exprCopy** &)
 - compare copies of expressions*
- virtual int **rank** ()
 - used in rank-based branching variable choice: original variables have rank 1; auxiliary w=f(x) has rank r(w) = r(x)+1; finally, auxiliary w=f(x1,x2...,xk) has rank r(w) = 1+max{r(xi):i=1..k}.*
- virtual bool **impliedBound** (int, **CouNumber** *, **CouNumber** *, **t_chg_bounds** *, enum **auxSign**=**expression**::**AUX_EQ**)
 - does a backward implied bound processing on every expression, including exprSums although already done by Clp (useful when repeated within Couenne).*
- virtual int **Multiplicity** ()
 - multiplicity of a variable*
- virtual **CouNumber** **selectBranch** (const **CouenneObject** *obj, const **OsiBranchingInformation** *info, **expression** *&var, double *&brpts, double *&brDist, int &way)
 - set up branching object by evaluating many branching points for each expression's arguments.*
- virtual void **replace** (**exprVar** *, **exprVar** *)
 - replace expression with another*
- virtual void **fillDepSet** (std::set< **DepNode** *, **compNode** > *, **DepGraph** *)
 - update dependence set with index of variables on which this expression depends*
- virtual void **linkDomain** (**Domain** *d)
 - empty function to update domain pointer*
- virtual void **realign** (const **CouenneProblem** *p)
 - empty function to redirect variables to proper variable vector*
- virtual bool **isBijective** () const
 - indicating if function is monotonically increasing*
- virtual **CouNumber** **inverse** (**expression** *vardep) const
 - compute the inverse function*

- virtual void `closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right)`
const
closest feasible points in function in both directions
- virtual bool `isCuttable (CouenneProblem *problem, int index)` const
can this expression be further linearized or are we on its concave ("bad") side
- virtual bool `isaCopy ()` const
return true if this is a copy of something (i.e. an `exprCopy`)
- virtual `expression * Copy ()` const
return copy of this expression (only makes sense in `exprCopy`)

7.67.1 Detailed Description

Expression base class.

An empty expression class with no type or operator() from which all other expression classes (for constants, variables, and operators) are derived.

Definition at line 48 of file CouenneExpression.hpp.

7.67.2 Member Enumeration Documentation

7.67.2.1 enum Couenne::expression::auxSign

"sign" of the constraint defining an auxiliary.

If the auxiliary is defined as $w \leq f(x)$, then it is LEQ. It is EQ and GEQ, respectively, if it is defined with = and \geq .

Enumerator:

```
AUX_UNDEF
AUX_LEQ
AUX_EQ
AUX_GEQ
```

Definition at line 55 of file CouenneExpression.hpp.

7.67.3 Constructor & Destructor Documentation

7.67.3.1 Couenne::expression::expression () [inline]

Constructor.

Definition at line 58 of file CouenneExpression.hpp.

7.67.3.2 Couenne::expression::expression (const expression & e, Domain * d=NULL) [inline]

Copy constructor.

Pass pointer to variable vector when generating new problem, whose set of variables is equivalent but may be changed or whose value is independent.

Definition at line 63 of file CouenneExpression.hpp.

7.67.3.3 virtual Couenne::expression::~expression() [inline], [virtual]

Destructor.

Definition at line 66 of file CouenneExpression.hpp.

7.67.4 Member Function Documentation**7.67.4.1 virtual expression* Couenne::expression::clone(Domain * d=NULL) const [inline], [virtual]**

Cloning method.

Reimplemented in Couenne::exprUpperBound, Couenne::exprUBMul, Couenne::exprQuad, Couenne::exprUBDiv, Couenne::exprUBCos, Couenne::exprUBSin, Couenne::exprAux, Couenne::exprUBQuad, Couenne::exprVar, Couenne::exprCopy, Couenne::exprGroup, Couenne::exprSin, Couenne::exprLowerBound, Couenne::exprStore, Couenne::exprConst, Couenne::exprLBMul, Couenne::exprPow, Couenne::exprLBDiv, Couenne::exprMin, Couenne::exprInv, Couenne::exprLBQuad, Couenne::exprLBCos, Couenne::exprLBSin, Couenne::exprClone, Couenne::exprIVar, Couenne::exprMax, Couenne::exprDiv, Couenne::exprOpp, Couenne::exprSum, Couenne::exprAbs, Couenne::exprOddPow, Couenne::exprSub, Couenne::exprExp, Couenne::exprLog, Couenne::exprCeil, Couenne::exprCos, and Couenne::exprFloor.

Definition at line 69 of file CouenneExpression.hpp.

7.67.4.2 virtual int Couenne::expression::Index() const [inline], [virtual]

Return index of variable (only valid for exprVar and exprAux)

Reimplemented in Couenne::exprCopy, and Couenne::exprVar.

Definition at line 73 of file CouenneExpression.hpp.

7.67.4.3 virtual int Couenne::expression::nArgs() const [inline], [virtual]

return number of arguments (when applicable, that is, with N-ary functions)

Reimplemented in Couenne::exprCopy, Couenne::exprOp, and Couenne::exprUnary.

Definition at line 77 of file CouenneExpression.hpp.

7.67.4.4 virtual expression Couenne::expression::ArgList() const [inline], [virtual]**

return arglist (when applicable, that is, with N-ary functions)

Reimplemented in Couenne::exprCopy, and Couenne::exprOp.

Definition at line 81 of file CouenneExpression.hpp.

7.67.4.5 virtual void Couenne::expression::ArgList(expression ** al) [inline], [virtual]

set arglist (used in deleting nodes without deleting children)

Reimplemented in Couenne::exprCopy, and Couenne::exprOp.

Definition at line 85 of file CouenneExpression.hpp.

7.67.4.6 virtual expression* Couenne::expression::Argument() const [inline], [virtual]

return argument (when applicable, i.e., with univariate functions)

Reimplemented in Couenne::exprCopy, and Couenne::exprUnary.

Definition at line 88 of file CouenneExpression.hpp.

7.67.4.7 virtual expression Couenne::expression::ArgPtr() [inline], [virtual]**

return pointer to argument (when applicable, i.e., with univariate functions)

Reimplemented in [Couenne::exprCopy](#), and [Couenne::exprUnary](#).

Definition at line 92 of file CouenneExpression.hpp.

7.67.4.8 virtual enum nodeType Couenne::expression::Type() const [inline], [virtual]

node type

Reimplemented in [Couenne::exprUpperBound](#), [Couenne::exprAux](#), [Couenne::exprVar](#), [Couenne::exprLowerBound](#), [Couenne::exprUnary](#), [Couenne::exprOp](#), [Couenne::exprCopy](#), and [Couenne::exprConst](#).

Definition at line 96 of file CouenneExpression.hpp.

7.67.4.9 virtual expression* Couenne::expression::Image() const [inline], [virtual]

return pointer to corresponding expression (for auxiliary variables only)

Reimplemented in [Couenne::exprAux](#), and [Couenne::exprCopy](#).

Definition at line 100 of file CouenneExpression.hpp.

7.67.4.10 virtual void Couenne::expression::Image(expression * image) [inline], [virtual]

set expression associated with this auxiliary variable (for compatibility with [exprAux](#))

Reimplemented in [Couenne::exprAux](#).

Definition at line 105 of file CouenneExpression.hpp.

7.67.4.11 virtual CouNumber Couenne::expression::Value() const [inline], [virtual]

value (empty)

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprClone](#), and [Couenne::exprConst](#).

Definition at line 108 of file CouenneExpression.hpp.

7.67.4.12 virtual const expression* Couenne::expression::Original() const [inline], [virtual]

If this is an [exprClone](#) of a [exprClone](#) of an [expr???](#), point to the original [expr???](#) instead of an [exprClone](#) – improve computing efficiency.

Only overloaded for [exprClones](#)/[exprCopy](#), of course.

Reimplemented in [Couenne::exprCopy](#).

Definition at line 114 of file CouenneExpression.hpp.

7.67.4.13 virtual void Couenne::expression::print(std::ostream & s = std::cout, bool = false) const [inline], [virtual]

print expression to ostream

descend into auxiliaries' image?

Reimplemented in [Couenne::exprUpperBound](#), [Couenne::exprCopy](#), [Couenne::exprQuad](#), [Couenne::exprAux](#), [Couenne::exprUBQuad](#), [Couenne::exprVar](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), [Couenne::exprGroup](#), [Couenne::exprLowerBound](#), [Couenne::exprClone](#), [Couenne::exprConst](#), [Couenne::exprLBQuad](#), [Couenne::exprInv](#), [Couenne::exprStore](#), [Couenne::exprOpp](#), and [Couenne::exprVar](#).

Definition at line 118 of file CouenneExpression.hpp.

7.67.4.14 virtual CouNumber Couenne::expression::operator() () [pure virtual]

null function for evaluating the expression

Implemented in [Couenne::exprCopy](#), [Couenne::exprUpperBound](#), [Couenne::exprAux](#), [Couenne::exprQuad](#), [Couenne::exprUBMul](#), [Couenne::exprUBDiv](#), [Couenne::exprUBCos](#), [Couenne::exprUBSin](#), [Couenne::exprVar](#), [Couenne::exprUBQuad](#), [Couenne::exprUnary](#), [Couenne::exprGroup](#), [Couenne::exprLowerBound](#), [Couenne::exprClone](#), [Couenne::exprConst](#), [Couenne::exprStore](#), [Couenne::exprMin](#), [Couenne::exprPow](#), [Couenne::exprLBMul](#), [Couenne::exprLBDiv](#), [Couenne::exprMax](#), [Couenne::exprLBQuad](#), [Couenne::exprDiv](#), [Couenne::exprLBCos](#), [Couenne::exprLBSin](#), [Couenne::exprSum](#), [Couenne::exprOddPow](#), and [Couenne::exprSub](#).

7.67.4.15 virtual CouNumber Couenne::expression::gradientNorm (const double * x) [inline], [virtual]

return l-2 norm of gradient at given point

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprQuad](#), [Couenne::exprVar](#), [Couenne::exprGroup](#), [Couenne::exprSin](#), [Couenne::exprPow](#), [Couenne::exprInv](#), [Couenne::exprOpp](#), [Couenne::exprDiv](#), [Couenne::exprAbs](#), [Couenne::exprExp](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), and [Couenne::exprLog](#).

Definition at line 126 of file CouenneExpression.hpp.

7.67.4.16 virtual expression* Couenne::expression::differentiate (int) [virtual]

differentiation

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprUpperBound](#), [Couenne::exprQuad](#), [Couenne::exprVar](#), [Couenne::exprGroup](#), [Couenne::exprSin](#), [Couenne::exprLowerBound](#), [Couenne::exprConst](#), [Couenne::exprPow](#), [Couenne::exprMin](#), [Couenne::exprInv](#), [Couenne::exprMax](#), [Couenne::exprOpp](#), [Couenne::exprDiv](#), [Couenne::exprAbs](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), [Couenne::exprSum](#), [Couenne::exprExp](#), [Couenne::exprSub](#), and [Couenne::exprLog](#).

7.67.4.17 virtual int Couenne::expression::dependsOn (int * ind, int n, enum dig_type type = STOP_AT_AUX) [virtual]

dependence on variable set: return cardinality of subset of the set of indices in first argument which occur in expression.

Reimplemented in [Couenne::exprUpperBound](#), [Couenne::exprLowerBound](#), and [Couenne::exprConst](#).

7.67.4.18 int Couenne::expression::dependsOn (int singleton, enum dig_type type = STOP_AT_AUX) [inline]

version with one index only

Definition at line 137 of file CouenneExpression.hpp.

7.67.4.19 virtual int Couenne::expression::DepList (std::set< int > & depList, enum dig_type type = ORIG_ONLY) [inline], [virtual]

fill std::set with indices of variables on which this expression depends.

Also deal with expressions that have no variable pointers ([exprGroup](#), [exprQuad](#))

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprAux](#), [Couenne::exprVar](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), and [Couenne::exprGroup](#).

Definition at line 143 of file CouenneExpression.hpp.

7.67.4.20 virtual expression* Couenne::expression::simplify () [inline], [virtual]

simplify expression (useful for derivatives)

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprVar](#), [Couenne::exprAux](#), [Couenne::exprQuad](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), [Couenne::exprGroup](#), [Couenne::exprMin](#), [Couenne::exprPow](#), [Couenne::exprMax](#), [Couenne::exprOpp](#), [Couenne::exprDiv](#), [Couenne::exprSum](#), and [Couenne::exprSub](#).

Definition at line 148 of file CouenneExpression.hpp.

7.67.4.21 virtual int Couenne::expression::Linearity() [inline], [virtual]

get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprUpperBound](#), [Couenne::exprVar](#), [Couenne::exprAux](#), [Couenne::exprQuad](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), [Couenne::exprGroup](#), [Couenne::exprLowerBound](#), [Couenne::exprConst](#), [Couenne::exprMin](#), [Couenne::exprPow](#), [Couenne::exprMax](#), [Couenne::exprInv](#), [Couenne::exprOpp](#), [Couenne::exprDiv](#), [Couenne::exprSum](#), and [Couenne::exprSub](#).

Definition at line 152 of file CouenneExpression.hpp.

7.67.4.22 virtual bool Couenne::expression::isDefinedInteger() [inline], [virtual]

is this expression defined as an integer?

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprAux](#), [Couenne::exprVar](#), and [Couenne::exprInv](#).

Definition at line 156 of file CouenneExpression.hpp.

7.67.4.23 virtual bool Couenne::expression::isInteger() [inline], [virtual]

is this expression integer?

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprAux](#), [Couenne::exprCopy](#), [Couenne::exprVar](#), [Couenne::exprOp](#), [Couenne::exprGroup](#), [Couenne::exprUnary](#), [Couenne::exprConst](#), [Couenne::exprDiv](#), [Couenne::exprOpp](#), [Couenne::exprPow](#), [Couenne::exprAbs](#), and [Couenne::exprInv](#).

Definition at line 160 of file CouenneExpression.hpp.

7.67.4.24 virtual void Couenne::expression::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprVar](#), [Couenne::exprQuad](#), [Couenne::exprGroup](#), [Couenne::exprSin](#), [Couenne::exprConst](#), [Couenne::exprPow](#), [Couenne::exprMin](#), [Couenne::exprInv](#), [Couenne::exprDiv](#), [Couenne::exprMax](#), [Couenne::exprOpp](#), [Couenne::exprSub](#), [Couenne::exprSum](#), [Couenne::exprAbs](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), [Couenne::exprExp](#), [Couenne::exprLog](#), and [Couenne::exprOddPow](#).

7.67.4.25 virtual void Couenne::expression::getBounds(CouNumber &, CouNumber &) [virtual]

Get lower and upper bound of an expression (if any) – real values.

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprVar](#), [Couenne::exprQuad](#), [Couenne::exprGroup](#), [Couenne::exprConst](#), [Couenne::exprSin](#), [Couenne::exprPow](#), [Couenne::exprInv](#), [Couenne::exprDiv](#), [Couenne::exprOpp](#), [Couenne::exprSub](#), [Couenne::exprSum](#), [Couenne::exprAbs](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), [Couenne::exprExp](#), [Couenne::exprLog](#), and [Couenne::exprOddPow](#).

7.67.4.26 virtual exprAux* Couenne::expression::standardize(CouenneProblem * p, bool addAux = true) [inline], [virtual]

Create standard form of this expression, by:

- creating auxiliary w variables and corresponding expressions
- returning linear counterpart as new constraint (to replace current one)

For the base [exprOp](#) class we only do the first part (for argument list components only), and the calling class (Sum, Sub, Mul, Pow, and the like) will do the part for its own object

addAux is true if a new auxiliary variable should be added associated with the standardized expression

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), [Couenne::exprOpp](#), [Couenne::exprPow](#), [Couenne::exprMin](#), [Couenne::exprDiv](#), [Couenne::exprMax](#), [Couenne::exprSub](#), [Couenne::exprSum](#), and [Couenne::exprOddPow](#).

Definition at line 181 of file CouenneExpression.hpp.

7.67.4.27 virtual void Couenne::expression::generateCuts (expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * chg = NULL, int wind = -1, CouNumber lb = -COUENNE_INFINITY, CouNumber ub = COUENNE_INFINITY) [inline], [virtual]

generate convexification cut for constraint w = this

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprVar](#), [Couenne::exprQuad](#), [Couenne::exprGroup](#), [Couenne::exprConst](#), [Couenne::exprPow](#), [Couenne::exprSin](#), [Couenne::exprMin](#), [Couenne::exprDiv](#), [Couenne::exprMax](#), [Couenne::exprInv](#), [Couenne::exprSub](#), [Couenne::exprOpp](#), [Couenne::exprSum](#), [Couenne::exprAbs](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), [Couenne::exprOddPow](#), [Couenne::exprExp](#), and [Couenne::exprLog](#).

Definition at line 185 of file CouenneExpression.hpp.

7.67.4.28 virtual enum expr_type Couenne::expression::code() [inline], [virtual]

return integer for comparing expressions (used to recognize common expression)

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprVar](#), [Couenne::exprUpperBound](#), [Couenne::exprOp](#), [Couenne::exprGroup](#), [Couenne::exprUnary](#), [Couenne::exprPow](#), [Couenne::exprConst](#), [Couenne::exprSin](#), [Couenne::exprMin](#), [Couenne::exprDiv](#), [Couenne::exprLowerBound](#), [Couenne::exprMax](#), [Couenne::exprInv](#), [Couenne::exprSub](#), [Couenne::exprOpp](#), [Couenne::exprSum](#), [Couenne::exprOddPow](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), [Couenne::exprAbs](#), [Couenne::exprExp](#), and [Couenne::exprLog](#).

Definition at line 193 of file CouenneExpression.hpp.

7.67.4.29 virtual enum convexity Couenne::expression::convexity() const [inline], [virtual]

either CONVEX, CONCAVE, AFFINE, or NONCONVEX

Reimplemented in [Couenne::exprVar](#), and [Couenne::exprCopy](#).

Definition at line 197 of file CouenneExpression.hpp.

7.67.4.30 virtual int Couenne::expression::compare (expression &) [virtual]

compare expressions

Reimplemented in [Couenne::exprCopy](#).

7.67.4.31 virtual int Couenne::expression::compare (exprCopy &) [virtual]

compare copies of expressions

7.67.4.32 virtual int Couenne::expression::rank () [inline], [virtual]

used in rank-based branching variable choice: original variables have rank 1; auxiliary w=f(x) has rank r(w) = r(x)+1; finally, auxiliary w=f(x1,x2...,xk) has rank r(w) = 1+max{r(xi):i=1..k}.

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprVar](#), [Couenne::exprAux](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), [Couenne::exprGroup](#), and [Couenne::exprConst](#).

Definition at line 209 of file CouenneExpression.hpp.

7.67.4.33 virtual bool Couenne::expression::impliedBound(int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [inline], [virtual]

does a backward implied bound processing on every expression, including exprSums although already done by Clp (useful when repeated within [Couenne](#)).

Parameters are the index of the (auxiliary) variable in question and the current lower/upper bound. The method returns true if there has been a change on any bound on the variables on which the expression depends.

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprVar](#), [Couenne::exprPow](#), [Couenne::exprSum](#), [Couenne::exprSin](#), [Couenne::exprDiv](#), [Couenne::exprOpp](#), [Couenne::exprInv](#), [Couenne::exprSub](#), [Couenne::exprOddPow](#), [Couenne::exprAbs](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), [Couenne::exprExp](#), and [Couenne::exprLog](#).

Definition at line 218 of file CouenneExpression.hpp.

7.67.4.34 virtual int Couenne::expression::Multiplicity() [inline], [virtual]

multiplicity of a variable

Reimplemented in [Couenne::exprCopy](#), and [Couenne::exprAux](#).

Definition at line 222 of file CouenneExpression.hpp.

7.67.4.35 virtual CouNumber Couenne::expression::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [inline], [virtual]

set up branching object by evaluating many branching points for each expression's arguments.

Return estimated improvement in objective function

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprSin](#), [Couenne::exprPow](#), [Couenne::exprDiv](#), [Couenne::exprInv](#), [Couenne::exprCos](#), [Couenne::exprCeil](#), [Couenne::exprFloor](#), [Couenne::exprOddPow](#), [Couenne::exprAbs](#), [Couenne::exprExp](#), and [Couenne::exprLog](#).

Definition at line 228 of file CouenneExpression.hpp.

7.67.4.36 virtual void Couenne::expression::replace(exprVar *, exprVar *) [inline], [virtual]

replace expression with another

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), and [Couenne::exprGroup](#).

Definition at line 238 of file CouenneExpression.hpp.

7.67.4.37 virtual void Couenne::expression::fillDepSet(std::set< DepNode *, compNode > * , DepGraph *) [inline], [virtual]

update dependence set with index of variables on which this expression depends

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprVar](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), and [Couenne::exprGroup](#).

Definition at line 242 of file CouenneExpression.hpp.

7.67.4.38 virtual void Couenne::expression::linkDomain(Domain * d) [inline], [virtual]

empty function to update domain pointer

Reimplemented in [Couenne::exprVar](#), and [Couenne::exprAux](#).

Definition at line 245 of file CouenneExpression.hpp.

7.67.4.39 virtual void Couenne::expression::realign (const CouenneProblem * *p*) [inline], [virtual]

empty function to redirect variables to proper variable vector

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprOp](#), [Couenne::exprUnary](#), and [Couenne::exprGroup](#).

Definition at line 248 of file CouenneExpression.hpp.

7.67.4.40 virtual bool Couenne::expression::isBijective () const [inline], [virtual]

indicating if function is monotonically increasing

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprInv](#), [Couenne::exprExp](#), and [Couenne::exprLog](#).

Definition at line 251 of file CouenneExpression.hpp.

7.67.4.41 virtual CouNumber Couenne::expression::inverse (expression * *vardep*) const [inline], [virtual]

compute the inverse function

Reimplemented in [Couenne::exprCopy](#), [Couenne::exprInv](#), [Couenne::exprExp](#), and [Couenne::exprLog](#).

Definition at line 255 of file CouenneExpression.hpp.

7.67.4.42 virtual void Couenne::expression::closestFeasible (expression * *varind*, expression * *vardep*, CouNumber & *left*, CouNumber & *right*) const [virtual]

closest feasible points in function in both directions

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprSin](#), [Couenne::exprPow](#), [Couenne::exprDiv](#), [Couenne::exprCos](#), [Couenne::exprCeil](#), [Couenne::exprFloor](#), and [Couenne::exprAbs](#).

7.67.4.43 virtual bool Couenne::expression::isCuttable (CouenneProblem * *problem*, int *index*) const [inline], [virtual]

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprCopy](#), [Couenne::exprSin](#), [Couenne::exprPow](#), [Couenne::exprDiv](#), [Couenne::exprInv](#), [Couenne::exprCos](#), [Couenne::exprCeil](#), [Couenne::exprFloor](#), [Couenne::exprAbs](#), [Couenne::exprExp](#), [Couenne::exprLog](#), and [Couenne::exprOddPow](#).

Definition at line 264 of file CouenneExpression.hpp.

7.67.4.44 virtual bool Couenne::expression::isaCopy () const [inline], [virtual]

return true if this is a copy of something (i.e. an [exprCopy](#))

Reimplemented in [Couenne::exprCopy](#).

Definition at line 268 of file CouenneExpression.hpp.

7.67.4.45 virtual expression* Couenne::expression::Copy () const [inline], [virtual]

return copy of this expression (only makes sense in [exprCopy](#))

Reimplemented in [Couenne::exprCopy](#).

Definition at line 272 of file CouenneExpression.hpp.

The documentation for this class was generated from the following file:

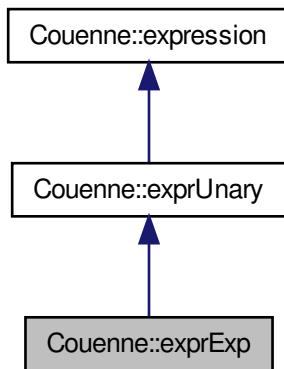
- /home/ted/COIN/trunk/Couenne/src/expression/CouenneExpression.hpp

7.68 Couenne::exprExp Class Reference

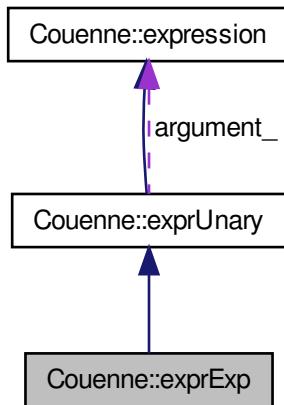
class for the exponential, $e^{f(x)}$

```
#include <CouenneExprExp.hpp>
```

Inheritance diagram for Couenne::exprExp:



Collaboration diagram for Couenne::exprExp:



Public Member Functions

- **exprExp (expression *al)**
Constructor.
- **expression * clone (Domain *d=NULL) const**
Cloning method.
- **unary_function F ()**
The operator's function.
- **std::string printOp () const**
Print operator.
- **CouNumber gradientNorm (const double *x)**
return l-2 norm of gradient at given point
- **expression * differentiate (int index)**
Differentiation.
- **void getBounds (expression *&, expression *&)**
Get lower and upper bound of an expression (if any)
- **virtual void getBounds (CouNumber &lb, CouNumber &ub)**
Get expression of lower and upper bound of an expression (if any)
- **void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)**
Generate convexification cuts for this expression.
- **virtual enum expr_type code ()**
Code for comparisons.
- **bool impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)**
Implied bound processing.
- **virtual CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)**
Set up branching object by evaluating many branching points for each expression's arguments.
- **virtual bool isBijective () const**
return true if bijective
- **virtual CouNumber inverse (expression *vardep) const**
inverse of exponential
- **virtual bool isCuttable (CouenneProblem *problem, int index) const**
can this expression be further linearized or are we on its concave ("bad") side

Additional Inherited Members

7.68.1 Detailed Description

class for the exponential, $e^{f(x)}$

Definition at line 22 of file CouenneExprExp.hpp.

7.68.2 Constructor & Destructor Documentation

7.68.2.1 Couenne::exprExp::exprExp (expression * al) [inline]

Constructor.

Definition at line 27 of file CouenneExprExp.hpp.

7.68.3 Member Function Documentation

7.68.3.1 `expression* Couenne::exprExp::clone(Domain * d = NULL) const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::expression](#).

Definition at line 31 of file CouenneExprExp.hpp.

7.68.3.2 `unary_function Couenne::exprExp::F() [inline], [virtual]`

The operator's function.

Reimplemented from [Couenne::exprUnary](#).

Definition at line 35 of file CouenneExprExp.hpp.

7.68.3.3 `std::string Couenne::exprExp::printOp() const [inline], [virtual]`

Print operator.

Reimplemented from [Couenne::exprUnary](#).

Definition at line 38 of file CouenneExprExp.hpp.

7.68.3.4 `CouNumber Couenne::exprExp::gradientNorm(const double * x) [inline], [virtual]`

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Definition at line 42 of file CouenneExprExp.hpp.

7.68.3.5 `expression* Couenne::exprExp::differentiate(int index) [virtual]`

Differentiation.

Reimplemented from [Couenne::expression](#).

7.68.3.6 `void Couenne::exprExp::getBounds(expression *&, expression *&) [virtual]`

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.68.3.7 `virtual void Couenne::exprExp::getBounds(CouNumber & lb, CouNumber & ub) [virtual]`

Get expression of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.68.3.8 `void Couenne::exprExp::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]`

Generate convexification cuts for this expression.

Reimplemented from [Couenne::expression](#).

7.68.3.9 `virtual enum expr_type Couenne::exprExp::code() [inline], [virtual]`

Code for comparisons.

Reimplemented from [Couenne::exprUnary](#).

Definition at line 62 of file [CouenneExprExp.hpp](#).

7.68.3.10 `bool Couenne::exprExp::impliedBound(int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [virtual]`

Implied bound processing.

Reimplemented from [Couenne::expression](#).

7.68.3.11 `virtual CouNumber Couenne::exprExp::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]`

Set up branching object by evaluating many branching points for each expression's arguments.

Reimplemented from [Couenne::expression](#).

7.68.3.12 `virtual bool Couenne::exprExp::isBijective() const [inline], [virtual]`

return true if bijective

Reimplemented from [Couenne::expression](#).

Definition at line 78 of file [CouenneExprExp.hpp](#).

7.68.3.13 `virtual CouNumber Couenne::exprExp::inverse(expression * vardep) const [inline], [virtual]`

inverse of exponential

Reimplemented from [Couenne::expression](#).

Definition at line 81 of file [CouenneExprExp.hpp](#).

7.68.3.14 `virtual bool Couenne::exprExp::isCuttable(CouenneProblem * problem, int index) const [virtual]`

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

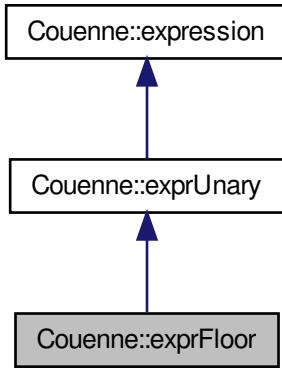
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprExp.hpp](#)

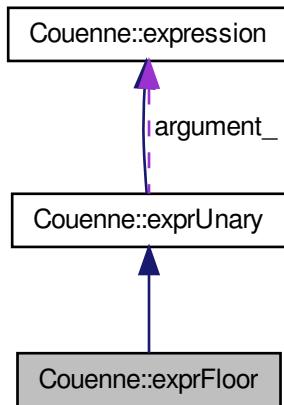
7.69 Couenne::exprFloor Class Reference

```
class floor,  $\lfloor f(x) \rfloor$ 
#include <CouenneExprFloor.hpp>
```

Inheritance diagram for Couenne::exprFloor:



Collaboration diagram for Couenne::exprFloor:



Public Member Functions

- [exprFloor \(expression *arg\)](#)
constructor, destructor
- [expression * clone \(Domain *d=NULL\) const](#)
cloning method
- [unary_function F \(\)](#)

- `std::string printOp () const`
print operator
- `CouNumber gradientNorm (const double *x)`
return L2 norm of gradient at given point
- `expression * differentiate (int index)`
obtain derivative of expression
- `void getBounds (expression *&, expression *&)`
Get lower and upper bound of an expression (if any)
- `void getBounds (CouNumber &lb, CouNumber &ub)`
Get value of lower and upper bound of an expression.
- `void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *chgs = NULL, int = -1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
*generate equality between *this and *w*
- `virtual enum expr_type code ()`
code for comparisons
- `bool impliedBound (int index, CouNumber *l, CouNumber *u, t_chg_bounds *chgs, enum auxSign=expression::AUX_EQ)`
implied bound processing
- `virtual CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
Set up branching object by evaluating many branching points for each expression's arguments.
- `virtual void closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right) const`
closest feasible points in function in both directions
- `virtual bool isCuttable (CouenneProblem *problem, int index) const`
can this expression be further linearized or are we on its concave ("bad") side?

Additional Inherited Members

7.69.1 Detailed Description

class floor, $\lfloor f(x) \rfloor$

Definition at line 20 of file CouenneExprFloor.hpp.

7.69.2 Constructor & Destructor Documentation

7.69.2.1 Couenne::exprFloor::exprFloor (expression * arg) [inline]

constructor, destructor

Definition at line 25 of file CouenneExprFloor.hpp.

7.69.3 Member Function Documentation

7.69.3.1 expression* Couenne::exprFloor::clone (Domain * d = NULL) const [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 29 of file CouenneExprFloor.hpp.

7.69.3.2 unary_function Couenne::exprFloor::F() [inline], [virtual]

the operator itself (e.g. sin, log...)

Reimplemented from [Couenne::exprUnary](#).

Definition at line 33 of file CouenneExprFloor.hpp.

7.69.3.3 std::string Couenne::exprFloor::printOp() const [inline], [virtual]

print operator

Reimplemented from [Couenne::exprUnary](#).

Definition at line 37 of file CouenneExprFloor.hpp.

7.69.3.4 CouNumber Couenne::exprFloor::gradientNorm(const double *x) [inline], [virtual]

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Definition at line 41 of file CouenneExprFloor.hpp.

7.69.3.5 expression* Couenne::exprFloor::differentiate(int index) [virtual]

obtain derivative of expression

Reimplemented from [Couenne::expression](#).

7.69.3.6 void Couenne::exprFloor::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.69.3.7 void Couenne::exprFloor::getBounds(CouNumber & lb, CouNumber & ub) [virtual]

Get value of lower and upper bound of an expression.

Reimplemented from [Couenne::expression](#).

7.69.3.8 void Couenne::exprFloor::generateCuts(expression *w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

7.69.3.9 virtual enum expr_type Couenne::exprFloor::code() [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::exprUnary](#).

Definition at line 63 of file CouenneExprFloor.hpp.

7.69.3.10 `bool Couenne::exprFloor::impliedBound(int index, CouNumber * l, CouNumber * u, t_chg_bounds * chg, enum auxSign = expression::AUX_EQ) [inline], [virtual]`

implied bound processing

Reimplemented from [Couenne::expression](#).

Definition at line 67 of file CouenneExprFloor.hpp.

7.69.3.11 `virtual CouNumber Couenne::exprFloor::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [inline], [virtual]`

Set up branching object by evaluating many branching points for each expression's arguments.

Reimplemented from [Couenne::expression](#).

Definition at line 75 of file CouenneExprFloor.hpp.

7.69.3.12 `virtual void Couenne::exprFloor::closestFeasible(expression * varind, expression * vardep, CouNumber & left, CouNumber & right) const [virtual]`

closest feasible points in function in both directions

Reimplemented from [Couenne::expression](#).

7.69.3.13 `virtual bool Couenne::exprFloor::isCuttable(CouenneProblem * problem, int index) const [inline], [virtual]`

can this expression be further linearized or are we on its concave ("bad") side?

Reimplemented from [Couenne::expression](#).

Definition at line 90 of file CouenneExprFloor.hpp.

The documentation for this class was generated from the following file:

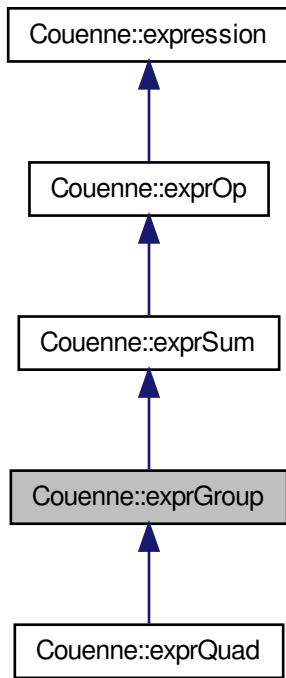
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprFloor.hpp](#)

7.70 Couenne::exprGroup Class Reference

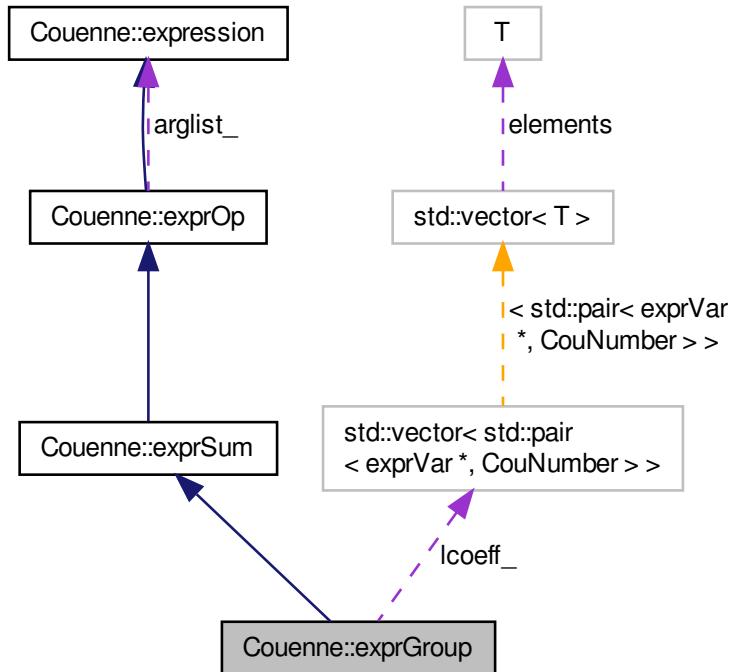
class Group, with constant, linear and nonlinear terms: $a_0 + \sum_{i=1}^n a_i x_i$

```
#include <CouenneExprGroup.hpp>
```

Inheritance diagram for Couenne::exprGroup:



Collaboration diagram for Couenne::exprGroup:



Public Types

- `typedef std::vector< std::pair< exprVar *, CouNumber > > lincoeff`

Public Member Functions

- `exprGroup (CouNumber, lincoeff &, expression ***=NULL, int=0)`
Constructor.
- `exprGroup (const exprGroup &src, Domain *d=NULL)`
Copy constructor.
- `virtual ~exprGroup ()`
Destructor – needed to clear bounds.
- `virtual expression * clone (Domain *d=NULL) const`
Cloning method.
- `CouNumber getc0 ()`
return constant term
- `lincoeff & lcoeff () const`
return linear term coefficients

- virtual void `print` (std::ostream &=std::cout, bool=false) const
Print expression to ostream.
- virtual `CouNumber operator()` ()
function for the evaluation of the expression
- virtual `CouNumber gradientNorm` (const double *x)
return L2 norm of gradient at given point
- virtual int `DepList` (std::set< int > &depList, enum `dig_type` type=`ORIG_ONLY`)
fill in the set with all indices of variables appearing in the expression
- virtual `expression * differentiate` (int index)
differentiation
- virtual `expression * simplify` ()
simplification
- virtual int `Linearity` ()
get a measure of "how linear" the expression is:
- virtual void `getBounds` (`expression *&`, `expression *&`)
Get lower and upper bound of an expression (if any)
- virtual void `getBounds` (`CouNumber &`, `CouNumber &`)
Get lower and upper bound of an expression (if any)
- virtual void `generateCuts` (`expression *`, `OsiCuts &`, const `CouenneCutGenerator *`, `t_chg_bounds * =NULL`, int=-1, `CouNumber=-COUENNE_INFINITY`, `CouNumber=COUENNE_INFINITY`)
special version for linear constraints
- virtual int `compare` (`exprGroup &`)
only compare with people of the same kind
- virtual enum `expr_type code` ()
code for comparisons
- virtual bool `isInteger` ()
is this expression integer?
- virtual int `rank` ()
used in rank-based branching variable choice
- virtual void `fillDepSet` (std::set< `DepNode *`, `compNode > *>`, `DepGraph *`)
update dependence set with index of this variable
- virtual void `replace` (`exprVar *x`, `exprVar *w`)
replace variable x with new (aux) w
- virtual void `realign` (const `CouenneProblem *p`)
redirect variables to proper variable vector

Static Public Member Functions

- static `expression * genExprGroup` (`CouNumber lincoeff &`, `expression ** =NULL`, int=0)
Generalized (static) constructor: check parameters and return a constant, a single variable, or a real exprGroup.

Protected Attributes

- `lincoeff lcoeff_`
coefficients and indices of the linear term
- `CouNumber c0_`
constant term

Additional Inherited Members

7.70.1 Detailed Description

class Group, with constant, linear and nonlinear terms: $a_0 + \sum_{i=1}^n a_i x_i$

Definition at line 25 of file CouenneExprGroup.hpp.

7.70.2 Member Typedef Documentation

7.70.2.1 `typedef std::vector<std::pair <exprVar *, CouNumber>> Couenne::exprGroup::lincoeff`

Definition at line 29 of file CouenneExprGroup.hpp.

7.70.3 Constructor & Destructor Documentation

7.70.3.1 `Couenne::exprGroup::exprGroup (CouNumber , lincoeff & , expression ** =NULL, int =0)`

Constructor.

7.70.3.2 `Couenne::exprGroup::exprGroup (const exprGroup & src, Domain * d=NULL)`

Copy constructor.

7.70.3.3 `virtual Couenne::exprGroup::~exprGroup() [virtual]`

Destructor – needed to clear bounds.

7.70.4 Member Function Documentation

7.70.4.1 `static expression* Couenne::exprGroup::genExprGroup (CouNumber , lincoeff & , expression ** =NULL, int =0) [static]`

Generalized (static) constructor: check parameters and return a constant, a single variable, or a real [exprGroup](#).

7.70.4.2 `virtual expression* Couenne::exprGroup::clone (Domain * d=NULL) const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

Definition at line 58 of file CouenneExprGroup.hpp.

7.70.4.3 `CouNumber Couenne::exprGroup::getc0 () [inline]`

return constant term

Definition at line 62 of file CouenneExprGroup.hpp.

7.70.4.4 `lincoeff& Couenne::exprGroup::lcoeff () const [inline]`

return linear term coefficients

Definition at line 63 of file CouenneExprGroup.hpp.

7.70.4.5 `virtual void Couenne::exprGroup::print(std::ostream & = std::cout, bool = false) const [virtual]`

Print expression to ostream.

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.6 `CouNumber Couenne::exprGroup::operator()() [inline], [virtual]`

function for the evaluation of the expression

compute sum of linear and nonlinear terms

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

Definition at line 127 of file CouenneExprGroup.hpp.

7.70.4.7 `virtual CouNumber Couenne::exprGroup::gradientNorm(const double *x) [virtual]`

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.8 `virtual int Couenne::exprGroup::DepList(std::set<int> & depList, enum dig_type type = ORIG_ONLY) [virtual]`

fill in the set with all indices of variables appearing in the expression

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.9 `virtual expression* Couenne::exprGroup::differentiate(int index) [virtual]`

differentiation

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.10 `virtual expression* Couenne::exprGroup::simplify() [virtual]`

simplification

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.11 `virtual int Couenne::exprGroup::Linearity() [virtual]`

get a measure of "how linear" the expression is:

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.12 `virtual void Couenne::exprGroup::getBounds(expression *&, expression *&) [virtual]`

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.13 virtual void Couenne::exprGroup::getBounds (CouNumber & , CouNumber &) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.14 virtual void Couenne::exprGroup::generateCuts (expression * , OsiCuts & , const CouenneCutGenerator * , t_chg_bounds * =NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

special version for linear constraints

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.15 virtual int Couenne::exprGroup::compare (exprGroup &) [virtual]

only compare with people of the same kind

7.70.4.16 virtual enum expr_type Couenne::exprGroup::code () [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::exprSum](#).

Reimplemented in [Couenne::exprQuad](#).

Definition at line 106 of file CouenneExprGroup.hpp.

7.70.4.17 virtual bool Couenne::exprGroup::isInteger () [virtual]

is this expression integer?

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.18 virtual int Couenne::exprGroup::rank () [virtual]

used in rank-based branching variable choice

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.19 virtual void Couenne::exprGroup::fillDepSet (std::set< DepNode *, compNode > * , DepGraph *) [virtual]

update dependence set with index of this variable

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.20 virtual void Couenne::exprGroup::replace (exprVar * x, exprVar * w) [virtual]

replace variable x with new (aux) w

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.4.21 virtual void Couenne::exprGroup::realign (const CouenneProblem * p) [virtual]

redirect variables to proper variable vector

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#).

7.70.5 Member Data Documentation

7.70.5.1 lincoeff Couenne::exprGroup::lcoeff_ [mutable], [protected]

coefficients and indices of the linear term

Definition at line 33 of file [CouenneExprGroup.hpp](#).

7.70.5.2 CouNumber Couenne::exprGroup::c0_ [protected]

constant term

Definition at line 34 of file [CouenneExprGroup.hpp](#).

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprGroup.hpp](#)

7.71 Couenne::ExprHess Class Reference

expression matrices.

```
#include <CouenneExprHess.hpp>
```

Public Member Functions

- [ExprHess \(\)](#)
- [ExprHess \(CouenneProblem *\)](#)
- [ExprHess \(const ExprHess &\)](#)
- [ExprHess & operator= \(const ExprHess &\)](#)
- [ExprHess * clone \(\)](#)
- [~ExprHess \(\)](#)
- int [nnz \(\)](#)
- int * [iRow \(\)](#)
- int * [jCol \(\)](#)
- int * [numL \(\)](#)
- int ** [lamL \(\)](#)
- [expression *** expr \(\)](#)

7.71.1 Detailed Description

expression matrices.

Used to evaluate the Hessian of the Lagrangian function at an optimal solution of the NLP
Definition at line 21 of file CouenneExprHess.hpp.

7.71.2 Constructor & Destructor Documentation

7.71.2.1 `Couenne::ExprHess::ExprHess()`

7.71.2.2 `Couenne::ExprHess::ExprHess(CouenneProblem *)`

7.71.2.3 `Couenne::ExprHess::ExprHess(const ExprHess &)`

7.71.2.4 `Couenne::ExprHess::~ExprHess()`

7.71.3 Member Function Documentation

7.71.3.1 `ExprHess& Couenne::ExprHess::operator=(const ExprHess &)`

7.71.3.2 `ExprHess* Couenne::ExprHess::clone()`

7.71.3.3 `int Couenne::ExprHess::nnz() [inline]`

Definition at line 63 of file CouenneExprHess.hpp.

7.71.3.4 `int* Couenne::ExprHess::iRow() [inline]`

Definition at line 64 of file CouenneExprHess.hpp.

7.71.3.5 `int* Couenne::ExprHess::jCol() [inline]`

Definition at line 65 of file CouenneExprHess.hpp.

7.71.3.6 `int* Couenne::ExprHess::numL() [inline]`

Definition at line 66 of file CouenneExprHess.hpp.

7.71.3.7 `int** Couenne::ExprHess::laml() [inline]`

Definition at line 67 of file CouenneExprHess.hpp.

7.71.3.8 `expression*** Couenne::ExprHess::expr() [inline]`

Definition at line 69 of file CouenneExprHess.hpp.

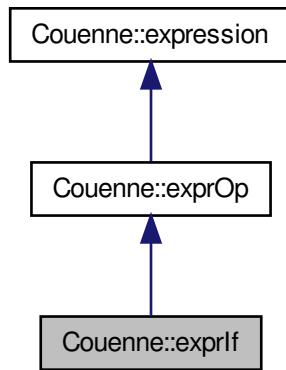
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/partial/[CouenneExprHess.hpp](#)

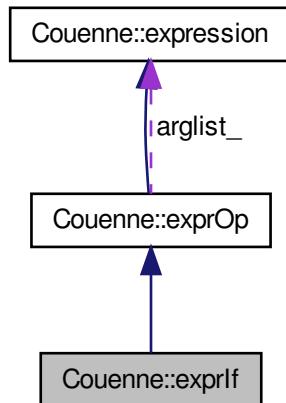
7.72 Couenne::exprlf Class Reference

```
#include <CouenneExprIf.hpp>
```

Inheritance diagram for Couenne::exprlf:



Collaboration diagram for Couenne::exprlf:



Additional Inherited Members

7.72.1 Detailed Description

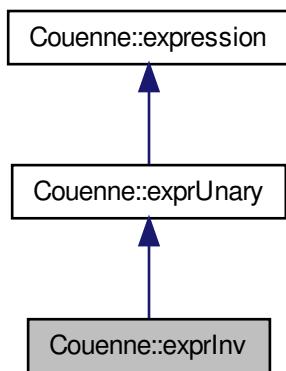
Definition at line 18 of file CouenneExprlf.hpp.

The documentation for this class was generated from the following file:

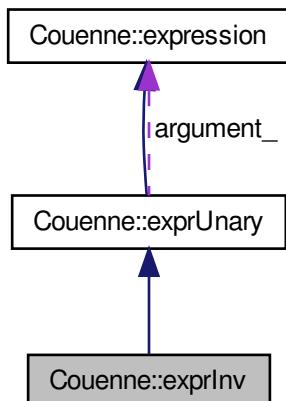
- /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprIf.hpp

7.73 Couenne::exprInv Class Reference

```
class inverse: 1/f(x)  
#include <CouenneExprInv.hpp>  
Inheritance diagram for Couenne::exprInv:
```



Collaboration diagram for Couenne::exprInv:



Public Member Functions

- **exprInv (expression *al)**
Constructors, destructor.
- **expression * clone (Domain *d=NULL) const**
cloning method
- **unary_function F ()**
the operator's function
- **virtual void print (std::ostream &out=std::cout, bool=false) const**
output "1/argument"
- **CouNumber gradientNorm (const double *x)**
return L2 norm of gradient at given point
- **expression * differentiate (int index)**
differentiation
- **virtual int Linearity ()**
get a measure of "how linear" the expression is (see CouenneTypes.h)
- **void getBounds (expression *&, expression *&)**
Get lower and upper bound of an expression (if any)
- **void getBounds (CouNumber &lb, CouNumber &ub)**
Get value of lower and upper bound of an expression (if any)
- **void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)**
*generate equality between *this and *w*
- **virtual enum expr_type code ()**
code for comparisons
- **bool impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)**
implied bound processing
- **virtual CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)**
set up branching object by evaluating many branching points for each expression's arguments
- **virtual bool isBijective () const**
return true if bijective
- **virtual CouNumber inverse (expression *vardep) const**
return inverse of y=f(x)=1/x, i.e., x=1/y
- **virtual bool isCuttable (CouenneProblem *problem, int index) const**
can this expression be further linearized or are we on its concave ("bad") side

Additional Inherited Members

7.73.1 Detailed Description

class inverse: $1/f(x)$

Definition at line 35 of file CouenneExprInv.hpp.

7.73.2 Constructor & Destructor Documentation

7.73.2.1 **Couenne::exprInv::exprInv (expression * *al*) [inline]**

Constructors, destructor.

Definition at line 40 of file CouenneExprInv.hpp.

7.73.3 Member Function Documentation

7.73.3.1 **expression* Couenne::exprInv::clone (Domain * *d* = NULL) const [inline], [virtual]**

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 44 of file CouenneExprInv.hpp.

7.73.3.2 **unary_function Couenne::exprInv::F () [inline], [virtual]**

the operator's function

Reimplemented from [Couenne::exprUnary](#).

Definition at line 48 of file CouenneExprInv.hpp.

7.73.3.3 **virtual void Couenne::exprInv::print (std::ostream & *out* = std::cout, bool *l* = false) const [virtual]**

output "1/argument"

Reimplemented from [Couenne::exprUnary](#).

7.73.3.4 **CouNumber Couenne::exprInv::gradientNorm (const double * *x*) [virtual]**

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

7.73.3.5 **expression* Couenne::exprInv::differentiate (int *index*) [virtual]**

differentiation

Reimplemented from [Couenne::expression](#).

7.73.3.6 **virtual int Couenne::exprInv::Linearity () [inline], [virtual]**

get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::exprUnary](#).

Definition at line 60 of file CouenneExprInv.hpp.

7.73.3.7 **void Couenne::exprInv::getBounds (expression *&, expression *&) [virtual]**

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.73.3.8 **void Couenne::exprInv::getBounds (CouNumber & *lb*, CouNumber & *ub*) [virtual]**

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.73.3.9 void Couenne::exprInv::generateCuts (expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =COUENNE_INFINITY) [virtual]

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

7.73.3.10 virtual enum expr_type Couenne::exprInv::code () [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::exprUnary](#).

Definition at line 79 of file CouenneExprInv.hpp.

7.73.3.11 bool Couenne::exprInv::impliedBound (int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [virtual]

implied bound processing

Reimplemented from [Couenne::expression](#).

7.73.3.12 virtual CouNumber Couenne::exprInv::selectBranch (const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]

set up branching object by evaluating many branching points for each expression's arguments

Reimplemented from [Couenne::expression](#).

7.73.3.13 virtual bool Couenne::exprInv::isBijective () const [inline], [virtual]

return true if bijective

Reimplemented from [Couenne::expression](#).

Definition at line 95 of file CouenneExprInv.hpp.

7.73.3.14 virtual CouNumber Couenne::exprInv::inverse (expression * vardep) const [inline], [virtual]

return inverse of $y=f(x)=1/x$, i.e., $x=1/y$

Reimplemented from [Couenne::expression](#).

Definition at line 98 of file CouenneExprInv.hpp.

7.73.3.15 virtual bool Couenne::exprInv::isCuttable (CouenneProblem * problem, int index) const [virtual]

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

The documentation for this class was generated from the following file:

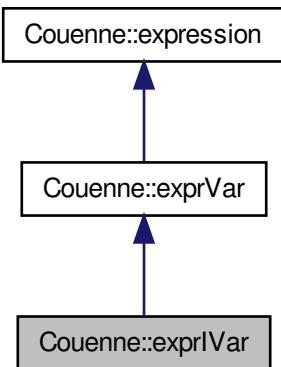
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprInv.hpp](#)

7.74 Couenne::exprVar Class Reference

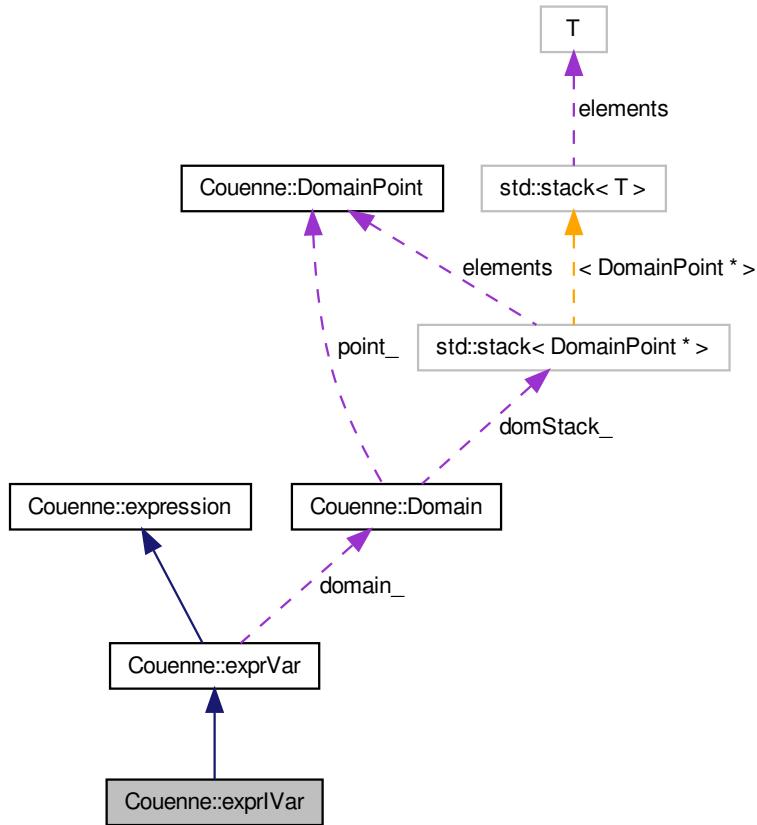
variable-type operator.

```
#include <CouenneExprIVar.hpp>
```

Inheritance diagram for Couenne::exprIVar:



Collaboration diagram for Couenne::exprVar:



Public Member Functions

- `exprVar (int varIndex, Domain *d=NULL)`
Constructor.
- `exprVar (const exprVar &e, Domain *d=NULL)`
Copy constructor – must go.
- `virtual exprVar * clone (Domain *d=NULL) const`
Cloning method.
- `virtual void print (std::ostream &out=std::cout, bool=false) const`
Print.
- `virtual bool isDefinedInteger ()`
Is this expression defined as an integer?
- `virtual bool isInteger ()`
Is this expression integer?

Additional Inherited Members

7.74.1 Detailed Description

variable-type operator.

All variables of the expression must be objects of this class

Definition at line 25 of file CouenneExprVar.hpp.

7.74.2 Constructor & Destructor Documentation

7.74.2.1 Couenne::exprVar::exprVar (int varIndex, Domain * d = NULL) [inline]

Constructor.

Definition at line 30 of file CouenneExprVar.hpp.

7.74.2.2 Couenne::exprVar::exprVar (const exprVar & e, Domain * d = NULL) [inline]

Copy constructor – must go.

Definition at line 34 of file CouenneExprVar.hpp.

7.74.3 Member Function Documentation

7.74.3.1 virtual exprVar* Couenne::exprVar::clone (Domain * d = NULL) const [inline], [virtual]

Cloning method.

Reimplemented from [Couenne::exprVar](#).

Definition at line 38 of file CouenneExprVar.hpp.

7.74.3.2 virtual void Couenne::exprVar::print (std::ostream & out = std::cout, bool = false) const [inline], [virtual]

Print.

Reimplemented from [Couenne::exprVar](#).

Definition at line 42 of file CouenneExprVar.hpp.

7.74.3.3 virtual bool Couenne::exprVar::isDefinedInteger () [inline], [virtual]

is this expression defined as an integer?

Reimplemented from [Couenne::exprVar](#).

Definition at line 46 of file CouenneExprVar.hpp.

7.74.3.4 virtual bool Couenne::exprVar::isInteger () [inline], [virtual]

Is this expression integer?

Reimplemented from [Couenne::exprVar](#).

Definition at line 50 of file CouenneExprVar.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprVar.hpp

7.75 Couenne::ExprJac Class Reference

Jacobian of the problem (computed through [Couenne](#) expression classes).

```
#include <CouenneExprJac.hpp>
```

Public Member Functions

- [ExprJac \(\)](#)
- [ExprJac \(CouenneProblem *\)](#)
- [~ExprJac \(\)](#)
- [ExprJac \(const ExprJac &\)](#)
- [ExprJac * clone \(\)](#)
- [ExprJac & operator= \(const ExprJac &\)](#)
- [int nnz \(\) const](#)
- [int * iRow \(\) const](#)
- [int * jCol \(\) const](#)
- [expression ** expr \(\) const](#)
- [int nRows \(\) const](#)

7.75.1 Detailed Description

Jacobian of the problem (computed through [Couenne](#) expression classes).

Definition at line 21 of file CouenneExprJac.hpp.

7.75.2 Constructor & Destructor Documentation

7.75.2.1 [Couenne::ExprJac::ExprJac \(\)](#)

7.75.2.2 [Couenne::ExprJac::ExprJac \(CouenneProblem * \)](#)

7.75.2.3 [Couenne::ExprJac::~ExprJac \(\)](#)

7.75.2.4 [Couenne::ExprJac::ExprJac \(const ExprJac & \)](#)

7.75.3 Member Function Documentation

7.75.3.1 [ExprJac* Couenne::ExprJac::clone \(\)](#)

7.75.3.2 [ExprJac& Couenne::ExprJac::operator= \(const ExprJac & \)](#)

7.75.3.3 [int Couenne::ExprJac::nnz \(\) const \[inline\]](#)

Definition at line 43 of file CouenneExprJac.hpp.

7.75.3.4 [int* Couenne::ExprJac::iRow \(\) const \[inline\]](#)

Definition at line 44 of file CouenneExprJac.hpp.

7.75.3.5 `int* Couenne::ExprJac::jCol() const [inline]`

Definition at line 45 of file CouenneExprJac.hpp.

7.75.3.6 `expression** Couenne::ExprJac::expr() const [inline]`

Definition at line 47 of file CouenneExprJac.hpp.

7.75.3.7 `int Couenne::ExprJac::nRows() const [inline]`

Definition at line 49 of file CouenneExprJac.hpp.

The documentation for this class was generated from the following file:

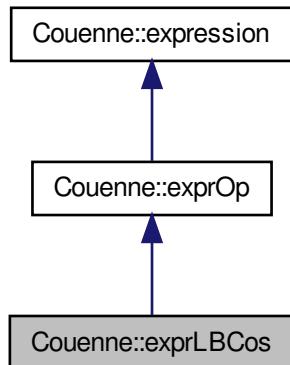
- /home/ted/COIN/trunk/Couenne/src/expression/partial/[CouenneExprJac.hpp](#)

7.76 Couenne::exprLBCos Class Reference

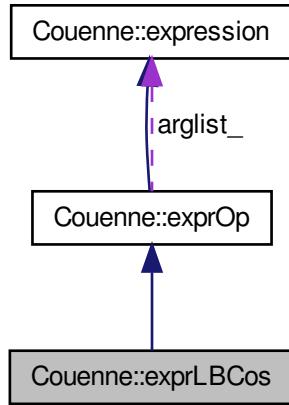
class to compute lower bound of a cosine based on the bounds of its arguments

```
#include <CouenneExprBCos.hpp>
```

Inheritance diagram for Couenne::exprLBCos:



Collaboration diagram for Couenne::exprLBCos:



Public Member Functions

- [exprLBCos \(expression *lb, expression *ub\)](#)
Constructors, destructor.
- [expression * clone \(Domain *d=NULL\) const](#)
cloning method
- [CouNumber operator\(\) \(\)](#)
function for the evaluation of the expression
- enum [pos printPos \(\) const](#)
print position (PRE, INSIDE, POST)
- std::string [printOp \(\) const](#)
print operator

Additional Inherited Members

7.76.1 Detailed Description

class to compute lower bound of a cosine based on the bounds of its arguments

Definition at line 27 of file CouenneExprBCos.hpp.

7.76.2 Constructor & Destructor Documentation

7.76.2.1 Couenne::exprLBCos::exprLBCos (expression * lb, expression * ub) [inline]

Constructors, destructor.

Definition at line 32 of file CouenneExprBCos.hpp.

7.76.3 Member Function Documentation

7.76.3.1 **expression* Couenne::exprLBCos::clone(Domain * d=NULL) const [inline], [virtual]**

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 39 of file CouenneExprBCos.hpp.

7.76.3.2 **CouNumber Couenne::exprLBCos::operator()() [inline], [virtual]**

function for the evaluation of the expression

compute sum

Implements [Couenne::expression](#).

Definition at line 58 of file CouenneExprBCos.hpp.

7.76.3.3 **enum pos Couenne::exprLBCos::printPos() const [inline], [virtual]**

print position (PRE, INSIDE, POST)

Reimplemented from [Couenne::exprOp](#).

Definition at line 47 of file CouenneExprBCos.hpp.

7.76.3.4 **std::string Couenne::exprLBCos::printOp() const [inline], [virtual]**

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 51 of file CouenneExprBCos.hpp.

The documentation for this class was generated from the following file:

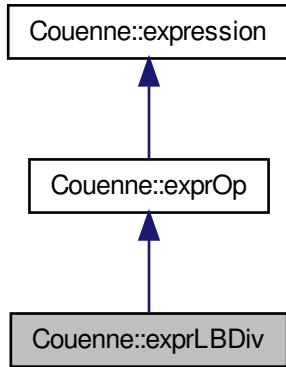
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/[CouenneExprBCos.hpp](#)

7.77 Couenne::exprLBDiv Class Reference

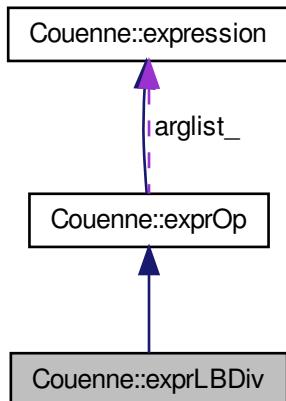
class to compute lower bound of a fraction based on the bounds of both numerator and denominator

```
#include <CouenneExprBDiv.hpp>
```

Inheritance diagram for Couenne::exprLBDiv:



Collaboration diagram for Couenne::exprLBDiv:



Public Member Functions

- [exprLBDiv \(expression **al, int n\)](#)
Constructors, destructor.
- [expression * clone \(Domain *d=NULL\) const](#)
cloning method
- [CouNumber operator\(\) \(\)](#)

- *function for the evaluation of the expression*
- enum [pos printPos \(\) const](#)
 - print position (PRE, INSIDE, POST)*
- std::string [printOp \(\) const](#)
 - print operator*

Additional Inherited Members

7.77.1 Detailed Description

class to compute lower bound of a fraction based on the bounds of both numerator and denominator

Definition at line 37 of file CouenneExprBDiv.hpp.

7.77.2 Constructor & Destructor Documentation

7.77.2.1 Couenne::exprLBDiv::exprLBDiv ([expression ** al](#), int [n](#)) [inline]

Constructors, destructor.

Definition at line 42 of file CouenneExprBDiv.hpp.

7.77.3 Member Function Documentation

7.77.3.1 [expression* Couenne::exprLBDiv::clone \(Domain * d=](#)[NULL](#)) const [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 46 of file CouenneExprBDiv.hpp.

7.77.3.2 CouNumber [Couenne::exprLBDiv::operator\(\) \(\)](#) [inline], [virtual]

function for the evaluation of the expression

compute sum

Implements [Couenne::expression](#).

Definition at line 64 of file CouenneExprBDiv.hpp.

7.77.3.3 enum [pos Couenne::exprLBDiv::printPos \(\) const](#) [inline], [virtual]

print position (PRE, INSIDE, POST)

Reimplemented from [Couenne::exprOp](#).

Definition at line 53 of file CouenneExprBDiv.hpp.

7.77.3.4 std::string [Couenne::exprLBDiv::printOp \(\) const](#) [inline], [virtual]

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 57 of file CouenneExprBDiv.hpp.

The documentation for this class was generated from the following file:

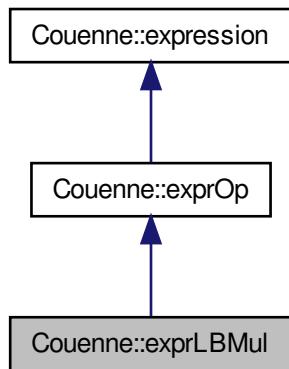
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBDiv.hpp

7.78 Couenne::exprLBMul Class Reference

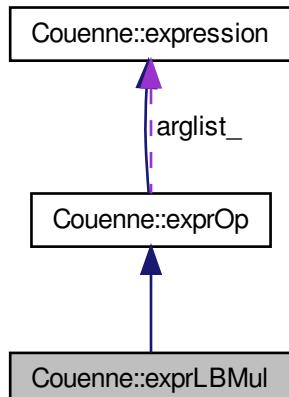
class to compute lower bound of a product based on the bounds of both factors

```
#include <CouenneExprBMul.hpp>
```

Inheritance diagram for Couenne::exprLBMul:



Collaboration diagram for Couenne::exprLBMul:



Public Member Functions

- **exprLBMul (expression **al, int n)**
Constructors, destructor.
- **expression * clone (Domain *d=NULL) const**
cloning method
- **CouNumber operator() ()**
function for the evaluation of the expression
- **enum pos printPos () const**
print position (PRE, INSIDE, POST)
- **std::string printOp () const**
print operator

Additional Inherited Members

7.78.1 Detailed Description

class to compute lower bound of a product based on the bounds of both factors
Definition at line 40 of file CouenneExprBMul.hpp.

7.78.2 Constructor & Destructor Documentation

7.78.2.1 Couenne::exprLBMul::exprLBMul (expression ** al, int n) [inline]

Constructors, destructor.

Definition at line 45 of file CouenneExprBMul.hpp.

7.78.3 Member Function Documentation

7.78.3.1 expression* Couenne::exprLBMul::clone (Domain * d=NULL) const [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 49 of file CouenneExprBMul.hpp.

7.78.3.2 CouNumber Couenne::exprLBMul::operator() () [inline], [virtual]

function for the evaluation of the expression

compute sum

Implements [Couenne::expression](#).

Definition at line 67 of file CouenneExprBMul.hpp.

7.78.3.3 enum pos Couenne::exprLBMul::printPos () const [inline], [virtual]

print position (PRE, INSIDE, POST)

Reimplemented from [Couenne::exprOp](#).

Definition at line 56 of file CouenneExprBMul.hpp.

7.78.3.4 std::string Couenne::exprLBMul::printOp() const [inline], [virtual]

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 60 of file CouenneExprBMul.hpp.

The documentation for this class was generated from the following file:

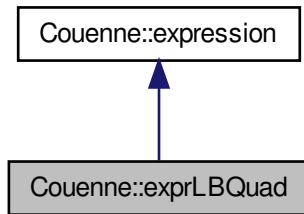
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/[CouenneExprBMul.hpp](#)

7.79 Couenne::exprLBQuad Class Reference

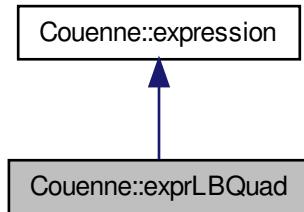
class to compute lower bound of a fraction based on the bounds of both numerator and denominator

```
#include <CouenneExprBQuad.hpp>
```

Inheritance diagram for Couenne::exprLBQuad:



Collaboration diagram for Couenne::exprLBQuad:



Public Member Functions

- [exprLBQuad\(exprQuad *ref\)](#)

- Constructor.*
- `exprLBQuad (const exprLBQuad &src, Domain *d=NULL)`
copy constructor
 - `~exprLBQuad ()`
destructor
 - `expression * clone (Domain *d=NULL) const`
cloning method
 - `CouNumber operator() ()`
function for the evaluation of the expression
 - `virtual void print (std::ostream &s=std::cout, bool descend=false) const`
I/O.

Additional Inherited Members

7.79.1 Detailed Description

class to compute lower bound of a fraction based on the bounds of both numerator and denominator

Definition at line 22 of file CouenneExprBQuad.hpp.

7.79.2 Constructor & Destructor Documentation

7.79.2.1 Couenne::exprLBQuad::exprLBQuad (`exprQuad * ref`) [inline]

Constructor.

Definition at line 29 of file CouenneExprBQuad.hpp.

7.79.2.2 Couenne::exprLBQuad::exprLBQuad (`const exprLBQuad & src, Domain * d = NULL`) [inline]

copy constructor

Definition at line 33 of file CouenneExprBQuad.hpp.

7.79.2.3 Couenne::exprLBQuad::~exprLBQuad () [inline]

destructor

Definition at line 39 of file CouenneExprBQuad.hpp.

7.79.3 Member Function Documentation

7.79.3.1 `expression* Couenne::exprLBQuad::clone (Domain * d = NULL) const` [inline], [virtual]

cloning method

Reimplemented from `Couenne::expression`.

Definition at line 42 of file CouenneExprBQuad.hpp.

7.79.3.2 `CouNumber Couenne::exprLBQuad::operator() ()` [inline], [virtual]

function for the evaluation of the expression

Implements `Couenne::expression`.

Definition at line 46 of file CouenneExprBQuad.hpp.

```
7.79.3.3 virtual void Couenne::exprLBQuad::print( std::ostream & s = std::cout, bool descend = false ) const  
[inline], [virtual]
```

I/O.

Reimplemented from [Couenne::expression](#).

Definition at line 50 of file CouenneExprBQuad.hpp.

The documentation for this class was generated from the following file:

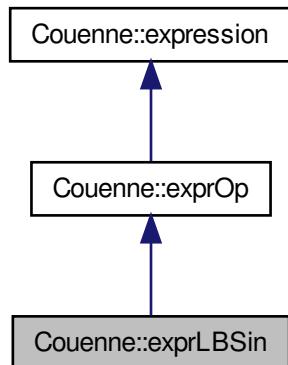
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/[CouenneExprBQuad.hpp](#)

7.80 Couenne::exprLBSin Class Reference

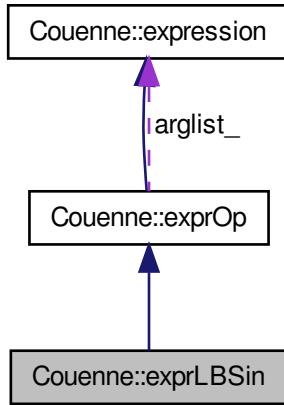
class to compute lower bound of a sine based on the bounds on its arguments

```
#include <CouenneExprBSin.hpp>
```

Inheritance diagram for Couenne::exprLBSin:



Collaboration diagram for Couenne::exprLBSin:



Public Member Functions

- **`exprLBSin (expression *lb, expression *ub)`**
Constructors, destructor.
- **`expression * clone (Domain *d=NULL) const`**
cloning method
- **`CouNumber operator() ()`**
function for the evaluation of the expression
- `enum pos printPos () const`
print position (PRE, INSIDE, POST)
- `std::string printOp () const`
print operator

Additional Inherited Members

7.80.1 Detailed Description

class to compute lower bound of a sine based on the bounds on its arguments

Definition at line 27 of file CouenneExprBSin.hpp.

7.80.2 Constructor & Destructor Documentation

7.80.2.1 Couenne::exprLBSin::exprLBSin (`expression * lb, expression * ub`) [inline]

Constructors, destructor.

Definition at line 32 of file CouenneExprBSin.hpp.

7.80.3 Member Function Documentation

7.80.3.1 **expression* Couenne::exprLBSin::clone (Domain * d=NULL) const** [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 39 of file CouenneExprBSin.hpp.

7.80.3.2 **CouNumber Couenne::exprLBSin::operator() ()** [inline], [virtual]

function for the evaluation of the expression

compute sum

Implements [Couenne::expression](#).

Definition at line 58 of file CouenneExprBSin.hpp.

7.80.3.3 **enum pos Couenne::exprLBSin::printPos () const** [inline], [virtual]

print position (PRE, INSIDE, POST)

Reimplemented from [Couenne::exprOp](#).

Definition at line 47 of file CouenneExprBSin.hpp.

7.80.3.4 **std::string Couenne::exprLBSin::printOp () const** [inline], [virtual]

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 51 of file CouenneExprBSin.hpp.

The documentation for this class was generated from the following file:

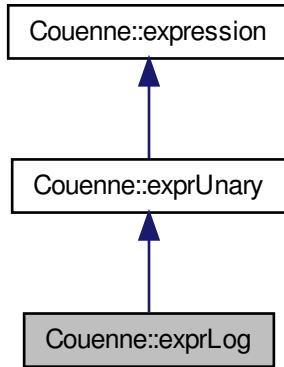
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/[CouenneExprBSin.hpp](#)

7.81 Couenne::exprLog Class Reference

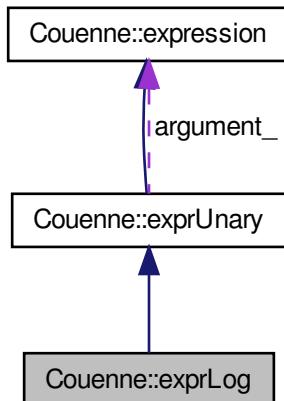
class logarithm, $\log f(x)$

```
#include <CouenneExprLog.hpp>
```

Inheritance diagram for Couenne::exprLog:



Collaboration diagram for Couenne::exprLog:



Public Member Functions

- [exprLog \(expression *al\)](#)
Constructors, destructor.
- [expression * clone \(Domain *d=NULL\) const](#)
cloning method
- [unary_function F \(\)](#)

- `std::string printOp () const`
print operator
- `CouNumber gradientNorm (const double *x)`
return L2 norm of gradient at given point
- `expression * differentiate (int index)`
differentiation
- `void getBounds (expression *&, expression *&)`
Get lower and upper bound of an expression (if any)
- `void getBounds (CouNumber &lb, CouNumber &ub)`
Get value of lower and upper bound of an expression (if any)
- `void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *t_chg_bounds =NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
*generate equality between *this and *w*
- `virtual enum expr_type code ()`
code for comparisons
- `bool impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)`
implied bound processing
- `virtual CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
set up branching object by evaluating many branching points for each expression's arguments
- `virtual bool isBijective () const`
return true if feasible
- `virtual CouNumber inverse (expression *vardep) const`
inverse of this operator
- `virtual bool isCuttable (CouenneProblem *problem, int index) const`
can this expression be further linearized or are we on its concave ("bad") side

Additional Inherited Members

7.81.1 Detailed Description

class logarithm, $\log f(x)$

Definition at line 21 of file CouenneExprLog.hpp.

7.81.2 Constructor & Destructor Documentation

7.81.2.1 Couenne::exprLog::exprLog (expression * al) [inline]

Constructors, destructor.

Definition at line 26 of file CouenneExprLog.hpp.

7.81.3 Member Function Documentation

7.81.3.1 `expression* Couenne::exprLog::clone(Domain * d=NULL) const [inline], [virtual]`

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 30 of file CouenneExprLog.hpp.

7.81.3.2 `unary_function Couenne::exprLog::F() [inline], [virtual]`

the operator's function

Reimplemented from [Couenne::exprUnary](#).

Definition at line 34 of file CouenneExprLog.hpp.

7.81.3.3 `std::string Couenne::exprLog::printOp() const [inline], [virtual]`

print operator

Reimplemented from [Couenne::exprUnary](#).

Definition at line 37 of file CouenneExprLog.hpp.

7.81.3.4 `CouNumber Couenne::exprLog::gradientNorm(const double * x) [virtual]`

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

7.81.3.5 `expression* Couenne::exprLog::differentiate(int index) [virtual]`

differentiation

Reimplemented from [Couenne::expression](#).

7.81.3.6 `void Couenne::exprLog::getBounds(expression *&, expression *&) [virtual]`

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.81.3.7 `void Couenne::exprLog::getBounds(CouNumber & lb, CouNumber & ub) [virtual]`

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.81.3.8 `void Couenne::exprLog::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =COUENNE_INFINITY) [virtual]`

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

7.81.3.9 `virtual enum expr_type Couenne::exprLog::code() [inline], [virtual]`

code for comparisons

Reimplemented from [Couenne::exprUnary](#).

Definition at line 60 of file CouenneExprLog.hpp.

7.81.3.10 `bool Couenne::exprLog::impliedBound(int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign = expression::AUX_EQ) [virtual]`

implied bound processing

Reimplemented from [Couenne::expression](#).

7.81.3.11 `virtual CouNumber Couenne::exprLog::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]`

set up branching object by evaluating many branching points for each expression's arguments

Reimplemented from [Couenne::expression](#).

7.81.3.12 `virtual bool Couenne::exprLog::isBijective() const [inline], [virtual]`

return true if feasible

Reimplemented from [Couenne::expression](#).

Definition at line 77 of file CouenneExprLog.hpp.

7.81.3.13 `virtual CouNumber Couenne::exprLog::inverse(expression * vardep) const [inline], [virtual]`

inverse of this operator

Reimplemented from [Couenne::expression](#).

Definition at line 80 of file CouenneExprLog.hpp.

7.81.3.14 `virtual bool Couenne::exprLog::isCuttable(CouenneProblem * problem, int index) const [virtual]`

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

The documentation for this class was generated from the following file:

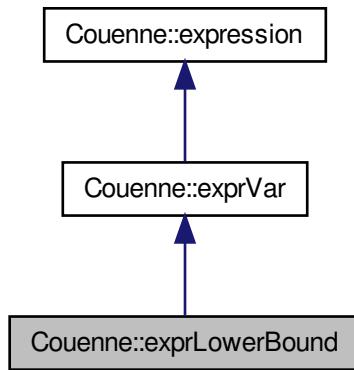
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprLog.hpp](#)

7.82 Couenne::exprLowerBound Class Reference

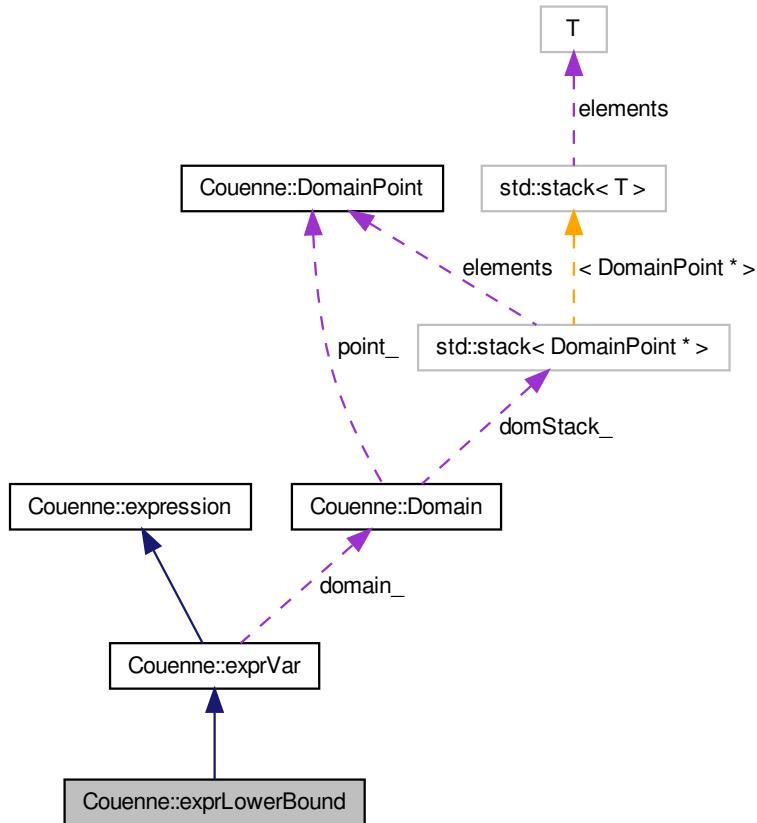
These are bound expression classes.

```
#include <CouenneExprBound.hpp>
```

Inheritance diagram for Couenne::exprLowerBound:



Collaboration diagram for Couenne::exprLowerBound:



Public Member Functions

- enum `nodeType Type () const`
Node type.
- `exprLowerBound (int varIndex, Domain *d=NULL)`
Constructor.
- `exprLowerBound (const exprLowerBound &src, Domain *d=NULL)`
Copy constructor.
- `exprLowerBound * clone (Domain *d=NULL) const`
cloning method
- `void print (std::ostream &out=std::cout, bool=false) const`
Print to ostream.
- `CouNumber operator() ()`
return the value of the variable
- `expression * differentiate (int)`
differentiation

- int [dependsOn](#) (int *, int, enum [dig_type](#) type=[STOP_AT_AUX](#))
dependence on variable set
- virtual int [Linearity](#) ()
get a measure of "how linear" the expression is:
- virtual enum [expr_type](#) [code](#) ()
code for comparisons

Additional Inherited Members

7.82.1 Detailed Description

These are bound expression classes.

They are used in the parametric convexification part to obtain lower/upper bounds of an expression as a function of the expression itself.

For example, the lower and upper bounds to expression ($x_1 - \exp(x_2)$) are ($l_1 - \exp(u_2)$) and ($u_1 - \exp(l_2)$), respectively, where l_1 (l_2) is the lower bound of x_1 (x_2) and u_1 (u_2) is the upper bound of x_1 (x_2).

A lower/upper bound of an expression is a function of all bounds in the expression and is known only when all variables bounds are known. lower bound

Definition at line 38 of file CouenneExprBound.hpp.

7.82.2 Constructor & Destructor Documentation

7.82.2.1 Couenne::exprLowerBound::exprLowerBound (int varIndex, Domain * d=NULL) [inline]

Constructor.

Definition at line 47 of file CouenneExprBound.hpp.

7.82.2.2 Couenne::exprLowerBound::exprLowerBound (const exprLowerBound & src, Domain * d=NULL) [inline]

Copy constructor.

Definition at line 51 of file CouenneExprBound.hpp.

7.82.3 Member Function Documentation

7.82.3.1 enum nodeType Couenne::exprLowerBound::Type () const [inline], [virtual]

[Node](#) type.

Reimplemented from [Couenne::exprVar](#).

Definition at line 43 of file CouenneExprBound.hpp.

7.82.3.2 exprLowerBound* Couenne::exprLowerBound::clone (Domain * d=NULL) const [inline], [virtual]

cloning method

Reimplemented from [Couenne::exprVar](#).

Definition at line 55 of file CouenneExprBound.hpp.

7.82.3.3 void Couenne::exprLowerBound::print(std::ostream & *out* = std::cout, bool = false) const [inline], [virtual]

Print to ostream.

Reimplemented from [Couenne::exprVar](#).

Definition at line 59 of file CouenneExprBound.hpp.

7.82.3.4 CouNumber Couenne::exprLowerBound::operator()() [inline], [virtual]

return the value of the variable

Reimplemented from [Couenne::exprVar](#).

Definition at line 64 of file CouenneExprBound.hpp.

7.82.3.5 expression* Couenne::exprLowerBound::differentiate(int) [inline], [virtual]

differentiation

Reimplemented from [Couenne::exprVar](#).

Definition at line 68 of file CouenneExprBound.hpp.

7.82.3.6 int Couenne::exprLowerBound::dependsOn(int *, int , enum dig_type type = STOP_AT_AUX) [inline], [virtual]

dependence on variable set

Reimplemented from [Couenne::expression](#).

Definition at line 72 of file CouenneExprBound.hpp.

7.82.3.7 virtual int Couenne::exprLowerBound::Linearity() [inline], [virtual]

get a measure of "how linear" the expression is:

Reimplemented from [Couenne::exprVar](#).

Definition at line 76 of file CouenneExprBound.hpp.

7.82.3.8 virtual enum expr_type Couenne::exprLowerBound::code() [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::exprVar](#).

Definition at line 80 of file CouenneExprBound.hpp.

The documentation for this class was generated from the following file:

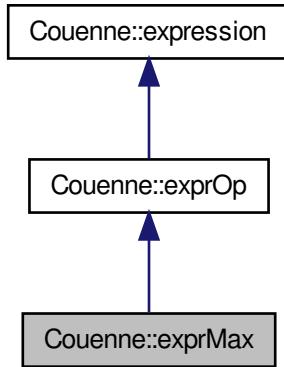
- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprBound.hpp](#)

7.83 Couenne::exprMax Class Reference

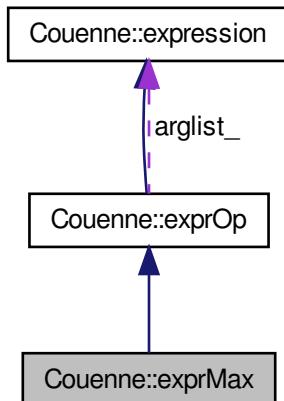
class for maxima

```
#include <CouenneExprMax.hpp>
```

Inheritance diagram for Couenne::exprMax:



Collaboration diagram for Couenne::exprMax:



Public Member Functions

- `exprMax (expression **al, int n)`
Constructor.
- `exprMax (expression *el0, expression *el1)`
Constructor with only two arguments.
- `exprMax * clone (Domain *d=NULL) const`

- *cloning method*
- `std::string printOp () const`
print operator
- `enum pos printPos () const`
print position
- `CouNumber operator() ()`
function for the evaluation of the expression
- `expression * differentiate (int)`
differentiation
- `expression * simplify ()`
simplification
- `virtual int Linearity ()`
get a measure of "how linear" the expression is (see CouenneTypes.h)
- `void getBounds (expression *&, expression *&)`
Get lower and upper bound of an expression (if any)
- `virtual exprAux * standardize (CouenneProblem *, bool addAux=true)`
reduce expression in standard form, creating additional aux variables (and constraints)
- `void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *tchgb=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
*generate equality between *this and *w*
- `virtual enum expr_type code ()`
code for comparisons

Additional Inherited Members

7.83.1 Detailed Description

class for maxima

Definition at line 22 of file CouenneExprMax.hpp.

7.83.2 Constructor & Destructor Documentation

7.83.2.1 Couenne::exprMax::exprMax (`expression ** al, int n`) [inline]

Constructor.

Definition at line 27 of file CouenneExprMax.hpp.

7.83.2.2 Couenne::exprMax::exprMax (`expression * el0, expression * el1`) [inline]

Constructor with only two arguments.

Definition at line 31 of file CouenneExprMax.hpp.

7.83.3 Member Function Documentation

7.83.3.1 `exprMax* Couenne::exprMax::clone (Domain * d=NULL) const` [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 38 of file CouenneExprMax.hpp.

7.83.3.2 std::string Couenne::exprMax::printOp() const [inline], [virtual]

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 42 of file CouenneExprMax.hpp.

7.83.3.3 enum pos Couenne::exprMax::printPos() const [inline], [virtual]

print position

Reimplemented from [Couenne::exprOp](#).

Definition at line 46 of file CouenneExprMax.hpp.

7.83.3.4 CouNumber Couenne::exprMax::operator()() [inline], [virtual]

function for the evaluation of the expression

compute maximum

Implements [Couenne::expression](#).

Definition at line 87 of file CouenneExprMax.hpp.

7.83.3.5 expression* Couenne::exprMax::differentiate(int) [inline], [virtual]

differentiation

Reimplemented from [Couenne::expression](#).

Definition at line 53 of file CouenneExprMax.hpp.

7.83.3.6 expression* Couenne::exprMax::simplify() [inline], [virtual]

simplification

Reimplemented from [Couenne::exprOp](#).

Definition at line 57 of file CouenneExprMax.hpp.

7.83.3.7 virtual int Couenne::exprMax::Linearity() [inline], [virtual]

get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::exprOp](#).

Definition at line 61 of file CouenneExprMax.hpp.

7.83.3.8 void Couenne::exprMax::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.83.3.9 virtual exprAux* Couenne::exprMax::standardize(CouenneProblem * , bool addAux = true) [inline], [virtual]

reduce expression in standard form, creating additional aux variables (and constraints)

Reimplemented from [Couenne::exprOp](#).

Definition at line 69 of file CouenneExprMax.hpp.

```
7.83.3.10 void Couenne::exprMax::generateCuts ( expression * w, OsiCuts & cs, const CouenneCutGenerator *
    cg, t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber =
    COUENNE_INFINITY ) [virtual]
```

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

```
7.83.3.11 virtual enum expr_type Couenne::exprMax::code ( ) [inline], [virtual]
```

code for comparisons

Reimplemented from [Couenne::exprOp](#).

Definition at line 80 of file CouenneExprMax.hpp.

The documentation for this class was generated from the following file:

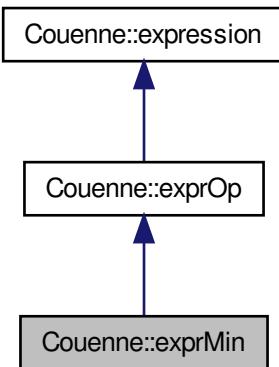
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprMax.hpp](#)

7.84 Couenne::exprMin Class Reference

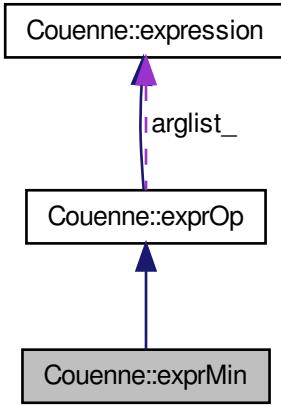
class for minima

```
#include <CouenneExprMin.hpp>
```

Inheritance diagram for Couenne::exprMin:



Collaboration diagram for Couenne::exprMin:



Public Member Functions

- `exprMin (expression **al, int n)`
Constructor.
- `exprMin (expression *el0, expression *el1)`
Constructor with only two arguments.
- `exprMin * clone (Domain *d=NULL) const`
Cloning method.
- `std::string printOp () const`
Print operator.
- `enum pos printPos () const`
Print operator.
- `CouNumber operator() ()`
Function for the evaluation of the expression.
- `expression * differentiate (int)`
Differentiation.
- `expression * simplify ()`
Simplification.
- `virtual int Linearity ()`
get a measure of "how linear" the expression is (see CouenneTypes.h)
- `void getBounds (expression *&, expression *&)`
Get lower and upper bound of an expression (if any)
- `virtual exprAux * standardize (CouenneProblem *, bool addAux=true)`
Reduce expression in standard form, creating additional aux variables (and constraints)
- `void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
*Generate equality between *this and *w.*

- virtual enum [expr_type code \(\)](#)

Code for comparisons.

Additional Inherited Members

7.84.1 Detailed Description

class for minima

Definition at line 29 of file CouenneExprMin.hpp.

7.84.2 Constructor & Destructor Documentation

7.84.2.1 Couenne::exprMin::exprMin (**expression** ** *al*, **int** *n*) [inline]

Constructor.

Definition at line 34 of file CouenneExprMin.hpp.

7.84.2.2 Couenne::exprMin::exprMin (**expression** * *eI0*, **expression** * *eI1*) [inline]

Constructor with only two arguments.

Definition at line 38 of file CouenneExprMin.hpp.

7.84.3 Member Function Documentation

7.84.3.1 **exprMin*** Couenne::exprMin::clone (**Domain** * *d*=NULL)**const** [inline], [virtual]

Cloning method.

Reimplemented from [Couenne::expression](#).

Definition at line 45 of file CouenneExprMin.hpp.

7.84.3.2 **std::string** Couenne::exprMin::printOp ()**const** [inline], [virtual]

Print operator.

Reimplemented from [Couenne::exprOp](#).

Definition at line 49 of file CouenneExprMin.hpp.

7.84.3.3 **enum pos** Couenne::exprMin::printPos ()**const** [inline], [virtual]

Print operator.

Reimplemented from [Couenne::exprOp](#).

Definition at line 53 of file CouenneExprMin.hpp.

7.84.3.4 **CouNumber** Couenne::exprMin::operator() () [inline], [virtual]

Function for the evaluation of the expression.

Compute minimum.

Implements [Couenne::expression](#).

Definition at line 94 of file CouenneExprMin.hpp.

7.84.3.5 expression* Couenne::exprMin::differentiate(int) [inline], [virtual]

Differentiation.

Reimplemented from [Couenne::expression](#).

Definition at line 60 of file CouenneExprMin.hpp.

7.84.3.6 expression* Couenne::exprMin::simplify() [inline], [virtual]

Simplification.

Reimplemented from [Couenne::exprOp](#).

Definition at line 64 of file CouenneExprMin.hpp.

7.84.3.7 virtual int Couenne::exprMin::Linearity() [inline], [virtual]

get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::exprOp](#).

Definition at line 68 of file CouenneExprMin.hpp.

7.84.3.8 void Couenne::exprMin::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.84.3.9 virtual exprAux* Couenne::exprMin::standardize(CouenneProblem * , bool addAux = true) [inline], [virtual]

Reduce expression in standard form, creating additional aux variables (and constraints)

Reimplemented from [Couenne::exprOp](#).

Definition at line 76 of file CouenneExprMin.hpp.

7.84.3.10 void Couenne::exprMin::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

Generate equality between *this and *w.

Reimplemented from [Couenne::expression](#).

7.84.3.11 virtual enum expr_type Couenne::exprMin::code() [inline], [virtual]

Code for comparisons.

Reimplemented from [Couenne::exprOp](#).

Definition at line 87 of file CouenneExprMin.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprMin.hpp](#)

7.85 Couenne::exprMultiLin Class Reference

another class for multiplications, $\prod_{i=1}^n f_i(x)$

```
#include <CouenneExprMultiLin.hpp>
```

Public Member Functions

- **exprMultiLin (expression **, int)**
Constructor.
- **exprMultiLin (expression *, expression *)**
Constructor with two arguments.
- **CouNumber gradientNorm (const double *x)**
return L2 norm of gradient at given point
- **expression * differentiate (int index)**
differentiation
- **expression * simplify ()**
simplification
- **virtual int Linearity ()**
get a measure of "how linear" the expression is:
- **virtual void getBounds (expression *&, expression *&)**
Get lower and upper bound of an expression (if any)
- **virtual void getBounds (CouNumber &lb, CouNumber &ub)**
Get value of lower and upper bound of an expression (if any)
- **virtual exprAux * standardize (CouenneProblem *p, bool addAux=true)**
reduce expression in standard form, creating additional aux variables (and constraints)
- **void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)**
*generate equality between *this and *w*
- **virtual enum expr_type code ()**
code for comparison
- **bool impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum Couenne::expression::aux-Sign=Couenne::expression::AUX_EQ)**
implied bound processing
- **virtual CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)**
set up branching object by evaluating many branching points for each expression's arguments
- **virtual void closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right) const**
compute y^{lv} and y^{uv} for Violation Transfer algorithm

Protected Member Functions

- **int impliedBoundMul (CouNumber wl, CouNumber wu, std::vector< CouNumber > &xl, std::vector< CouNumber > &xu, std::vector< std::pair< int, CouNumber > > &nl, std::vector< std::pair< int, CouNumber > > &nu)**
inferring bounds on factors of a product
- **CouNumber balancedMul (const OsiBranchingInformation *info, int index, int wind)**
balanced strategy for branching point selection in products
- **virtual bool isCuttable (CouenneProblem *problem, int index) const**
can this expression be further linearized or are we on its concave ("bad") side

7.85.1 Detailed Description

another class for multiplications, $\prod_{i=1}^n f_i(x)$

Definition at line 21 of file CouenneExprMultiLin.hpp.

7.85.2 Constructor & Destructor Documentation

7.85.2.1 Couenne::exprMultiLin::exprMultiLin (expression ** , int)

Constructor.

7.85.2.2 Couenne::exprMultiLin::exprMultiLin (expression * , expression *)

Constructor with two arguments.

7.85.3 Member Function Documentation

7.85.3.1 CouNumber Couenne::exprMultiLin::gradientNorm (const double * x)

return l-2 norm of gradient at given point

7.85.3.2 expression* Couenne::exprMultiLin::differentiate (int index)

differentiation

7.85.3.3 expression* Couenne::exprMultiLin::simplify ()

simplification

7.85.3.4 virtual int Couenne::exprMultiLin::Linearity () [virtual]

get a measure of "how linear" the expression is:

7.85.3.5 virtual void Couenne::exprMultiLin::getBounds (expression *& , expression *&) [virtual]

Get lower and upper bound of an expression (if any)

7.85.3.6 virtual void Couenne::exprMultiLin::getBounds (CouNumber & lb, CouNumber & ub) [virtual]

Get value of lower and upper bound of an expression (if any)

7.85.3.7 virtual exprAux* Couenne::exprMultiLin::standardize (CouenneProblem * p, bool addAux = true) [virtual]

reduce expression in standard form, creating additional aux variables (and constraints)

7.85.3.8 void Couenne::exprMultiLin::generateCuts (expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * =NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY)

generate equality between *this and *w

7.85.3.9 virtual enum expr_type Couenne::exprMultiLin::code () [inline], [virtual]

code for comparison

Definition at line 61 of file CouenneExprMultiLin.hpp.

7.85.3.10 `bool Couenne::exprMultiLin::impliedBound(int , CouNumber * , CouNumber * , t_chg_bounds * , enum Couenne::expression::auxSign = Couenne::expression::AUX_EQ)`

implied bound processing

7.85.3.11 `virtual CouNumber Couenne::exprMultiLin::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]`

set up branching object by evaluating many branching points for each expression's arguments

7.85.3.12 `virtual void Couenne::exprMultiLin::closestFeasible(expression * varind, expression * vardep, CouNumber & left, CouNumber & right) const [virtual]`

compute y^{lv} and y^{uv} for Violation Transfer algorithm

7.85.3.13 `int Couenne::exprMultiLin::impliedBoundMul(CouNumber wl, CouNumber wu, std::vector< CouNumber > & xl, std::vector< CouNumber > & xu, std::vector< std::pair< int, CouNumber > > & nl, std::vector< std::pair< int, CouNumber > > & nu) [protected]`

inferring bounds on factors of a product

7.85.3.14 `CouNumber Couenne::exprMultiLin::balancedMul(const OsiBranchingInformation * info, int index, int wind) [protected]`

balanced strategy for branching point selection in products

7.85.3.15 `virtual bool Couenne::exprMultiLin::isCuttable(CouenneProblem * problem, int index) const [inline], [protected], [virtual]`

can this expression be further linearized or are we on its concave ("bad") side

Definition at line 97 of file CouenneExprMultiLin.hpp.

The documentation for this class was generated from the following file:

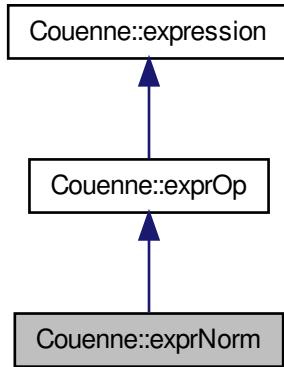
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprMultiLin.hpp](#)

7.86 Couenne::exprNorm Class Reference

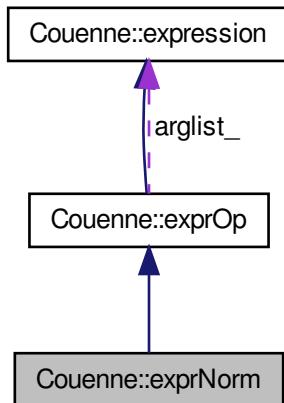
Class for p -norms, $\|f(x)\|_p = (\sum_{i=1}^n f_i(x)^p)^{\frac{1}{p}}$.

```
#include <CouenneExprNorm.hpp>
```

Inheritance diagram for Couenne::exprNorm:



Collaboration diagram for Couenne::exprNorm:



Additional Inherited Members

7.86.1 Detailed Description

Class for p -norms, $\|f(x)\|_p = (\sum_{i=1}^n f_i(x)^p)^{\frac{1}{p}}$.

Definition at line 20 of file CouenneExprNorm.hpp.

The documentation for this class was generated from the following file:

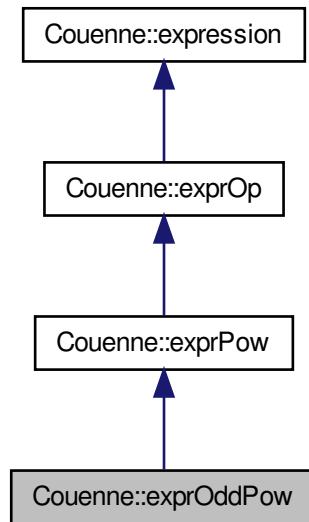
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprNorm.hpp](#)

7.87 Couenne::exprOddPow Class Reference

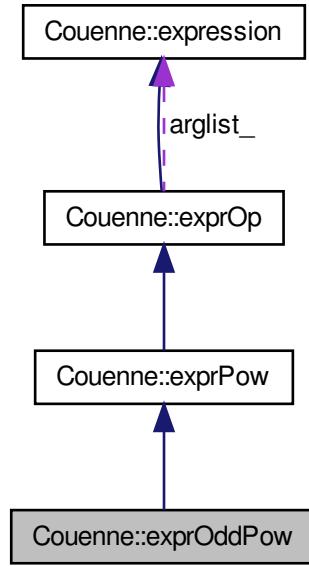
Power of an expression (binary operator), $f(x)^k$ with k constant.

```
#include <CouenneExprOddPow.hpp>
```

Inheritance diagram for Couenne::exprOddPow:



Collaboration diagram for Couenne::exprOddPow:



Public Member Functions

- `exprOddPow (expression **al, int n=2)`
Constructor.
- `exprOddPow (expression *arg0, expression *arg1)`
Constructor with only two arguments.
- `expression * clone (Domain *d=NULL) const`
cloning method
- `std::string printOp () const`
print operator
- `CouNumber operator() ()`
function for the evaluation of the expression
- `void getBounds (expression *&, expression *&)`
Get lower and upper bound of an expression (if any)
- `void getBounds (CouNumber &lb, CouNumber &ub)`
Get value of lower and upper bound of an expression (if any)
- `exprAux * standardize (CouenneProblem *p, bool addAux=true)`
reduce expression in standard form, creating additional aux variables (and constraints)
- `void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
*generate equality between *this and *w*
- `expression * getFixVar ()`

- *return an index to the variable's argument that is better fixed in a branching rule for solving a nonconvexity gap*
- virtual enum `expr_type code ()`
 - code for comparison*
- bool `impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)`
 - implied bound processing*
- virtual `CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
 - set up branching object by evaluating many branching points for each expression's arguments*
- virtual bool `isCuttable (CouenneProblem *problem, int index) const`
 - can this expression be further linearized or are we on its concave ("bad") side*

Additional Inherited Members

7.87.1 Detailed Description

Power of an expression (binary operator), $f(x)^k$ with k constant.

Definition at line 22 of file CouenneExprOddPow.hpp.

7.87.2 Constructor & Destructor Documentation

7.87.2.1 Couenne::exprOddPow::exprOddPow (`expression ** al, int n = 2`) [inline]

Constructor.

Definition at line 27 of file CouenneExprOddPow.hpp.

7.87.2.2 Couenne::exprOddPow::exprOddPow (`expression * arg0, expression * arg1`) [inline]

Constructor with only two arguments.

Definition at line 31 of file CouenneExprOddPow.hpp.

7.87.3 Member Function Documentation

7.87.3.1 `expression* Couenne::exprOddPow::clone (Domain * d = NULL) const` [inline], [virtual]

cloning method

Reimplemented from [Couenne::exprPow](#).

Definition at line 35 of file CouenneExprOddPow.hpp.

7.87.3.2 `std::string Couenne::exprOddPow::printOp () const` [inline], [virtual]

print operator

Reimplemented from [Couenne::exprPow](#).

Definition at line 39 of file CouenneExprOddPow.hpp.

7.87.3.3 `CouNumber Couenne::exprOddPow::operator() ()` [inline], [virtual]

function for the evaluation of the expression

compute power

Reimplemented from [Couenne::exprPow](#).

Definition at line 90 of file CouenneExprOddPow.hpp.

7.87.3.4 void Couenne::exprOddPow::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::exprPow](#).

7.87.3.5 void Couenne::exprOddPow::getBounds(CouNumber & lb, CouNumber & ub) [virtual]

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::exprPow](#).

7.87.3.6 exprAux* Couenne::exprOddPow::standardize(CouenneProblem * p, bool addAux = true) [virtual]

reduce expression in standard form, creating additional aux variables (and constraints)

Reimplemented from [Couenne::exprPow](#).

7.87.3.7 void Couenne::exprOddPow::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

generate equality between *this and *w

Reimplemented from [Couenne::exprPow](#).

7.87.3.8 expression* Couenne::exprOddPow::getFixVar() [inline], [virtual]

return an index to the variable's argument that is better fixed in a branching rule for solving a nonconvexity gap

Reimplemented from [Couenne::exprPow](#).

Definition at line 64 of file CouenneExprOddPow.hpp.

7.87.3.9 virtual enum expr_type Couenne::exprOddPow::code() [inline], [virtual]

code for comparison

Reimplemented from [Couenne::exprPow](#).

Definition at line 68 of file CouenneExprOddPow.hpp.

7.87.3.10 bool Couenne::exprOddPow::impliedBound(int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [virtual]

implied bound processing

Reimplemented from [Couenne::exprPow](#).

7.87.3.11 virtual CouNumber Couenne::exprOddPow::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]

set up branching object by evaluating many branching points for each expression's arguments

Reimplemented from [Couenne::exprPow](#).

7.87.3.12 virtual bool Couenne::exprOddPow::isCuttable (CouenneProblem * problem, int index) const [virtual]

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::exprPow](#).

The documentation for this class was generated from the following file:

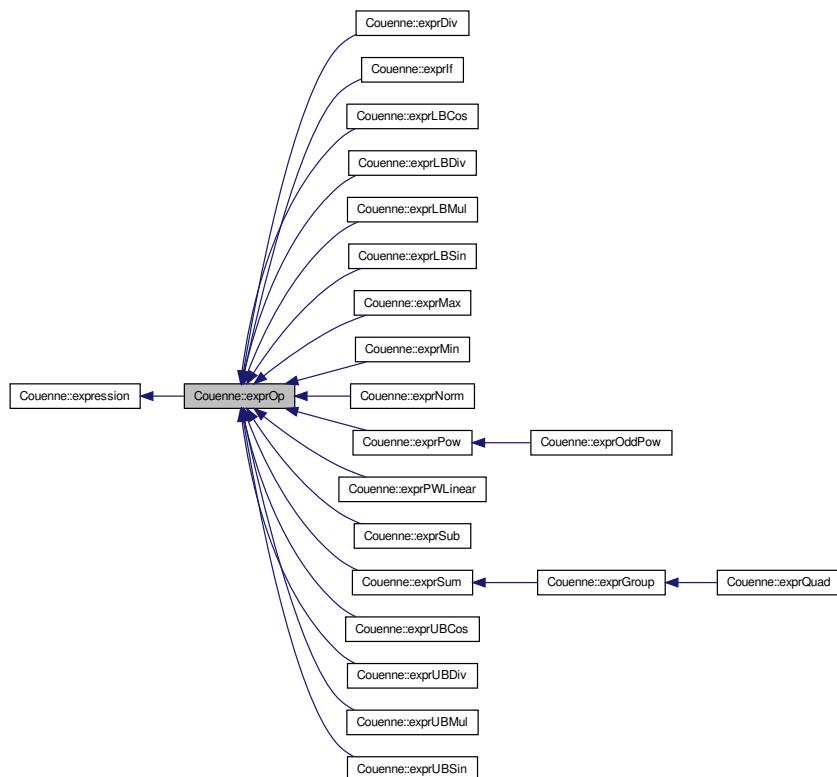
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprOddPow.hpp](#)

7.88 Couenne::exprOp Class Reference

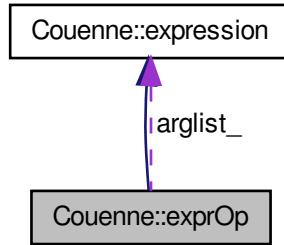
general n-ary operator-type expression: requires argument list.

```
#include <CouenneExprOp.hpp>
```

Inheritance diagram for Couenne::exprOp:



Collaboration diagram for Couenne::exprOp:



Public Member Functions

- virtual enum `nodeType Type () const`
Node type.
- `exprOp (expression **arglist, int nargs)`
Constructor.
- `exprOp (expression *arg0, expression *arg1)`
Constructor with two arguments (for convenience)
- virtual `~exprOp ()`
Destructor.
- `exprOp (const exprOp &e, Domain *d=NULL)`
Copy constructor: only allocate space for argument list, which will be copied with `clonearglist()`
- `expression ** ArgList () const`
return argument list
- virtual void `ArgList (expression **al)`
set arglist (used in deleting nodes without deleting children)
- int `nArgs () const`
return number of arguments
- virtual void `print (std::ostream &out=std::cout, bool=false) const`
I/O.
- virtual enum `pos printPos () const`
print position (PRE, INSIDE, POST)
- virtual std::string `printOp () const`
print operator
- virtual int `DepList (std::set< int > &depelist, enum dig_type type=ORIG_ONLY)`
fill in the set with all indices of variables appearing in the expression
- virtual `expression * simplify ()`
simplification
- `expression ** clonearglist (Domain *d=NULL) const`
clone argument list (for use with clone method)
- int `shrink_arglist (CouNumber, CouNumber)`

- compress argument list
- virtual int **Linearity** ()
get a measure of "how linear" the expression is (see *CouenneTypes.h*)
- virtual **exprAux *** **standardize** (**CouenneProblem** *, bool addAux=true)
generate auxiliary variable
- virtual enum **expr_type** **code** ()
return code to classify type of expression
- virtual bool **isInteger** ()
is this expression integer?
- virtual int **compare** (**exprOp** &)
compare with other generic **exprOp**
- virtual int **rank** ()
used in rank-based branching variable choice
- virtual void **fillDepSet** (std::set< **DepNode** *, **compNode** > *dep, **DepGraph** *g)
fill in dependence structure update dependence set with index of this variable
- virtual void **replace** (**exprVar** *, **exprVar** *)
replace variable with other
- virtual void **realign** (const **CouenneProblem** *p)
empty function to redirect variables to proper variable vector

Protected Attributes

- **expression ** arglist_**
argument list is an array of pointers to other expressions
- int **nargs_**
number of arguments (cardinality of arglist)

Additional Inherited Members

7.88.1 Detailed Description

general n-ary operator-type expression: requires argument list.

All non-unary and non-leaf operators, i.e., sum, subtraction, multiplication, power, division, max, min, etc. are derived from this class.

Definition at line 31 of file *CouenneExprOp.hpp*.

7.88.2 Constructor & Destructor Documentation

7.88.2.1 Couenne::exprOp::exprOp (**expression ** arglist**, **int nargs**) [inline]

Constructor.

Definition at line 45 of file *CouenneExprOp.hpp*.

7.88.2.2 Couenne::exprOp::exprOp (**expression * arg0**, **expression * arg1**) [inline]

Constructor with two arguments (for convenience)

Definition at line 51 of file *CouenneExprOp.hpp*.

7.88.2.3 virtual Couenne::exprOp::~exprOp() [virtual]

Destructor.

7.88.2.4 Couenne::exprOp::exprOp(const exprOp & e, Domain * d = NULL) [inline]

Copy constructor: only allocate space for argument list, which will be copied with [clonearglist\(\)](#)

Definition at line 61 of file CouenneExprOp.hpp.

7.88.3 Member Function Documentation**7.88.3.1 virtual enum nodeType Couenne::exprOp::Type() const [inline], [virtual]**

[Node](#) type.

Reimplemented from [Couenne::expression](#).

Definition at line 41 of file CouenneExprOp.hpp.

7.88.3.2 expression Couenne::exprOp::ArgList() const [inline], [virtual]**

return argument list

Reimplemented from [Couenne::expression](#).

Definition at line 66 of file CouenneExprOp.hpp.

7.88.3.3 virtual void Couenne::exprOp::ArgList(expression ** al) [inline], [virtual]

set arglist (used in deleting nodes without deleting children)

Reimplemented from [Couenne::expression](#).

Definition at line 70 of file CouenneExprOp.hpp.

7.88.3.4 int Couenne::exprOp::nArgs() const [inline], [virtual]

return number of arguments

Reimplemented from [Couenne::expression](#).

Definition at line 74 of file CouenneExprOp.hpp.

7.88.3.5 virtual void Couenne::exprOp::print(std::ostream & out = std::cout, bool = false) const [virtual]

I/O.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.88.3.6 virtual enum pos Couenne::exprOp::printPos() const [inline], [virtual]

print position (PRE, INSIDE, POST)

Reimplemented in [Couenne::exprUBMul](#), [Couenne::exprUBCos](#), [Couenne::exprUBSin](#), [Couenne::exprUBDiv](#), [Couenne::exprLBMul](#), [Couenne::exprLBDiv](#), [Couenne::exprMin](#), [Couenne::exprLBCos](#), [Couenne::exprLBSin](#), and [Couenne::exprMax](#).

Definition at line 82 of file CouenneExprOp.hpp.

7.88.3.7 `virtual std::string Couenne::exprOp::printOp() const [inline], [virtual]`

print operator

Reimplemented in `Couenne::exprUBMul`, `Couenne::exprUBDiv`, `Couenne::exprUBCos`, `Couenne::exprUBSin`, `Couenne::exprLBMul`, `Couenne::exprLBDiv`, `Couenne::exprPow`, `Couenne::exprLBCos`, `Couenne::exprLBSin`, `Couenne::exprMin`, `Couenne::exprMax`, `Couenne::exprDiv`, `Couenne::exprSum`, `Couenne::exprOddPow`, and `Couenne::exprSub`.

Definition at line 86 of file `CouenneExprOp.hpp`.

7.88.3.8 `virtual int Couenne::exprOp::DepList(std::set< int > & depList, enum dig_type type = ORIG_ONLY) [virtual]`

fill in the set with all indices of variables appearing in the expression

Reimplemented from `Couenne::expression`.

Reimplemented in `Couenne::exprQuad`, and `Couenne::exprGroup`.

7.88.3.9 `virtual expression* Couenne::exprOp::simplify() [virtual]`

simplification

Reimplemented from `Couenne::expression`.

Reimplemented in `Couenne::exprQuad`, `Couenne::exprGroup`, `Couenne::exprMin`, `Couenne::exprPow`, `Couenne::exprMax`, `Couenne::exprDiv`, `Couenne::exprSum`, and `Couenne::exprSub`.

7.88.3.10 `expression** Couenne::exprOp::clonearglist(Domain * d = NULL) const [inline]`

clone argument list (for use with clone method)

Definition at line 97 of file `CouenneExprOp.hpp`.

7.88.3.11 `int Couenne::exprOp::shrink_arglist(CouNumber , CouNumber)`

compress argument list

7.88.3.12 `virtual int Couenne::exprOp::Linearity() [inline], [virtual]`

get a measure of "how linear" the expression is (see `CouenneTypes.h`)

Reimplemented from `Couenne::expression`.

Reimplemented in `Couenne::exprQuad`, `Couenne::exprGroup`, `Couenne::exprMin`, `Couenne::exprPow`, `Couenne::exprMax`, `Couenne::exprDiv`, `Couenne::exprSum`, and `Couenne::exprSub`.

Definition at line 110 of file `CouenneExprOp.hpp`.

7.88.3.13 `virtual exprAux* Couenne::exprOp::standardize(CouenneProblem * , bool addAux = true) [virtual]`

generate auxiliary variable

Reimplemented from `Couenne::expression`.

Reimplemented in `Couenne::exprPow`, `Couenne::exprMin`, `Couenne::exprDiv`, `Couenne::exprMax`, `Couenne::exprSub`, `Couenne::exprSum`, and `Couenne::exprOddPow`.

7.88.3.14 `virtual enum expr_type Couenne::exprOp::code() [inline], [virtual]`

return code to classify type of expression

Reimplemented from `Couenne::expression`.

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprGroup](#), [Couenne::exprPow](#), [Couenne::exprMin](#), [Couenne::exprDiv](#), [Couenne::exprMax](#), [Couenne::exprSub](#), [Couenne::exprSum](#), and [Couenne::exprOddPow](#).

Definition at line 117 of file [CouenneExprOp.hpp](#).

7.88.3.15 virtual bool Couenne::exprOp::isInteger() [virtual]

is this expression integer?

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), [Couenne::exprGroup](#), [Couenne::exprDiv](#), and [Couenne::exprPow](#).

7.88.3.16 virtual int Couenne::exprOp::compare(exprOp &) [virtual]

compare with other generic [exprOp](#)

7.88.3.17 virtual int Couenne::exprOp::rank() [virtual]

used in rank-based branching variable choice

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.88.3.18 virtual void Couenne::exprOp::fillDepSet(std::set< DepNode *, compNode *> * dep, DepGraph * g) [inline], [virtual]

fill in dependence structure update dependence set with index of this variable

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

Definition at line 131 of file [CouenneExprOp.hpp](#).

7.88.3.19 virtual void Couenne::exprOp::replace(exprVar *, exprVar *) [virtual]

replace variable with other

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.88.3.20 virtual void Couenne::exprOp::realign(const CouenneProblem * p) [virtual]

empty function to redirect variables to proper variable vector

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.88.4 Member Data Documentation

7.88.4.1 expression** Couenne::exprOp::arglist_ [protected]

argument list is an array of pointers to other expressions

Definition at line 35 of file [CouenneExprOp.hpp](#).

7.88.4.2 int Couenne::exprOp::nargs_ [protected]

number of arguments (cardinality of arglist)

Definition at line 36 of file CouenneExprOp.hpp.

The documentation for this class was generated from the following file:

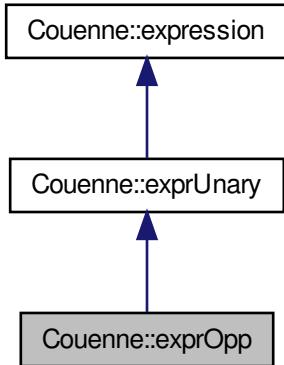
- /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprOp.hpp

7.89 Couenne::exprOpp Class Reference

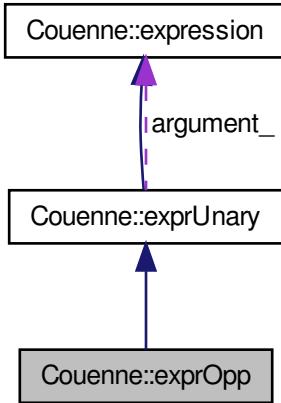
class opposite, $-f(x)$

```
#include <CouenneExprOpp.hpp>
```

Inheritance diagram for Couenne::exprOpp:



Collaboration diagram for Couenne::exprOpp:



Public Member Functions

- **exprOpp (expression *al)**
Constructors, destructor.
- **expression * clone (Domain *d=NULL) const**
cloning method
- **unary_function F ()**
the operator's function
- **void print (std::ostream &out, bool descend) const**
Output.
- **CouNumber gradientNorm (const double *x)**
return L2 norm of gradient at given point
- **expression * differentiate (int index)**
differentiation
- **virtual expression * simplify ()**
simplification
- **int Linearity ()**
get a measure of "how linear" the expression is (see CouenneTypes.h)
- **void getBounds (expression *&, expression *&)**
Get lower and upper bound of an expression (if any)
- **void getBounds (CouNumber &, CouNumber &)**
Get value of lower and upper bound of an expression (if any)
- **virtual void generateCuts (expression *, OsiCuts &, const CouenneCutGenerator *, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)**
special version for linear constraints
- **virtual enum expr_type code ()**
code for comparisons

- bool [isInteger \(\)](#)
is this expression integer?
- bool [impliedBound \(int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ\)](#)
implied bound processing
- [exprAux * standardize \(CouenneProblem *, bool addAux=true\)](#)
standardization (to deal with complex arguments)

Additional Inherited Members

7.89.1 Detailed Description

class opposite, $-f(x)$

Definition at line 27 of file CouenneExprOpp.hpp.

7.89.2 Constructor & Destructor Documentation

7.89.2.1 Couenne::exprOpp::exprOpp (**expression * al**) [inline]

Constructors, destructor.

Definition at line 32 of file CouenneExprOpp.hpp.

7.89.3 Member Function Documentation

7.89.3.1 **expression* Couenne::exprOpp::clone (Domain * d=NULL) const** [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 36 of file CouenneExprOpp.hpp.

7.89.3.2 **unary_function Couenne::exprOpp::F ()** [inline], [virtual]

the operator's function

Reimplemented from [Couenne::exprUnary](#).

Definition at line 40 of file CouenneExprOpp.hpp.

7.89.3.3 **void Couenne::exprOpp::print (std::ostream & out, bool descend) const** [virtual]

Output.

Reimplemented from [Couenne::exprUnary](#).

7.89.3.4 **CouNumber Couenne::exprOpp::gradientNorm (const double * x)** [inline], [virtual]

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Definition at line 48 of file CouenneExprOpp.hpp.

7.89.3.5 `expression* Couenne::exprOpp::differentiate(int index) [virtual]`

differentiation

Reimplemented from [Couenne::expression](#).

7.89.3.6 `virtual expression* Couenne::exprOpp::simplify() [virtual]`

simplification

Reimplemented from [Couenne::exprUnary](#).

7.89.3.7 `int Couenne::exprOpp::Linearity() [inline], [virtual]`

get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::exprUnary](#).

Definition at line 58 of file CouenneExprOpp.hpp.

7.89.3.8 `void Couenne::exprOpp::getBounds(expression *&, expression *&) [virtual]`

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.89.3.9 `void Couenne::exprOpp::getBounds(CouNumber &, CouNumber &) [virtual]`

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.89.3.10 `virtual void Couenne::exprOpp::generateCuts(expression *, OsiCuts &, const CouenneCutGenerator *, t_chg_bounds * = NULL, int s = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]`

special version for linear constraints

Reimplemented from [Couenne::expression](#).

7.89.3.11 `virtual enum expr_type Couenne::exprOpp::code() [inline], [virtual]`

code for comparisons

Reimplemented from [Couenne::exprUnary](#).

Definition at line 75 of file CouenneExprOpp.hpp.

7.89.3.12 `bool Couenne::exprOpp::isInteger() [inline], [virtual]`

is this expression integer?

Reimplemented from [Couenne::exprUnary](#).

Definition at line 79 of file CouenneExprOpp.hpp.

7.89.3.13 `bool Couenne::exprOpp::impliedBound(int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign = expression::AUX_EQ) [virtual]`

implied bound processing

Reimplemented from [Couenne::expression](#).

7.89.3.14 exprAux* Couenne::exprOpp::standardize(CouenneProblem *, bool addAux = true) [virtual]

standardization (to deal with complex arguments)

Reimplemented from [Couenne::exprUnary](#).

The documentation for this class was generated from the following file:

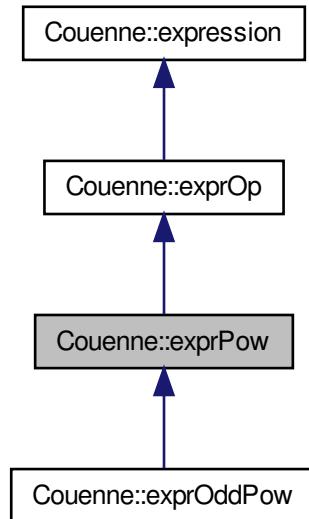
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprOpp.hpp](#)

7.90 Couenne::exprPow Class Reference

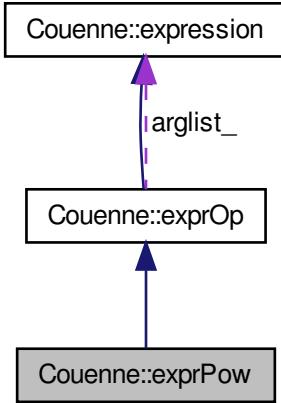
Power of an expression (binary operator), $f(x)^k$ with k constant.

```
#include <CouenneExprPow.hpp>
```

Inheritance diagram for Couenne::exprPow:



Collaboration diagram for Couenne::exprPow:



Public Member Functions

- **exprPow (expression **al, int n=2, bool signpower=false)**
Constructor.
- **exprPow (expression *arg0, expression *arg1, bool signpower=false)**
Constructor with only two arguments.
- **expression * clone (Domain *d=NULL) const**
cloning method
- **virtual std::string printOp () const**
print operator
- **virtual CouNumber operator() ()**
function for the evaluation of the expression
- **virtual CouNumber gradientNorm (const double *x)**
return L2 norm of gradient at given point
- **virtual expression * differentiate (int index)**
differentiation
- **virtual expression * simplify ()**
simplification
- **virtual int Linearity ()**
get a measure of "how linear" the expression is
- **virtual bool isInteger ()**
is this expression integer?
- **virtual void getBounds (expression *&, expression *&)**
Get lower and upper bound of an expression (if any)
- **virtual void getBounds (CouNumber &lb, CouNumber &ub)**
Get value of lower and upper bound of an expression (if any)

- virtual `exprAux * standardize (CouenneProblem *p, bool addAux=true)`
reduce expression in standard form, creating additional aux variables (and constraints)
- virtual void `generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
*generate equality between *this and *w*
- virtual `expression * getFixVar ()`
return an index to the variable's argument that is better fixed in a branching rule for solving a nonconvexity gap
- virtual enum `expr_type code ()`
code for comparison
- virtual bool `impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)`
implied bound processing
- virtual `CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
set up branching object by evaluating many branching points for each expression's arguments
- virtual void `closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right)` const
compute $y^{\{lv\}}$ and $y^{\{uv\}}$ for Violation Transfer algorithm
- virtual bool `isCuttable (CouenneProblem *problem, int index)` const
can this expression be further linearized or are we on its concave ("bad") side
- virtual bool `isSignpower ()` const
return whether this expression corresponds to a signed integer power

Additional Inherited Members

7.90.1 Detailed Description

Power of an expression (binary operator), $f(x)^k$ with k constant.

Definition at line 30 of file CouenneExprPow.hpp.

7.90.2 Constructor & Destructor Documentation

7.90.2.1 Couenne::exprPow::exprPow (`expression ** al, int n = 2, bool signpower = false`) [inline]

Constructor.

Definition at line 40 of file CouenneExprPow.hpp.

7.90.2.2 Couenne::exprPow::exprPow (`expression * arg0, expression * arg1, bool signpower = false`) [inline]

Constructor with only two arguments.

Definition at line 44 of file CouenneExprPow.hpp.

7.90.3 Member Function Documentation

7.90.3.1 `expression* Couenne::exprPow::clone (Domain * d=NULL) const` [inline], [virtual]

cloning method

Reimplemented from `Couenne::expression`.

Reimplemented in [Couenne::exprOddPow](#).

Definition at line 48 of file CouenneExprPow.hpp.

7.90.3.2 virtual std::string Couenne::exprPow::printOp() const [inline], [virtual]

print operator

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprOddPow](#).

Definition at line 52 of file CouenneExprPow.hpp.

7.90.3.3 CouNumber Couenne::exprPow::operator()() [inline], [virtual]

function for the evaluation of the expression

compute power

Implements [Couenne::expression](#).

Reimplemented in [Couenne::exprOddPow](#).

Definition at line 169 of file CouenneExprPow.hpp.

7.90.3.4 virtual CouNumber Couenne::exprPow::gradientNorm(const double *x) [virtual]

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

7.90.3.5 virtual expression* Couenne::exprPow::differentiate(int index) [virtual]

differentiation

Reimplemented from [Couenne::expression](#).

7.90.3.6 virtual expression* Couenne::exprPow::simplify() [virtual]

simplification

Reimplemented from [Couenne::exprOp](#).

7.90.3.7 virtual int Couenne::exprPow::Linearity() [virtual]

get a measure of "how linear" the expression is

Reimplemented from [Couenne::exprOp](#).

7.90.3.8 virtual bool Couenne::exprPow::isInteger() [virtual]

is this expression integer?

Reimplemented from [Couenne::exprOp](#).

7.90.3.9 virtual void Couenne::exprPow::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOddPow](#).

7.90.3.10 virtual void Couenne::exprPow::getBounds (CouNumber & lb, CouNumber & ub) [virtual]

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOddPow](#).

7.90.3.11 virtual exprAux* Couenne::exprPow::standardize (CouenneProblem * p, bool addAux = true) [virtual]

reduce expression in standard form, creating additional aux variables (and constraints)

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprOddPow](#).

7.90.3.12 virtual void Couenne::exprPow::generateCuts (expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOddPow](#).

7.90.3.13 virtual expression* Couenne::exprPow::getFixVar() [inline], [virtual]

return an index to the variable's argument that is better fixed in a branching rule for solving a nonconvexity gap

Reimplemented in [Couenne::exprOddPow](#).

Definition at line 92 of file CouenneExprPow.hpp.

7.90.3.14 virtual enum expr_type Couenne::exprPow::code() [inline], [virtual]

code for comparison

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprOddPow](#).

Definition at line 96 of file CouenneExprPow.hpp.

7.90.3.15 virtual bool Couenne::exprPow::impliedBound (int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [virtual]

implied bound processing

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOddPow](#).

7.90.3.16 virtual CouNumber Couenne::exprPow::selectBranch (const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]

set up branching object by evaluating many branching points for each expression's arguments

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOddPow](#).

7.90.3.17 virtual void Couenne::exprPow::closestFeasible (expression * varind, expression * vardep, CouNumber & left, CouNumber & right) const [virtual]

compute $y^{\{v\}}$ and $y^{\{uv\}}$ for Violation Transfer algorithm

Reimplemented from [Couenne::expression](#).

7.90.3.18 virtual bool Couenne::exprPow::isCuttable (CouenneProblem * problem, int index) const [virtual]

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOddPow](#).

7.90.3.19 virtual bool Couenne::exprPow::isSignpower () const [inline], [virtual]

return whether this expression corresponds to a signed integer power

Definition at line 123 of file [CouenneExprPow.hpp](#).

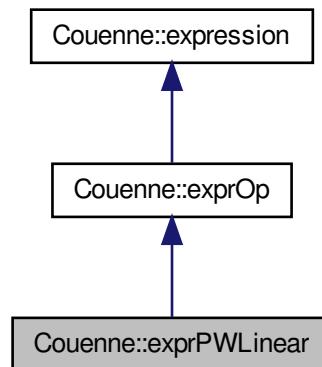
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprPow.hpp](#)

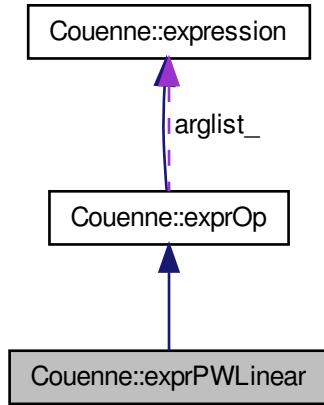
7.91 Couenne::exprPWLinear Class Reference

```
#include <CouenneExprPWLinear.hpp>
```

Inheritance diagram for Couenne::exprPWLinear:



Collaboration diagram for Couenne::exprPWLinear:



Additional Inherited Members

7.91.1 Detailed Description

Definition at line 18 of file CouenneExprPWLinear.hpp.

The documentation for this class was generated from the following file:

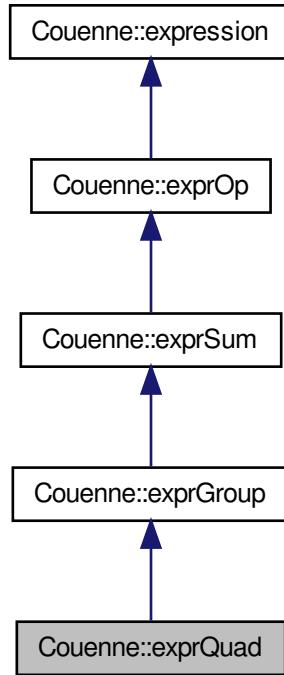
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprPWLinear.hpp](#)

7.92 Couenne::exprQuad Class Reference

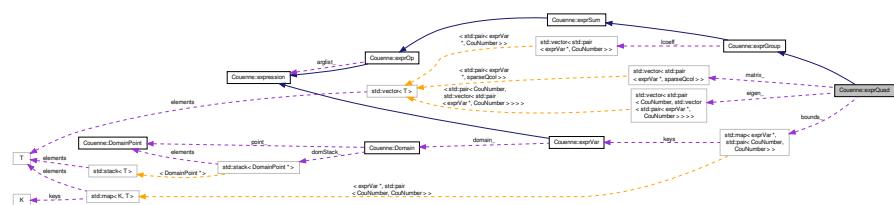
class [exprQuad](#), with constant, linear and quadratic terms

```
#include <CouenneExprQuad.hpp>
```

Inheritance diagram for Couenne::exprQuad:



Collaboration diagram for Couenne::exprQuad:



Public Types

- `typedef std::vector< std::pair<exprVar *, CouNumber > > sparseQcol`
`matrix`
- `typedef std::vector< std::pair<exprVar *, sparseQcol > > sparseQ`

Protected Attributes

Q matrix storage

Sparse implementation: given expression of the form $\sum_{i \in N, j \in N} q_{ij}x_i x_j$, qindexI_ and qindexJ_ contain respectively entries i and j for which q_{ij} is nonzero in $q_{ij}x_i x_j$:

- `sparseQ` matrix_

Convexification data structures

These are filled by alphaConvexify, which implements the alpha-convexification method described in the LaGO paper by Nowak and Vigerske – see also Adjiman and Floudas.

- `std::vector< std::pair< CouNumber, std::vector< std::pair< exprVar *, CouNumber > > > eigen_`
eigenvalues and eigenvectors
- `std::map< exprVar *, std::pair< CouNumber, CouNumber > > bounds_`
current bounds (checked before re-computing eigenvalues/vectors)
- `int nqterms_`
number of non-zeroes in Q
- `exprQuad (CouNumber c0, std::vector< std::pair< exprVar *, CouNumber > > &lcoeff, std::vector< quadElem > &qcoeff, expression **al=NULL, int n=0)`
Constructor.
- `exprQuad (const exprQuad &src, Domain *d=NULL)`
Copy constructor.
- `sparseQ & getQ () const`
- `int getnQTerms ()`
- `virtual expression * clone (Domain *d=NULL) const`
cloning method
- `virtual void print (std::ostream &=std::cout, bool=false) const`
Print expression to an ostream.
- `virtual CouNumber operator() ()`
Function for the evaluation of the expression.
- `CouNumber gradientNorm (const double *x)`
return L2 norm of gradient at given point
- `virtual expression * differentiate (int index)`
Compute derivative of this expression with respect to variable whose index is passed as argument.
- `virtual expression * simplify ()`
Simplify expression.
- `virtual int Linearity ()`
Get a measure of "how linear" the expression is.
- `virtual void getBounds (expression *&, expression *&)`
Get lower and upper bound of an expression (if any)
- `virtual void getBounds (CouNumber &, CouNumber &)`
Get lower and upper bound of an expression (if any)

- virtual void `generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`

Generate cuts for the quadratic expression, which are supporting hyperplanes of the concave upper envelope and the convex lower envelope.
- virtual bool `alphaConvexify (const CouenneProblem *)`

Compute data for α -convexification of a quadratic form (fills in dCoeff_ and dlIndex_ for the convex underestimator)
- void `quadCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg)`

method exprQuad::quadCuts
- virtual int `compare (exprQuad &)`

Compare two exprQuad.
- virtual enum `expr_type code ()`

Code for comparisons.
- virtual int `rank ()`

Used in rank-based branching variable choice.
- virtual bool `isInteger ()`

is this expression integer?
- virtual int `DepList (std::set< int > &depList, enum dig_type type=ORIG_ONLY)`

fill in the set with all indices of variables appearing in the expression
- virtual CouNumber `selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`

Set up branching object by evaluating many branching points for each expression's arguments.
- virtual void `fillDepSet (std::set< DepNode *, compNode > *dep, DepGraph *g)`

Fill dependence set of the expression associated with this auxiliary variable.
- virtual void `replace (exprVar *x, exprVar *w)`

replace variable x with new (aux) w
- virtual void `realign (const CouenneProblem *p)`

replace variable x with new (aux) w
- virtual bool `impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)`

implied bound processing
- CouNumber `computeQBound (int sign)`

method to compute the bound based on sign: -1 for lower, +1 for upper
- virtual void `closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right) const`

compute $y^{\{lv\}}$ and $y^{\{uv\}}$ for Violation Transfer algorithm
- void `computeQuadFiniteBound (CouNumber &qMin, CouNumber &qMax, CouNumber *l, CouNumber *u, int &indInfLo, int &indInfUp)`

return lower and upper bound of quadratic expression
- virtual bool `isCuttable (CouenneProblem *problem, int index) const`

can this expression be further linearized or are we on its concave ("bad") side

Additional Inherited Members

7.92.1 Detailed Description

class `exprQuad`, with constant, linear and quadratic terms

It represents an expression of the form $a_0 + \sum_{i \in I} b_i x_i + x^T Qx + \sum_{i \in J} h_i(x)$, with $a_0 + \sum_{i \in I} b_i x_i$ an affine term, $x^T Qx$ a quadratic term, and a nonlinear sum $\sum_{i \in J} h_i(x)$. Standardization checks possible quadratic or linear terms in the latter and includes them in the former parts.

If $h_i(x)$ is a product of two nonlinear, nonquadratic functions $h'(x)h''(x)$, two auxiliary variables $w' = f'(x)$ and $w'' = h''(x)$ are created and the product $w'w''$ is included in the quadratic part of the [exprQuad](#). If $h(x)$ nonquadratic, nonlinear function, an auxiliary variable $w = h(x)$ is created and included in the linear part.

Definition at line 44 of file CouenneExprQuad.hpp.

7.92.2 Member Typedef Documentation

7.92.2.1 `typedef std::vector<std::pair <exprVar *, CouNumber> > Couenne::exprQuad::sparseQcol`

matrix

Definition at line 49 of file CouenneExprQuad.hpp.

7.92.2.2 `typedef std::vector<std::pair <exprVar *, sparseQcol> > Couenne::exprQuad::sparseQ`

Definition at line 50 of file CouenneExprQuad.hpp.

7.92.3 Constructor & Destructor Documentation

7.92.3.1 `Couenne::exprQuad::exprQuad (CouNumber c0, std::vector< std::pair< exprVar *, CouNumber > > & lcoeff,
std::vector< quadElem > & qcoeff, expression ** al = NULL, int n = 0)`

Constructor.

7.92.3.2 `Couenne::exprQuad::exprQuad (const exprQuad & src, Domain * d = NULL)`

Copy constructor.

7.92.4 Member Function Documentation

7.92.4.1 `sparseQ& Couenne::exprQuad::getQ () const [inline]`

Definition at line 94 of file CouenneExprQuad.hpp.

7.92.4.2 `int Couenne::exprQuad::getnQTerms () [inline]`

Definition at line 97 of file CouenneExprQuad.hpp.

7.92.4.3 `virtual expression* Couenne::exprQuad::clone (Domain * d = NULL) const [inline], [virtual]`

cloning method

Reimplemented from [Couenne::exprGroup](#).

Definition at line 101 of file CouenneExprQuad.hpp.

7.92.4.4 `virtual void Couenne::exprQuad::print (std::ostream & = std::cout, bool = false) const [virtual]`

Print expression to an iostream.

Reimplemented from [Couenne::exprGroup](#).

7.92.4.5 CouNumber Couenne::exprQuad::operator() () [inline], [virtual]

Function for the evaluation of the expression.

Compute sum of linear and nonlinear terms.

Reimplemented from [Couenne::exprGroup](#).

Definition at line 293 of file CouenneExprQuad.hpp.

7.92.4.6 CouNumber Couenne::exprQuad::gradientNorm (const double * x) [virtual]

return l-2 norm of gradient at given point

Reimplemented from [Couenne::exprGroup](#).

7.92.4.7 virtual expression* Couenne::exprQuad::differentiate (int index) [virtual]

Compute derivative of this expression with respect to variable whose index is passed as argument.

Reimplemented from [Couenne::exprGroup](#).

7.92.4.8 virtual expression* Couenne::exprQuad::simplify () [virtual]

Simplify expression.

Reimplemented from [Couenne::exprGroup](#).

7.92.4.9 virtual int Couenne::exprQuad::Linearity () [inline], [virtual]

Get a measure of "how linear" the expression is.

Reimplemented from [Couenne::exprGroup](#).

Definition at line 121 of file CouenneExprQuad.hpp.

7.92.4.10 virtual void Couenne::exprQuad::getBounds (expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::exprGroup](#).

7.92.4.11 virtual void Couenne::exprQuad::getBounds (CouNumber &, CouNumber &) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::exprGroup](#).

7.92.4.12 virtual void Couenne::exprQuad::generateCuts (expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =COUENNE_INFINITY) [virtual]

Generate cuts for the quadratic expression, which are supporting hyperplanes of the concave upper envelope and the convex lower envelope.

Reimplemented from [Couenne::exprGroup](#).

7.92.4.13 virtual bool Couenne::exprQuad::alphaConvexify (const CouenneProblem *) [virtual]

Compute data for α -convexification of a quadratic form (fills in dCoeff_ and dIndex_ for the convex underestimator)

7.92.4.14 void Couenne::exprQuad::quadCuts (expression * w, OsiCuts & cs, const CouenneCutGenerator * cg)

method [exprQuad::quadCuts](#)

Based on the information (dIndex_, dCoeffLo_, dCoeffUp_) created/modified by [alphaConvexify\(\)](#), create convexification cuts for this expression.

The original constraint is :

$$\eta = a_0 + a^T x + x^T Qx$$

where η is the auxiliary corresponding to this expression and w_i are the auxiliaries corresponding to the other non-linear terms contained in the expression.

The under-estimator of $x^T Qx$ is given by

$$x^T Qx + \sum \lambda_{\min,i} (x_i - l_i)(u_i - x_i)$$

and its over-estimator is given by

$$x^T Qx + \sum \lambda_{\max,i} (x_i - l_i)(u_i - x_i)$$

(where $\lambda_{\min,i} = \frac{\lambda_{\min}}{w_i^2}$, $\lambda_{\max,i} = \frac{\lambda_{\max}}{w_i^2}$, and $w_i = u_i - l_i$), where λ_{\max} (λ_{\min}) is the minimum (maximum) eigenvalue of the matrix $A = \text{Diag}(\mathbf{u} - \mathbf{l})Q\text{Diag}(\mathbf{u} - \mathbf{l})$, obtained by pre- and post-multiplying Q by the diagonal matrix whose i -th element is $u_i - l_i$.

Let $\tilde{a}_0(\lambda)$, $\tilde{a}(\lambda)$ and $\tilde{Q}(\lambda)$ be

$$\tilde{a}_0(\lambda) = a_0 - \sum_{i=1}^n \lambda_i l_i u_i$$

$$\tilde{a}(\lambda) = a + \begin{bmatrix} \lambda_1(u_1 + l_1) \\ \vdots \\ \lambda_n(u_n + l_n) \end{bmatrix},$$

$$\tilde{Q}(\lambda) = Q - \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}.$$

The convex relaxation of the initial constraint is then given by the two constraints

$$\eta \geq \tilde{a}_0(\lambda_{\min}) + \tilde{a}(\lambda_{\min})^T x + x^T \tilde{Q}(\lambda_{\min}) x$$

$$\eta \leq \tilde{a}_0(\lambda_{\max}) + \tilde{a}(\lambda_{\max})^T x + x^T \tilde{Q}(\lambda_{\max}) x$$

The cut is computed as follow. Let (x^*, η^*) be the solution at hand. The two outer-approximation cuts are:

$$\eta \geq \tilde{a}_0(\lambda_{\min}) + \tilde{a}(\lambda_{\min})^T x + x^{*T} \tilde{Q}(\lambda_{\min})(2x - x^*)$$

and

$$\eta \leq \tilde{a}_0(\lambda_{\max}) + \tilde{a}(\lambda_{\max})^T x + x^{*T} \tilde{Q}(\lambda_{\max})(2x - x^*);$$

grouping coefficients, we get:

$$x^{*T} \tilde{Q}(\lambda_{\min}) x^* - \tilde{a}_0(\lambda_{\min}) \geq (\tilde{a}(\lambda_{\min}) + 2\tilde{Q}(\lambda_{\min})x^*)^T x - \eta$$

and

$$x^{*T} \tilde{Q}(\lambda_{\max}) x^* - \tilde{a}_0(\lambda_{\max}) \leq (\tilde{a}(\lambda_{\max}) + 2\tilde{Q}(\lambda_{\max})x^*)^T x - \eta$$

7.92.4.15 virtual int Couenne::exprQuad::compare (exprQuad &) [virtual]

Compare two [exprQuad](#).

7.92.4.16 virtual enum expr_type Couenne::exprQuad::code () [inline], [virtual]

Code for comparisons.

Reimplemented from [Couenne::exprGroup](#).

Definition at line 231 of file CouenneExprQuad.hpp.

7.92.4.17 virtual int Couenne::exprQuad::rank () [virtual]

Used in rank-based branching variable choice.

Reimplemented from [Couenne::exprGroup](#).

7.92.4.18 virtual bool Couenne::exprQuad::isInteger () [virtual]

is this expression integer?

Reimplemented from [Couenne::exprGroup](#).

7.92.4.19 virtual int Couenne::exprQuad::DepList (std::set< int > & depList, enum dig_type type = ORIG_ONLY) [virtual]

fill in the set with all indices of variables appearing in the expression

Reimplemented from [Couenne::exprGroup](#).

7.92.4.20 virtual CouNumber Couenne::exprQuad::selectBranch (const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [virtual]

Set up branching object by evaluating many branching points for each expression's arguments.

Reimplemented from [Couenne::expression](#).

7.92.4.21 virtual void Couenne::exprQuad::fillDepSet (std::set< DepNode *, compNode > * dep, DepGraph * g) [virtual]

Fill dependence set of the expression associated with this auxiliary variable.

Reimplemented from [Couenne::exprGroup](#).

7.92.4.22 virtual void Couenne::exprQuad::replace (**exprVar** * *x*, **exprVar** * *w*) [virtual]

replace variable *x* with new (aux) *w*

Reimplemented from [Couenne::exprGroup](#).

7.92.4.23 virtual void Couenne::exprQuad::realign (**const CouenneProblem** * *p*) [virtual]

replace variable *x* with new (aux) *w*

Reimplemented from [Couenne::exprGroup](#).

7.92.4.24 virtual bool Couenne::exprQuad::impliedBound (**int** , **CouNumber** * , **CouNumber** * , **t_chg_bounds** * , **enum auxSign = expression::AUX_EQ**) [virtual]

implied bound processing

Reimplemented from [Couenne::exprSum](#).

7.92.4.25 **CouNumber** Couenne::exprQuad::computeQBound (**int sign**)

method to compute the bound based on sign: -1 for lower, +1 for upper

7.92.4.26 virtual void Couenne::exprQuad::closestFeasible (**expression** * *varind*, **expression** * *vardep*, **CouNumber & left**, **CouNumber & right**) const [virtual]

compute $y^{\{lv\}}$ and $y^{\{uv\}}$ for Violation Transfer algorithm

Reimplemented from [Couenne::expression](#).

7.92.4.27 void Couenne::exprQuad::computeQuadFiniteBound (**CouNumber & qMin**, **CouNumber & qMax**, **CouNumber * l**, **CouNumber * u**, **int & indInfLo**, **int & indInfUp**) [protected]

return lower and upper bound of quadratic expression

7.92.4.28 virtual bool Couenne::exprQuad::isCuttable (**CouenneProblem** * *problem*, **int index**) const [inline], [protected], [virtual]

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

Definition at line 286 of file CouenneExprQuad.hpp.

7.92.5 Member Data Documentation

7.92.5.1 **sparseQ** Couenne::exprQuad::matrix_ [mutable], [protected]

Definition at line 61 of file CouenneExprQuad.hpp.

7.92.5.2 **std::vector<std::pair <CouNumber, std::vector <std::pair <exprVar *, CouNumber> >>**
Couenne::exprQuad::eigen_ [mutable], [protected]

eigenvalues and eigenvectors

Definition at line 73 of file CouenneExprQuad.hpp.

7.92.5.3 std::map<exprVar *, std::pair <CouNumber, CouNumber> > Couenne::exprQuad::bounds_ [protected]

current bounds (checked before re-computing eigenvalues/vectors)

Definition at line 76 of file CouenneExprQuad.hpp.

7.92.5.4 int Couenne::exprQuad::nqterms_ [protected]

number of non-zeroes in Q

Definition at line 79 of file CouenneExprQuad.hpp.

The documentation for this class was generated from the following file:

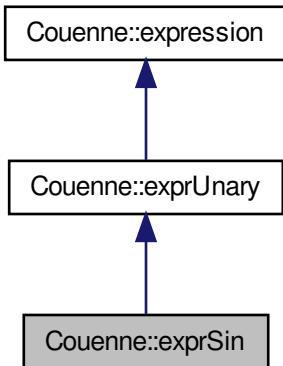
- /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprQuad.hpp

7.93 Couenne::exprSin Class Reference

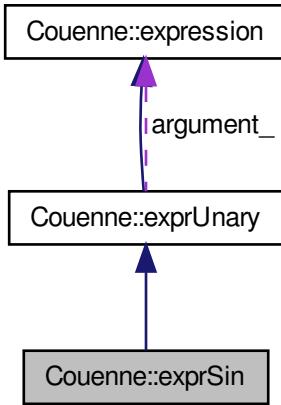
class for $\sin f(x)$

```
#include <CouenneExprSin.hpp>
```

Inheritance diagram for Couenne::exprSin:



Collaboration diagram for Couenne::exprSin:



Public Member Functions

- **`exprSin (expression *al)`**
Constructors, destructor.
- **`expression * clone (Domain *d=NULL) const`**
cloning method
- **`unary_function F ()`**
the operator itself (e.g. sin, log...)
- **`std::string printOp () const`**
print operator
- **`CouNumber gradientNorm (const double *x)`**
return L-2 norm of gradient at given point
- **`expression * differentiate (int index)`**
differentiation
- **`void getBounds (expression *&, expression *&)`**
Get lower and upper bound of an expression (if any)
- **`void getBounds (CouNumber &lb, CouNumber &ub)`**
Get value of lower and upper bound of an expression (if any)
- **`void generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`**
*generate equality between *this and *w*
- **`virtual enum expr_type code ()`**
code for comparisons
- **`bool impliedBound (int index, CouNumber *l, CouNumber *u, t_chg_bounds *chg, enum auxSign=expression::AUX_EQ)`**
implied bound processing

- virtual `CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
Set up branching object by evaluating many branching points for each expression's arguments.
- virtual void `closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right)` const
closest feasible points in function in both directions
- virtual bool `isCuttable (CouenneProblem *problem, int index)` const
can this expression be further linearized or are we on its concave ("bad") side

Additional Inherited Members

7.93.1 Detailed Description

class for $\sin f(x)$

Definition at line 47 of file CouenneExprSin.hpp.

7.93.2 Constructor & Destructor Documentation

7.93.2.1 Couenne::exprSin::exprSin (`expression * al`) [inline]

Constructors, destructor.

Definition at line 52 of file CouenneExprSin.hpp.

7.93.3 Member Function Documentation

7.93.3.1 `expression* Couenne::exprSin::clone (Domain * d = NULL) const` [inline], [virtual]

cloning method

Reimplemented from `Couenne::expression`.

Definition at line 56 of file CouenneExprSin.hpp.

7.93.3.2 `unary_function Couenne::exprSin::F ()` [inline], [virtual]

the operator itself (e.g. sin, log...)

Reimplemented from `Couenne::exprUnary`.

Definition at line 60 of file CouenneExprSin.hpp.

7.93.3.3 `std::string Couenne::exprSin::printOp () const` [inline], [virtual]

print operator

Reimplemented from `Couenne::exprUnary`.

Definition at line 64 of file CouenneExprSin.hpp.

7.93.3.4 `CouNumber Couenne::exprSin::gradientNorm (const double * x)` [inline], [virtual]

return L2 norm of gradient at given point

Reimplemented from `Couenne::expression`.

Definition at line 68 of file CouenneExprSin.hpp.

7.93.3.5 `expression* Couenne::exprSin::differentiate(int index) [virtual]`

differentiation

Reimplemented from [Couenne::expression](#).

7.93.3.6 `void Couenne::exprSin::getBounds(expression *&, expression *&) [virtual]`

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.93.3.7 `void Couenne::exprSin::getBounds(CouNumber & lb, CouNumber & ub) [virtual]`

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.93.3.8 `void Couenne::exprSin::generateCuts(expression * w, OsiCuts & cs, const CouenneCutGenerator * cg, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =COUENNE_INFINITY) [virtual]`

generate equality between *this and *w

Reimplemented from [Couenne::expression](#).

7.93.3.9 `virtual enum expr_type Couenne::exprSin::code() [inline], [virtual]`

code for comparisons

Reimplemented from [Couenne::exprUnary](#).

Definition at line 90 of file CouenneExprSin.hpp.

7.93.3.10 `bool Couenne::exprSin::impliedBound(int index, CouNumber * l, CouNumber * u, t_chg_bounds * chg, enum auxSign = expression::AUX_EQ) [inline], [virtual]`

implied bound processing

Reimplemented from [Couenne::expression](#).

Definition at line 94 of file CouenneExprSin.hpp.

7.93.3.11 `virtual CouNumber Couenne::exprSin::selectBranch(const CouenneObject * obj, const OsiBranchingInformation * info, expression *& var, double *& brpts, double *& brDist, int & way) [inline], [virtual]`

Set up branching object by evaluating many branching points for each expression's arguments.

Reimplemented from [Couenne::expression](#).

Definition at line 111 of file CouenneExprSin.hpp.

7.93.3.12 `virtual void Couenne::exprSin::closestFeasible(expression * varind, expression * vardep, CouNumber & left, CouNumber & right) const [virtual]`

closest feasible points in function in both directions

Reimplemented from [Couenne::expression](#).

```
7.93.3.13 virtual bool Couenne::exprSin::isCuttable ( CouenneProblem * problem, int index ) const [inline],  
[virtual]
```

can this expression be further linearized or are we on its concave ("bad") side

Reimplemented from [Couenne::expression](#).

Definition at line 127 of file CouenneExprSin.hpp.

The documentation for this class was generated from the following file:

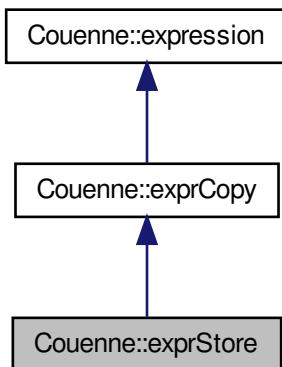
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprSin.hpp](#)

7.94 Couenne::exprStore Class Reference

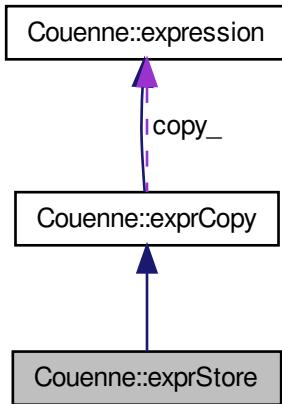
storage class for previously evaluated expressions

```
#include <CouenneExprStore.hpp>
```

Inheritance diagram for Couenne::exprStore:



Collaboration diagram for Couenne::exprStore:



Public Member Functions

- `exprStore (expression *copy)`
Constructor.
- `exprStore (const exprStore &e, Domain *d=NULL)`
Store constructor – Must go.
- `virtual ~exprStore ()`
Destructor.
- `virtual void print (std::ostream &out=std::cout, bool descend=false) const`
Printing.
- `virtual expression * clone (Domain *d=NULL) const`
Cloning method.
- `virtual CouNumber operator() ()`
function for evaluating the expression – returns value of `exprCopy` pointed to, which returns a value stored from a previous evaluation

Protected Attributes

- `CouNumber value_`
Value of the (previously evaluated) expression.

Additional Inherited Members

7.94.1 Detailed Description

storage class for previously evaluated expressions

Definition at line 23 of file CouenneExprStore.hpp.

7.94.2 Constructor & Destructor Documentation

7.94.2.1 `Couenne::exprStore::exprStore(expression * copy) [inline]`

Constructor.

Definition at line 33 of file CouenneExprStore.hpp.

7.94.2.2 `Couenne::exprStore::exprStore(const exprStore & e, Domain * d=NULL) [inline]`

Store constructor – Must go.

Definition at line 37 of file CouenneExprStore.hpp.

7.94.2.3 `virtual Couenne::exprStore::~exprStore() [inline], [virtual]`

Destructor.

Definition at line 43 of file CouenneExprStore.hpp.

7.94.3 Member Function Documentation

7.94.3.1 `virtual void Couenne::exprStore::print(std::ostream & out = std::cout, bool descend = false) const [virtual]`

Printing.

Reimplemented from [Couenne::exprCopy](#).

7.94.3.2 `virtual expression* Couenne::exprStore::clone(Domain * d=NULL) const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::exprCopy](#).

Definition at line 51 of file CouenneExprStore.hpp.

7.94.3.3 `virtual CouNumber Couenne::exprStore::operator()() [inline], [virtual]`

function for evaluating the expression – returns value of [exprCopy](#) pointed to, which returns a value stored from a previous evaluation

Reimplemented from [Couenne::exprCopy](#).

Definition at line 57 of file CouenneExprStore.hpp.

7.94.4 Member Data Documentation

7.94.4.1 `CouNumber Couenne::exprStore::value_ [protected]`

Value of the (previously evaluated) expression.

Definition at line 28 of file CouenneExprStore.hpp.

The documentation for this class was generated from the following file:

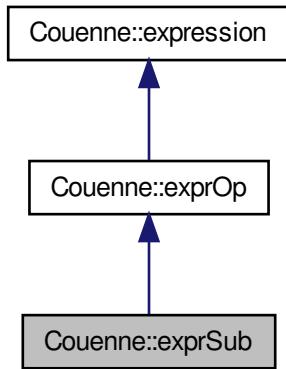
- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprStore.hpp](#)

7.95 Couenne::exprSub Class Reference

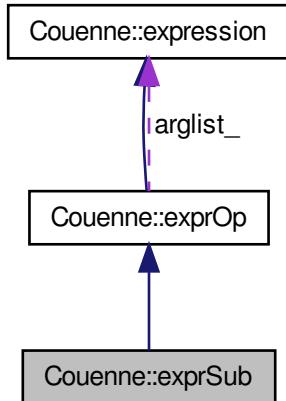
class for subtraction, $f(x) - g(x)$

```
#include <CouenneExprSub.hpp>
```

Inheritance diagram for Couenne::exprSub:



Collaboration diagram for Couenne::exprSub:



Public Member Functions

- [exprSub \(expression **al, int n=2\)](#)

- Constructor.*
- `exprSub (expression *arg0, expression *arg1)`

Constructor with two explicit elements.
 - `expression * clone (Domain *d=NULL) const`

Cloning method.
 - `std::string printOp () const`

print operator
 - `CouNumber operator() ()`

Function for the evaluation of the difference.
 - `expression * differentiate (int index)`

Differentiation.
 - `expression * simplify ()`

Simplification.
 - `virtual int Linearity ()`

Get a measure of "how linear" the expression is (see CouenneTypes.h)
 - `void getBounds (expression *&, expression *&)`

Get lower and upper bound of an expression (if any)
 - `void getBounds (CouNumber &lb, CouNumber &ub)`

Get value of lower and upper bound of an expression (if any)
 - `virtual exprAux * standardize (CouenneProblem *p, bool addAux=true)`

Reduce expression in standard form, creating additional aux variables (and constraints)
 - `virtual void generateCuts (expression *, OsiCuts &, const CouenneCutGenerator *, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`

Special version for linear constraints.
 - `virtual enum expr_type code ()`

Code for comparisons.
 - `bool impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)`

Implied bound processing.

Additional Inherited Members

7.95.1 Detailed Description

class for subtraction, $f(x) - g(x)$

Definition at line 22 of file CouenneExprSub.hpp.

7.95.2 Constructor & Destructor Documentation

7.95.2.1 Couenne::exprSub::exprSub (`expression ** al, int n = 2`) [inline]

Constructor.

Definition at line 27 of file CouenneExprSub.hpp.

7.95.2.2 Couenne::exprSub::exprSub (`expression * arg0, expression * arg1`) [inline]

Constructor with two explicit elements.

Definition at line 31 of file CouenneExprSub.hpp.

7.95.3 Member Function Documentation

7.95.3.1 `expression* Couenne::exprSub::clone(Domain * d = NULL) const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::expression](#).

Definition at line 35 of file CouenneExprSub.hpp.

7.95.3.2 `std::string Couenne::exprSub::printOp() const [inline], [virtual]`

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 39 of file CouenneExprSub.hpp.

7.95.3.3 `CouNumber Couenne::exprSub::operator()() [inline], [virtual]`

Function for the evaluation of the difference.

Compute difference.

Implements [Couenne::expression](#).

Definition at line 88 of file CouenneExprSub.hpp.

7.95.3.4 `expression* Couenne::exprSub::differentiate(int index) [virtual]`

Differentiation.

Reimplemented from [Couenne::expression](#).

7.95.3.5 `expression* Couenne::exprSub::simplify() [virtual]`

Simplification.

Reimplemented from [Couenne::exprOp](#).

7.95.3.6 `virtual int Couenne::exprSub::Linearity() [inline], [virtual]`

Get a measure of "how linear" the expression is (see CouenneTypes.h)

Reimplemented from [Couenne::exprOp](#).

Definition at line 52 of file CouenneExprSub.hpp.

7.95.3.7 `void Couenne::exprSub::getBounds(expression *&, expression *&) [virtual]`

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.95.3.8 `void Couenne::exprSub::getBounds(CouNumber & lb, CouNumber & ub) [virtual]`

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.95.3.9 `virtual exprAux* Couenne::exprSub::standardize(CouenneProblem * p, bool addAux = true) [virtual]`

Reduce expression in standard form, creating additional aux variables (and constraints)

Reimplemented from [Couenne::exprOp](#).

7.95.3.10 virtual void Couenne::exprSub::generateCuts (expression *, OsiCuts & , const CouenneCutGenerator * , t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =COUENNE_INFINITY) [virtual]

Special version for linear constraints.

Reimplemented from [Couenne::expression](#).

7.95.3.11 virtual enum expr_type Couenne::exprSub::code () [inline], [virtual]

Code for comparisons.

Reimplemented from [Couenne::exprOp](#).

Definition at line 79 of file CouenneExprSub.hpp.

7.95.3.12 bool Couenne::exprSub::impliedBound(int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [virtual]

Implied bound processing.

Reimplemented from [Couenne::expression](#).

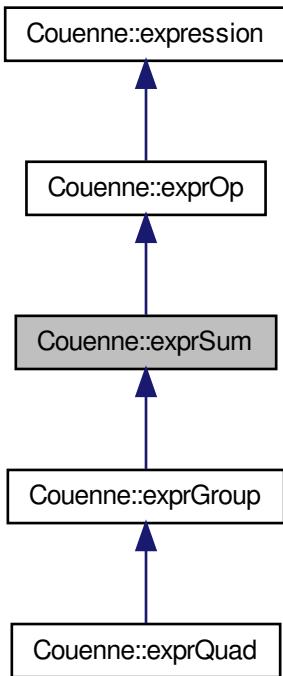
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprSub.hpp](#)

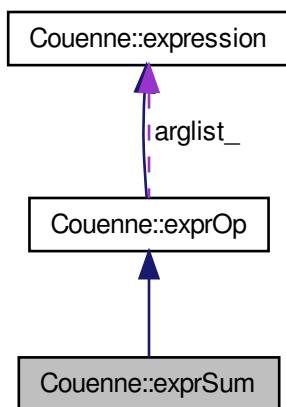
7.96 Couenne::exprSum Class Reference

```
class sum,  $\sum_{i=1}^n f_i(x)$ 
#include <CouenneExprSum.hpp>
```

Inheritance diagram for Couenne::exprSum:



Collaboration diagram for Couenne::exprSum:



Public Member Functions

- **exprSum (expression **=NULL, int=0)**
Constructors, destructor.
- **exprSum (expression *, expression *)**
Constructor with two elements.
- **virtual ~exprSum ()**
Empty destructor.
- **virtual expression * clone (Domain *d=NULL) const**
Cloning method.
- **std::string printOp () const**
Print operator.
- **virtual CouNumber operator() ()**
Function for the evaluation of the expression.
- **virtual expression * differentiate (int index)**
Differentiation.
- **virtual expression * simplify ()**
Simplification.
- **virtual int Linearity ()**
Get a measure of "how linear" the expression is:
- **virtual void getBounds (expression *&, expression *&)**
Get lower and upper bound of an expression (if any)
- **virtual void getBounds (CouNumber &, CouNumber &)**
Get lower and upper bound of an expression (if any)
- **virtual exprAux * standardize (CouenneProblem *p, bool addAux=true)**
Reduce expression in standard form, creating additional aux variables (and constraints)
- **virtual void generateCuts (expression *, OsiCuts &, const CouenneCutGenerator *, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)**
Special version for linear constraints.
- **virtual enum expr_type code ()**
Code for comparison.
- **virtual bool impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)**
Implied bound.
- **exprAux * createQuadratic (CouenneProblem *)**
Checks for quadratic terms in the expression and returns an `exprQuad` if there are enough to create something that can be convexified.

Protected Member Functions

- **int impliedBoundSum (CouNumber wl, CouNumber wu, std::vector< CouNumber > &xl, std::vector< CouNumber > &xu, std::vector< std::pair< int, CouNumber > > &nl, std::vector< std::pair< int, CouNumber > > &nu)**
inferring bounds on factors of a product

Additional Inherited Members

7.96.1 Detailed Description

class sum, $\sum_{i=1}^n f_i(x)$

Definition at line 22 of file CouenneExprSum.hpp.

7.96.2 Constructor & Destructor Documentation

7.96.2.1 Couenne::exprSum::exprSum (**expression** ** *=NULL*, **int** = 0)

Constructors, destructor.

7.96.2.2 Couenne::exprSum::exprSum (**expression** * , **expression** *)

Constructor with two elements.

7.96.2.3 virtual Couenne::exprSum::~exprSum () [inline], [virtual]

Empty destructor.

Definition at line 33 of file CouenneExprSum.hpp.

7.96.3 Member Function Documentation

7.96.3.1 virtual **expression*** Couenne::exprSum::clone (**Domain** * *d=NULL*) const [inline], [virtual]

Cloning method.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

Definition at line 36 of file CouenneExprSum.hpp.

7.96.3.2 std::string Couenne::exprSum::printOp () const [inline], [virtual]

Print operator.

Reimplemented from [Couenne::exprOp](#).

Definition at line 40 of file CouenneExprSum.hpp.

7.96.3.3 CouNumber Couenne::exprSum::operator() () [inline], [virtual]

Function for the evaluation of the expression.

compute sum

Implements [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

Definition at line 118 of file CouenneExprSum.hpp.

7.96.3.4 virtual **expression*** Couenne::exprSum::differentiate (**int** *index*) [virtual]

Differentiation.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.96.3.5 virtual expression* Couenne::exprSum::simplify() [virtual]

Simplification.

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.96.3.6 virtual int Couenne::exprSum::Linearity() [virtual]

Get a measure of "how linear" the expression is:

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.96.3.7 virtual void Couenne::exprSum::getBounds(expression *&, expression *&) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.96.3.8 virtual void Couenne::exprSum::getBounds(CouNumber &, CouNumber &) [virtual]

Get lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.96.3.9 virtual exprAux* Couenne::exprSum::standardize(CouenneProblem * p, bool addAux = true) [virtual]

Reduce expression in standard form, creating additional aux variables (and constraints)

Reimplemented from [Couenne::exprOp](#).

7.96.3.10 virtual void Couenne::exprSum::generateCuts(expression * , OsiCuts & , const CouenneCutGenerator * , t_chg_bounds * = NULL, int = -1, CouNumber = -COUENNE_INFINITY, CouNumber = COUENNE_INFINITY) [virtual]

Special version for linear constraints.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

7.96.3.11 virtual enum expr_type Couenne::exprSum::code() [inline], [virtual]

Code for comparison.

Reimplemented from [Couenne::exprOp](#).

Reimplemented in [Couenne::exprQuad](#), and [Couenne::exprGroup](#).

Definition at line 73 of file CouenneExprSum.hpp.

7.96.3.12 virtual bool Couenne::exprSum::impliedBound (int , CouNumber * , CouNumber * , t_chg_bounds * , enum auxSign = expression::AUX_EQ) [virtual]

Implied bound.

An expression

$$w = a0 + \sum_{i \in I1} a_i x_i + \sum_{i \in I2} a_i x_i$$

is given such that all a_i are positive for $i \in I1$ and negative for $i \in I2$. If the bounds on $w \in [l, u]$, implied bounds on all $x_i, i \in I1 \cup I2$ are as follows:

$$\forall i \in I1 \quad x_i \geq (l - a0 - \sum_{j \in I1 | j \neq i} a_j u_j - \sum_{j \in I2} a_j l_j) / a_i \quad x_i \leq (u - a0 - \sum_{j \in I1 | j \neq i} a_j l_j - \sum_{j \in I2} a_j u_j) / a_i$$

$$\forall i \in I2 \quad x_i \geq (u - a0 - \sum_{j \in I1} a_j u_j - \sum_{j \in I2 | j \neq i} a_j l_j) / a_i \quad x_i \leq (l - a0 - \sum_{j \in I1} a_j l_j - \sum_{j \in I2 | j \neq i} a_j u_j) / a_i,$$

where l_i and u_i are lower and upper bound, respectively, of x_i . We also have to check if some of these bounds are infinite.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprQuad](#).

7.96.3.13 exprAux* Couenne::exprSum::createQuadratic (CouenneProblem *)

Checks for quadratic terms in the expression and returns an [exprQuad](#) if there are enough to create something that can be convexified.

7.96.3.14 int Couenne::exprSum::impliedBoundSum (CouNumber wl, CouNumber wu, std::vector< CouNumber > & xl, std::vector< CouNumber > & xu, std::vector< std::pair< int, CouNumber > > & nl, std::vector< std::pair< int, CouNumber > > & nu) [protected]

inferring bounds on factors of a product

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprSum.hpp](#)

7.97 Couenne::exprTrilinear Class Reference

class for multiplications

```
#include <CouenneExprTrilinear.hpp>
```

Public Member Functions

- [exprTrilinear \(expression **, int\)](#)
Constructor.
- [exprTrilinear \(expression *, expression *, expression *\)](#)
Constructor with two arguments.
- [expression * clone \(Domain *d=NULL\) const](#)
Cloning method.
- [CouNumber gradientNorm \(const double *x\)](#)
return L2 norm of gradient at given point
- [virtual void getBounds \(expression *&, expression *&\)](#)
Get lower and upper bound of an expression (if any)
- [virtual void getBounds \(CouNumber &lb, CouNumber &ub\)](#)

- Get value of lower and upper bound of an expression (if any)*
- void `generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
 - generate equality between *this and *w*
 - virtual enum `expr_type code ()`
 - code for comparison*
 - bool `impliedBound (int, CouNumber *, CouNumber *, t_chg_bounds *, enum Couenne::expression::auxSign=Couenne::expression::AUX_EQ)`
 - implied bound processing*
 - virtual `CouNumber selectBranch (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way)`
 - set up branching object by evaluating many branching points for each expression's arguments*
 - virtual void `closestFeasible (expression *varind, expression *vardep, CouNumber &left, CouNumber &right)`
 - const
 - compute y^{lv} and y^{uv} for Violation Transfer algorithm*

7.97.1 Detailed Description

class for multiplications

Definition at line 21 of file CouenneExprTrilinear.hpp.

7.97.2 Constructor & Destructor Documentation

7.97.2.1 Couenne::exprTrilinear::exprTrilinear (`expression **, int`)

Constructor.

7.97.2.2 Couenne::exprTrilinear::exprTrilinear (`expression *, expression *, expression *`)

Constructor with two arguments.

7.97.3 Member Function Documentation

7.97.3.1 `expression* Couenne::exprTrilinear::clone (Domain * d=NULL) const [inline]`

Cloning method.

Definition at line 32 of file CouenneExprTrilinear.hpp.

7.97.3.2 `CouNumber Couenne::exprTrilinear::gradientNorm (const double * x)`

return l-2 norm of gradient at given point

7.97.3.3 `virtual void Couenne::exprTrilinear::getBounds (expression *&, expression *&) [virtual]`

Get lower and upper bound of an expression (if any)

7.97.3.4 `virtual void Couenne::exprTrilinear::getBounds (CouNumber & lb, CouNumber & ub) [virtual]`

Get value of lower and upper bound of an expression (if any)

7.97.3.5 void Couenne::exprTrilinear::generateCuts (*expression* * *w*, OsiCuts & *cs*, const CouenneCutGenerator * *cg*, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =COUENNE_INFINITY)

generate equality between *this and *w

7.97.3.6 virtual enum expr_type Couenne::exprTrilinear::code () [inline], [virtual]

code for comparison

Definition at line 52 of file CouenneExprTrilinear.hpp.

7.97.3.7 bool Couenne::exprTrilinear::impliedBound (int , CouNumber * , CouNumber * , t_chg_bounds * , enum Couenne::expression::auxSign =Couenne::expression::AUX_EQ)

implied bound processing

7.97.3.8 virtual CouNumber Couenne::exprTrilinear::selectBranch (const CouenneObject * *obj*, const OsiBranchingInformation * *info*, expression *& *var*, double *& *brpts*, double *& *brDist*, int & *way*) [virtual]

set up branching object by evaluating many branching points for each expression's arguments

7.97.3.9 virtual void Couenne::exprTrilinear::closestFeasible (expression * *varind*, expression * *vardep*, CouNumber & *left*, CouNumber & *right*) const [virtual]

compute y^{lv} and y^{uv} for Violation Transfer algorithm

The documentation for this class was generated from the following file:

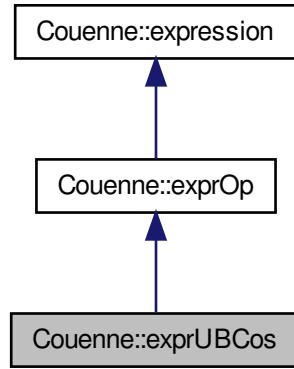
- /home/ted/COIN/trunk/Couenne/src/expression/operators/[CouenneExprTrilinear.hpp](#)

7.98 Couenne::exprUBCos Class Reference

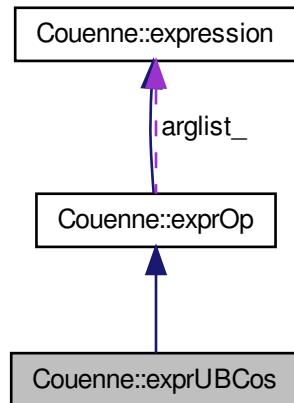
class to compute lower bound of a cosine based on the bounds of its arguments

```
#include <CouenneExprBCos.hpp>
```

Inheritance diagram for Couenne::exprUBCos:



Collaboration diagram for Couenne::exprUBCos:



Public Member Functions

- [exprUBCos \(expression *lb, expression *ub\)](#)
Constructors, destructor.
- [expression * clone \(Domain *d=NULL\) const](#)
cloning method
- [CouNumber operator\(\) \(\)](#)

- `std::string printOp () const`
print operator
- `enum pos printPos () const`
print position (PRE, INSIDE, POST)

Additional Inherited Members

7.98.1 Detailed Description

class to compute lower bound of a cosine based on the bounds of its arguments

Definition at line 80 of file CouenneExprBCos.hpp.

7.98.2 Constructor & Destructor Documentation

7.98.2.1 Couenne::exprUBCos::exprUBCos (`expression * lb, expression * ub`) [inline]

Constructors, destructor.

Definition at line 85 of file CouenneExprBCos.hpp.

7.98.3 Member Function Documentation

7.98.3.1 `expression* Couenne::exprUBCos::clone (Domain * d = NULL) const` [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 92 of file CouenneExprBCos.hpp.

7.98.3.2 `CouNumber Couenne::exprUBCos::operator() ()` [inline], [virtual]

function for the evaluation of the expression

compute sum

Implements [Couenne::expression](#).

Definition at line 111 of file CouenneExprBCos.hpp.

7.98.3.3 `std::string Couenne::exprUBCos::printOp () const` [inline], [virtual]

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 100 of file CouenneExprBCos.hpp.

7.98.3.4 `enum pos Couenne::exprUBCos::printPos () const` [inline], [virtual]

print position (PRE, INSIDE, POST)

Reimplemented from [Couenne::exprOp](#).

Definition at line 104 of file CouenneExprBCos.hpp.

The documentation for this class was generated from the following file:

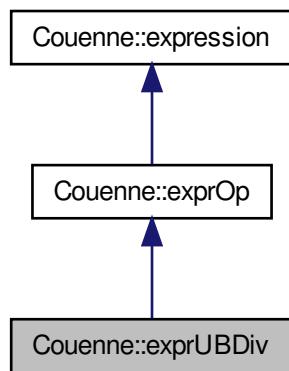
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBCos.hpp

7.99 Couenne::exprUBDiv Class Reference

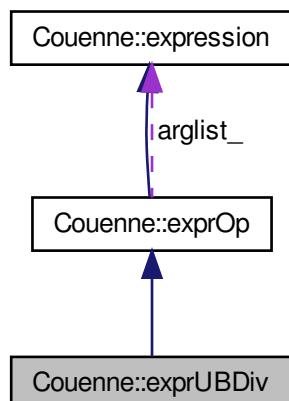
class to compute upper bound of a fraction based on the bounds of both numerator and denominator

```
#include <CouenneExprBDiv.hpp>
```

Inheritance diagram for Couenne::exprUBDiv:



Collaboration diagram for Couenne::exprUBDiv:



Public Member Functions

- **exprUBDiv (expression **al, int n)**
Constructors, destructor.
- **expression * clone (Domain *d=NULL) const**
cloning method
- **CouNumber operator() ()**
function for the evaluation of the expression
- **enum pos printPos () const**
print position (PRE, INSIDE, POST)
- **std::string printOp () const**
print operator

Additional Inherited Members

7.99.1 Detailed Description

class to compute upper bound of a fraction based on the bounds of both numerator and denominator
Definition at line 85 of file CouenneExprBDiv.hpp.

7.99.2 Constructor & Destructor Documentation

7.99.2.1 Couenne::exprUBDiv::exprUBDiv (expression ** al, int n) [inline]

Constructors, destructor.

Definition at line 90 of file CouenneExprBDiv.hpp.

7.99.3 Member Function Documentation

7.99.3.1 expression* Couenne::exprUBDiv::clone (Domain * d=NULL) const [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 94 of file CouenneExprBDiv.hpp.

7.99.3.2 CouNumber Couenne::exprUBDiv::operator() () [inline], [virtual]

function for the evaluation of the expression

compute sum

Implements [Couenne::expression](#).

Definition at line 112 of file CouenneExprBDiv.hpp.

7.99.3.3 enum pos Couenne::exprUBDiv::printPos () const [inline], [virtual]

print position (PRE, INSIDE, POST)

Reimplemented from [Couenne::exprOp](#).

Definition at line 101 of file CouenneExprBDiv.hpp.

7.99.3.4 std::string Couenne::exprUBDiv::printOp () const [inline], [virtual]

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 105 of file [CouenneExprBDiv.hpp](#).

The documentation for this class was generated from the following file:

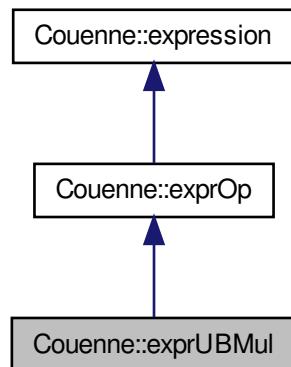
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/[CouenneExprBDiv.hpp](#)

7.100 Couenne::exprUBMul Class Reference

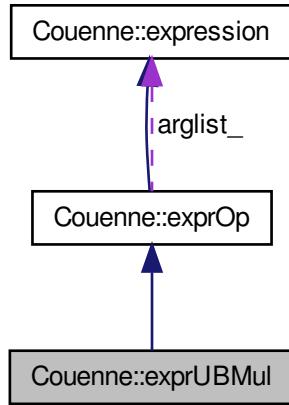
class to compute upper bound of a product based on the bounds of both factors

```
#include <CouenneExprBMul.hpp>
```

Inheritance diagram for Couenne::exprUBMul:



Collaboration diagram for Couenne::exprUBMul:



Public Member Functions

- **exprUBMul (expression **al, int n)**
Constructors, destructor.
- **expression * clone (Domain *d=NULL) const**
cloning method
- **CouNumber operator() ()**
function for the evaluation of the expression
- enum **pos printPos () const**
print position (PRE, INSIDE, POST)
- std::string **printOp () const**
print operator

Additional Inherited Members

7.100.1 Detailed Description

class to compute upper bound of a product based on the bounds of both factors

Definition at line 93 of file CouenneExprBMul.hpp.

7.100.2 Constructor & Destructor Documentation

7.100.2.1 Couenne::exprUBMul::exprUBMul (expression ** al, int n) [inline]

Constructors, destructor.

Definition at line 98 of file CouenneExprBMul.hpp.

7.100.3 Member Function Documentation

7.100.3.1 **expression* Couenne::exprUBMul::clone (Domain * d= NULL) const [inline], [virtual]**

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 102 of file CouenneExprBMul.hpp.

7.100.3.2 **CouNumber Couenne::exprUBMul::operator() () [inline], [virtual]**

function for the evaluation of the expression

compute sum

Implements [Couenne::expression](#).

Definition at line 120 of file CouenneExprBMul.hpp.

7.100.3.3 **enum pos Couenne::exprUBMul::printPos () const [inline], [virtual]**

print position (PRE, INSIDE, POST)

Reimplemented from [Couenne::exprOp](#).

Definition at line 109 of file CouenneExprBMul.hpp.

7.100.3.4 **std::string Couenne::exprUBMul::printOp () const [inline], [virtual]**

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 113 of file CouenneExprBMul.hpp.

The documentation for this class was generated from the following file:

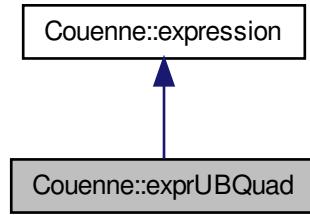
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/[CouenneExprBMul.hpp](#)

7.101 Couenne::exprUBQuad Class Reference

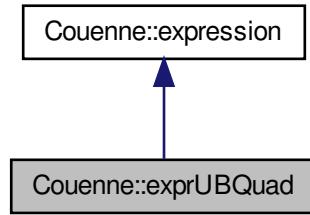
class to compute upper bound of a fraction based on the bounds of both numerator and denominator

```
#include <CouenneExprBQuad.hpp>
```

Inheritance diagram for Couenne::exprUBQuad:



Collaboration diagram for Couenne::exprUBQuad:



Public Member Functions

- [exprUBQuad \(exprQuad *ref\)](#)
Constructor.
- [exprUBQuad \(const exprUBQuad &src, Domain *d=NULL\)](#)
copy constructor
- [~exprUBQuad \(\)](#)
destructor
- [expression * clone \(Domain *d=NULL\) const](#)
cloning method
- [CouNumber operator\(\) \(\)](#)
function for the evaluation of the expression
- [virtual void print \(std::ostream &s=std::cout, bool descend=false\) const](#)
I/O.

Additional Inherited Members

7.101.1 Detailed Description

class to compute upper bound of a fraction based on the bounds of both numerator and denominator

Definition at line 60 of file CouenneExprBQuad.hpp.

7.101.2 Constructor & Destructor Documentation

7.101.2.1 Couenne::exprUBQuad::exprUBQuad (exprQuad * ref) [inline]

Constructor.

Definition at line 67 of file CouenneExprBQuad.hpp.

7.101.2.2 Couenne::exprUBQuad::exprUBQuad (const exprUBQuad & src, Domain * d = NULL) [inline]

copy constructor

Definition at line 71 of file CouenneExprBQuad.hpp.

7.101.2.3 Couenne::exprUBQuad::~exprUBQuad () [inline]

destructor

Definition at line 77 of file CouenneExprBQuad.hpp.

7.101.3 Member Function Documentation

7.101.3.1 expression* Couenne::exprUBQuad::clone (Domain * d = NULL) const [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 80 of file CouenneExprBQuad.hpp.

7.101.3.2 CouNumber Couenne::exprUBQuad::operator() () [inline], [virtual]

function for the evaluation of the expression

Implements [Couenne::expression](#).

Definition at line 84 of file CouenneExprBQuad.hpp.

7.101.3.3 virtual void Couenne::exprUBQuad::print (std::ostream & s = std::cout, bool descend = false) const [inline], [virtual]

I/O.

Reimplemented from [Couenne::expression](#).

Definition at line 88 of file CouenneExprBQuad.hpp.

The documentation for this class was generated from the following file:

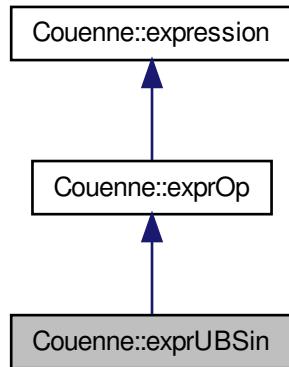
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/[CouenneExprBQuad.hpp](#)

7.102 Couenne::exprUBSin Class Reference

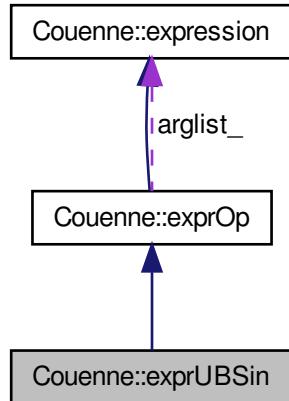
class to compute lower bound of a sine based on the bounds on its arguments

```
#include <CouenneExprBSin.hpp>
```

Inheritance diagram for Couenne::exprUBSin:



Collaboration diagram for Couenne::exprUBSin:



Public Member Functions

- [exprUBSin \(expression *lb, expression *ub\)](#)

Constructors, destructor.

- `expression * clone (Domain *d=NULL) const`
cloning method
- `CouNumber operator() ()`
function for the evaluation of the expression
- `std::string printOp () const`
print operator
- `enum pos printPos () const`
print position (PRE, INSIDE, POST)

Additional Inherited Members

7.102.1 Detailed Description

class to compute lower bound of a sine based on the bounds on its arguments

Definition at line 80 of file CouenneExprBSin.hpp.

7.102.2 Constructor & Destructor Documentation

7.102.2.1 Couenne::exprUBSin::exprUBSin (`expression * lb, expression * ub`) [inline]

Constructors, destructor.

Definition at line 85 of file CouenneExprBSin.hpp.

7.102.3 Member Function Documentation

7.102.3.1 `expression* Couenne::exprUBSin::clone (Domain * d = NULL) const` [inline], [virtual]

cloning method

Reimplemented from [Couenne::expression](#).

Definition at line 92 of file CouenneExprBSin.hpp.

7.102.3.2 `CouNumber Couenne::exprUBSin::operator() ()` [inline], [virtual]

function for the evaluation of the expression

compute sum

Implements [Couenne::expression](#).

Definition at line 111 of file CouenneExprBSin.hpp.

7.102.3.3 `std::string Couenne::exprUBSin::printOp () const` [inline], [virtual]

print operator

Reimplemented from [Couenne::exprOp](#).

Definition at line 100 of file CouenneExprBSin.hpp.

7.102.3.4 enum pos Couenne::exprUBSin::printPos() const [inline], [virtual]

print position (PRE, INSIDE, POST)

Reimplemented from [Couenne::exprOp](#).

Definition at line 104 of file CouenneExprBSin.hpp.

The documentation for this class was generated from the following file:

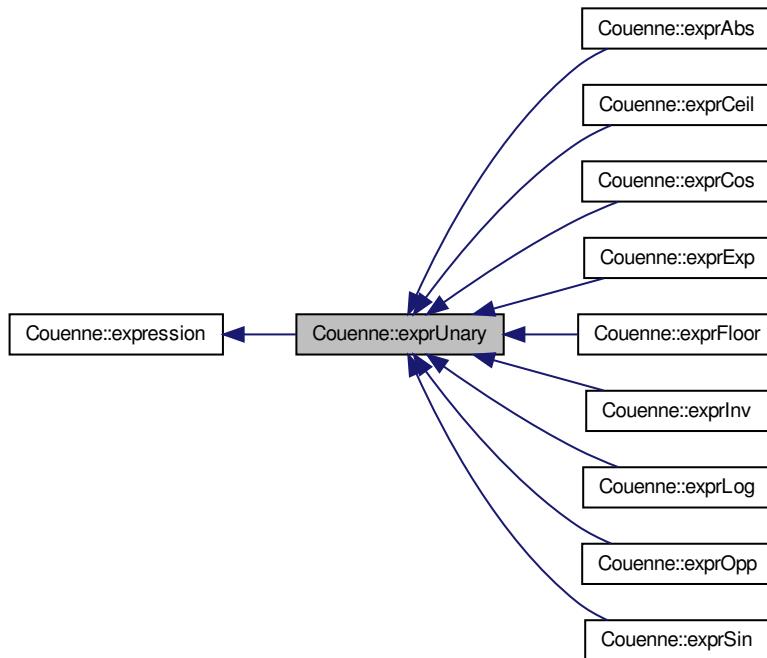
- /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/[CouenneExprBSin.hpp](#)

7.103 Couenne::exprUnary Class Reference

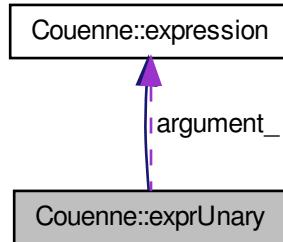
expression class for unary functions (sin, log, etc.)

```
#include <CouenneExprUnary.hpp>
```

Inheritance diagram for Couenne::exprUnary:



Collaboration diagram for Couenne::exprUnary:



Public Member Functions

- virtual enum `nodeType Type () const`
node type
- `exprUnary (expression *argument)`
Constructor.
- virtual `unary_function F ()`
the operator itself (e.g. sin, log...)
- `~exprUnary ()`
Destructor.
- int `nArgs () const`
return number of arguments
- virtual `expression * Argument () const`
return argument (when applicable, i.e., with univariate functions)
- virtual `expression ** ArgPtr ()`
return pointer to argument (when applicable, i.e., with univariate functions)
- virtual void `print (std::ostream &out=std::cout, bool=false) const`
print this expression to ostream
- virtual enum `pos printPos () const`
print position (PRE, INSIDE, POST)
- virtual std::string `printOp () const`
print operator
- virtual `CouNumber operator() ()`
compute value of unary operator
- virtual int `DepList (std::set< int > &depelist, enum dig_type type=ORIG_ONLY)`
fill in the set with all indices of variables appearing in the expression
- `expression * simplify ()`
simplification
- virtual int `Linearity ()`
get a measure of "how linear" the expression is (see CouenneTypes.h) for general univariate functions, return nonlinear.
- virtual `exprAux * standardize (CouenneProblem *, bool addAux=true)`

- *reduce expression in standard form, creating additional aux variables (and constraints)*
- virtual enum `expr_type code ()`
type of operator
- virtual bool `isInteger ()`
is this expression integer?
- virtual int `compare (exprUnary &)`
compare two unary functions
- virtual int `rank ()`
used in rank-based branching variable choice
- virtual void `fillDepSet (std::set< DepNode *, compNode > *dep, DepGraph *g)`
fill in dependence structure
- virtual void `replace (exprVar *, exprVar *)`
replace variable with other
- virtual void `realign (const CouenneProblem *p)`
empty function to redirect variables to proper variable vector

Protected Attributes

- `expression * argument_`
single argument taken by this expression

Additional Inherited Members

7.103.1 Detailed Description

expression class for unary functions (sin, log, etc.)

univariate operator-type expression: requires single argument. All unary functions are derived from this base class, which has a lot of common methods that need not be re-implemented by any univariate class.

Definition at line 33 of file CouenneExprUnary.hpp.

7.103.2 Constructor & Destructor Documentation

7.103.2.1 Couenne::exprUnary::exprUnary (`expression * argument`) [inline]

Constructor.

Definition at line 47 of file CouenneExprUnary.hpp.

7.103.2.2 Couenne::exprUnary::~exprUnary () [inline]

Destructor.

Definition at line 56 of file CouenneExprUnary.hpp.

7.103.3 Member Function Documentation

7.103.3.1 virtual enum nodeType Couenne::exprUnary::Type () const [inline], [virtual]

node type

Reimplemented from [Couenne::expression](#).

Definition at line 43 of file CouenneExprUnary.hpp.

7.103.3.2 virtual unary_function Couenne::exprUnary::F() [inline], [virtual]

the operator itself (e.g. sin, log...)

Reimplemented in [Couenne::exprSin](#), [Couenne::exprInv](#), [Couenne::exprOpp](#), [Couenne::exprExp](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), [Couenne::exprLog](#), and [Couenne::exprAbs](#).

Definition at line 52 of file CouenneExprUnary.hpp.

7.103.3.3 int Couenne::exprUnary::nArgs() const [inline], [virtual]

return number of arguments

Reimplemented from [Couenne::expression](#).

Definition at line 60 of file CouenneExprUnary.hpp.

7.103.3.4 virtual expression* Couenne::exprUnary::Argument() const [inline], [virtual]

return argument (when applicable, i.e., with univariate functions)

Reimplemented from [Couenne::expression](#).

Definition at line 64 of file CouenneExprUnary.hpp.

7.103.3.5 virtual expression** Couenne::exprUnary::ArgPtr() [inline], [virtual]

return pointer to argument (when applicable, i.e., with univariate functions)

Reimplemented from [Couenne::expression](#).

Definition at line 68 of file CouenneExprUnary.hpp.

7.103.3.6 virtual void Couenne::exprUnary::print(std::ostream & out = std::cout, bool = false) const [virtual]

print this expression to ostream

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprInv](#), and [Couenne::exprOpp](#).

7.103.3.7 virtual enum pos Couenne::exprUnary::printPos() const [inline], [virtual]

print position (PRE, INSIDE, POST)

Definition at line 75 of file CouenneExprUnary.hpp.

7.103.3.8 virtual std::string Couenne::exprUnary::printOp() const [inline], [virtual]

print operator

Reimplemented in [Couenne::exprSin](#), [Couenne::exprAbs](#), [Couenne::exprExp](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), and [Couenne::exprLog](#).

Definition at line 79 of file CouenneExprUnary.hpp.

7.103.3.9 virtual CouNumber Couenne::exprUnary::operator()() [inline], [virtual]

compute value of unary operator

Implements [Couenne::expression](#).

Definition at line 83 of file CouenneExprUnary.hpp.

7.103.3.10 `virtual int Couenne::exprUnary::DepList (std::set< int > & depList, enum dig_type type = ORIG_ONLY) [inline], [virtual]`

fill in the set with all indices of variables appearing in the expression

Reimplemented from [Couenne::expression](#).

Definition at line 88 of file CouenneExprUnary.hpp.

7.103.3.11 `expression* Couenne::exprUnary::simplify () [virtual]`

simplification

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOpp](#).

7.103.3.12 `virtual int Couenne::exprUnary::Linearity () [inline], [virtual]`

get a measure of "how linear" the expression is (see CouenneTypes.h) for general univariate functions, return nonlinear.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprInv](#), and [Couenne::exprOpp](#).

Definition at line 96 of file CouenneExprUnary.hpp.

7.103.3.13 `virtual exprAux* Couenne::exprUnary::standardize (CouenneProblem *, bool addAux = true) [virtual]`

reduce expression in standard form, creating additional aux variables (and constraints)

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOpp](#).

7.103.3.14 `virtual enum expr_type Couenne::exprUnary::code () [inline], [virtual]`

type of operator

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprSin](#), [Couenne::exprInv](#), [Couenne::exprOpp](#), [Couenne::exprCeil](#), [Couenne::exprCos](#), [Couenne::exprFloor](#), [Couenne::exprAbs](#), [Couenne::exprExp](#), and [Couenne::exprLog](#).

Definition at line 104 of file CouenneExprUnary.hpp.

7.103.3.15 `virtual bool Couenne::exprUnary::isInteger () [virtual]`

is this expression integer?

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprOpp](#), and [Couenne::exprAbs](#).

7.103.3.16 `virtual int Couenne::exprUnary::compare (exprUnary &) [virtual]`

compare two unary functions

7.103.3.17 `virtual int Couenne::exprUnary::rank() [inline], [virtual]`

used in rank-based branching variable choice

Reimplemented from [Couenne::expression](#).

Definition at line 114 of file CouenneExprUnary.hpp.

7.103.3.18 `virtual void Couenne::exprUnary::fillDepSet(std::set< DepNode *, compNode > * dep, DepGraph * g) [inline], [virtual]`

fill in dependence structure

Reimplemented from [Couenne::expression](#).

Definition at line 118 of file CouenneExprUnary.hpp.

7.103.3.19 `virtual void Couenne::exprUnary::replace(exprVar *, exprVar *) [virtual]`

replace variable with other

Reimplemented from [Couenne::expression](#).

7.103.3.20 `virtual void Couenne::exprUnary::realign(const CouenneProblem * p) [inline], [virtual]`

empty function to redirect variables to proper variable vector

Reimplemented from [Couenne::expression](#).

Definition at line 125 of file CouenneExprUnary.hpp.

7.103.4 Member Data Documentation

7.103.4.1 `expression* Couenne::exprUnary::argument_ [protected]`

single argument taken by this expression

Definition at line 38 of file CouenneExprUnary.hpp.

The documentation for this class was generated from the following file:

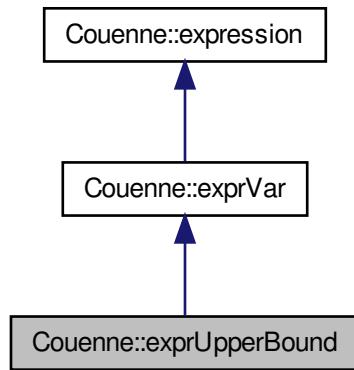
- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprUnary.hpp](#)

7.104 Couenne::exprUpperBound Class Reference

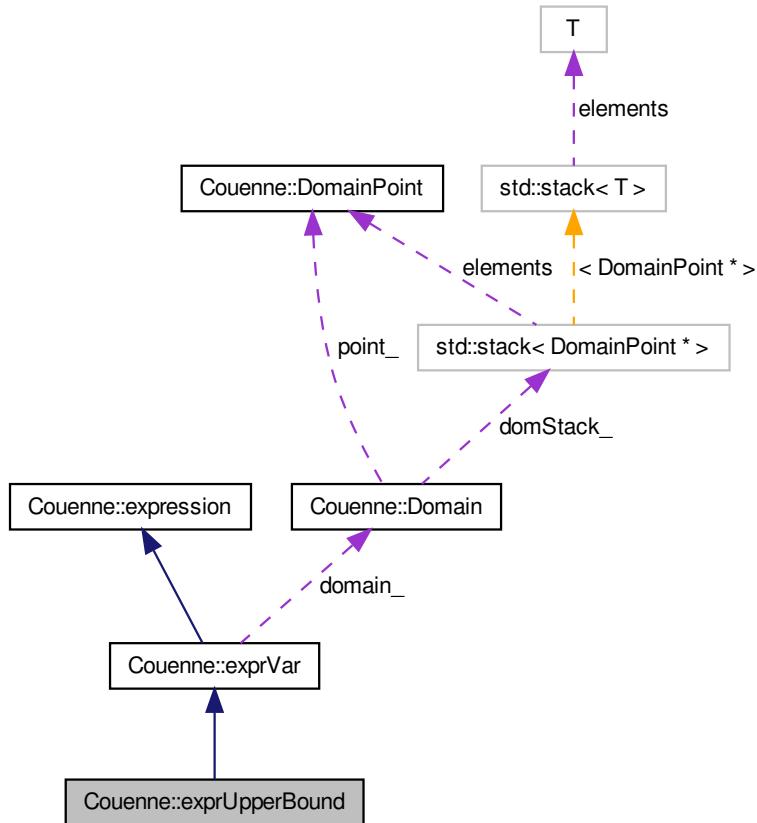
upper bound

```
#include <CouenneExprBound.hpp>
```

Inheritance diagram for Couenne::exprUpperBound:



Collaboration diagram for Couenne::exprUpperBound:



Public Member Functions

- enum `nodeType Type () const`
Node type.
- `exprUpperBound (int varIndex, Domain *d=NULL)`
Constructor.
- `exprUpperBound (const exprUpperBound &src, Domain *d=NULL)`
Copy constructor.
- `exprUpperBound * clone (Domain *d=NULL) const`
cloning method
- `void print (std::ostream &out=std::cout, bool=false) const`
Print to ostream.
- `CouNumber operator() ()`
return the value of the variable
- `expression * differentiate (int)`
differentiation

- int [dependsOn](#) (int *, int, enum [dig_type](#) type=[STOP_AT_AUX](#))
dependence on variable set
- virtual int [Linearity](#) ()
get a measure of "how linear" the expression is:
- virtual enum [expr_type](#) [code](#) ()
code for comparisons

Additional Inherited Members

7.104.1 Detailed Description

upper bound

Definition at line 87 of file CouenneExprBound.hpp.

7.104.2 Constructor & Destructor Documentation

7.104.2.1 Couenne::exprUpperBound::exprUpperBound (int varIndex, Domain * d=NULL) [inline]

Constructor.

Definition at line 96 of file CouenneExprBound.hpp.

7.104.2.2 Couenne::exprUpperBound::exprUpperBound (const exprUpperBound & src, Domain * d=NULL) [inline]

Copy constructor.

Definition at line 100 of file CouenneExprBound.hpp.

7.104.3 Member Function Documentation

7.104.3.1 enum nodeType Couenne::exprUpperBound::Type () const [inline], [virtual]

[Node](#) type.

Reimplemented from [Couenne::exprVar](#).

Definition at line 92 of file CouenneExprBound.hpp.

7.104.3.2 exprUpperBound* Couenne::exprUpperBound::clone (Domain * d=NULL) const [inline], [virtual]

cloning method

Reimplemented from [Couenne::exprVar](#).

Definition at line 104 of file CouenneExprBound.hpp.

7.104.3.3 void Couenne::exprUpperBound::print (std::ostream & out = std::cout, bool = false) const [inline], [virtual]

Print to iostream.

Reimplemented from [Couenne::exprVar](#).

Definition at line 108 of file CouenneExprBound.hpp.

7.104.3.4 CouNumber Couenne::exprUpperBound::operator()() [inline], [virtual]

return the value of the variable

Reimplemented from [Couenne::exprVar](#).

Definition at line 113 of file CouenneExprBound.hpp.

7.104.3.5 expression* Couenne::exprUpperBound::differentiate(int) [inline], [virtual]

differentiation

Reimplemented from [Couenne::exprVar](#).

Definition at line 117 of file CouenneExprBound.hpp.

7.104.3.6 int Couenne::exprUpperBound::dependsOn(int *, int , enum dig_type type = STOP_AT_AUX) [inline], [virtual]

dependence on variable set

Reimplemented from [Couenne::expression](#).

Definition at line 121 of file CouenneExprBound.hpp.

7.104.3.7 virtual int Couenne::exprUpperBound::Linearity() [inline], [virtual]

get a measure of "how linear" the expression is:

Reimplemented from [Couenne::exprVar](#).

Definition at line 125 of file CouenneExprBound.hpp.

7.104.3.8 virtual enum expr_type Couenne::exprUpperBound::code() [inline], [virtual]

code for comparisons

Reimplemented from [Couenne::exprVar](#).

Definition at line 129 of file CouenneExprBound.hpp.

The documentation for this class was generated from the following file:

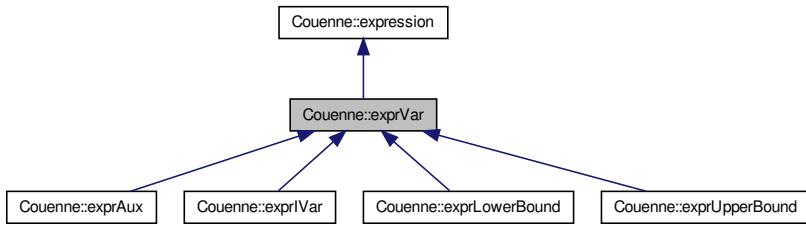
- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprBound.hpp](#)

7.105 Couenne::exprVar Class Reference

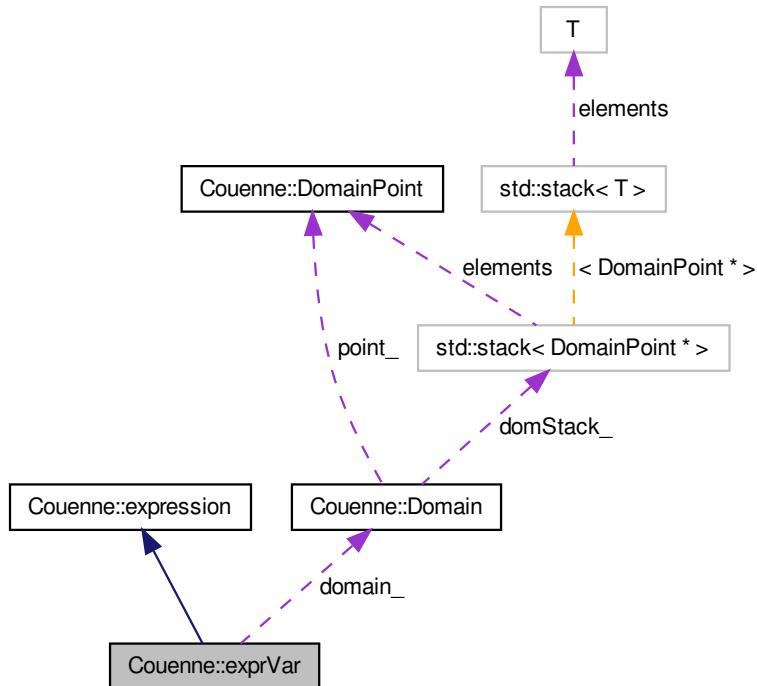
variable-type operator

```
#include <CouenneExprVar.hpp>
```

Inheritance diagram for Couenne::exprVar:



Collaboration diagram for Couenne::exprVar:



Public Member Functions

- virtual enum `nodeType Type () const`
`Node type.`
- `exprVar (int varIndex, Domain *d=NULL)`
`Constructor.`

- virtual `~exprVar ()`
destructor
- `exprVar (const exprVar &e, Domain *d=NULL)`
Copy constructor.
- virtual `exprVar * clone (Domain *d=NULL) const`
Cloning method.
- int `Index () const`
Get variable index in problem.
- virtual `expression * Lb ()`
Get lower bound expression.
- virtual `expression * Ub ()`
Get upper bound expression.
- virtual `CouNumber & lb ()`
Get/set lower bound value.
- virtual `CouNumber & ub ()`
Get/set upper bound value.
- virtual void `print (std::ostream &out=std::cout, bool=false) const`
print
- virtual `CouNumber operator() ()`
return the value of the variable
- virtual `CouNumber gradientNorm (const double *x)`
return l-2 norm of gradient at given point
- virtual `expression * differentiate (int index)`
differentiation
- virtual int `DepList (std::set< int > &depList, enum dig_type type=ORIG_ONLY)`
fill in the set with all indices of variables appearing in the expression
- virtual void `crossBounds ()`
set bounds depending on both branching rules and propagated bounds.
- virtual `expression * simplify ()`
simplify
- virtual int `Linearity ()`
get a measure of "how linear" the expression is (see CouenneTypes.hpp)
- virtual bool `isDefinedInteger ()`
is this expression defined as an integer?
- virtual bool `isInteger ()`
is this variable integer?
- virtual void `getBounds (expression *&, expression *&)`
Get expressions of lower and upper bound of an expression (if any)
- virtual void `getBounds (CouNumber &lb, CouNumber &ub)`
Get value of lower and upper bound of an expression (if any)
- virtual void `generateCuts (OsiCuts &, const CouenneCutGenerator *, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
Get values of lower and upper bound of an expression (if any)
- virtual void `generateCuts (expression *w, OsiCuts &cs, const CouenneCutGenerator *cg, t_chg_bounds *=NULL, int=-1, CouNumber=-COUENNE_INFINITY, CouNumber=COUENNE_INFINITY)`
generate convexification cut for constraint w = this
- virtual enum `expr_type code ()`

- *code for comparison*
- virtual bool **impliedBound** (int, CouNumber *, CouNumber *, t_chg_bounds *, enum auxSign=expression::AUX_EQ)
 - implied bound processing*
- virtual int **rank** ()
 - rank of an original variable is always one*
- virtual void **fillDepSet** (std::set< DepNode *, compNode > *, DepGraph *)
 - update dependence set with index of this variable*
- virtual bool **isFixed** ()
 - is this variable fixed?*
- virtual void **linkDomain** (Domain *d)
 - link this variable to a domain*
- virtual Domain * **domain** ()
 - return pointer to variable domain*
- virtual void **decreaseMult** ()
 - Disable variable (empty for compatibility with exprAux)*
- virtual void **setInteger** (bool value)
 - Set this variable as integer (empty for compatibility with exprAux)*
- virtual enum convexity **convexity** () const
 - either CONVEX, CONCAVE, AFFINE, or NONCONVEX*
- virtual CouenneObject * **properObject** (CouenneCutGenerator *c, CouenneProblem *p, Bonmin::BabSetupBase *base, JnlstPtr jnlst_)
 - return proper object to handle expression associated with this variable (NULL if this is not an auxiliary)*
- virtual enum auxSign **sign** () const
 - return its sign in the definition constraint*

Protected Attributes

- int **varIndex_**
 - The index of the variable.*
- Domain * **domain_**
 - Pointer to a descriptor of the current point/bounds.*

Additional Inherited Members

7.105.1 Detailed Description

variable-type operator

All variables of the expression must be objects of this class or of the derived **exprAux** class

Definition at line 45 of file CouenneExprVar.hpp.

7.105.2 Constructor & Destructor Documentation

7.105.2.1 Couenne::exprVar::exprVar (int varIndex, Domain * d = NULL) [inline]

Constructor.

Definition at line 59 of file CouenneExprVar.hpp.

7.105.2.2 `virtual Couenne::exprVar::~exprVar() [inline], [virtual]`

destructor

Definition at line 64 of file CouenneExprVar.hpp.

7.105.2.3 `Couenne::exprVar::exprVar(const exprVar & e, Domain * d=NULL) [inline]`

Copy constructor.

Definition at line 67 of file CouenneExprVar.hpp.

7.105.3 Member Function Documentation

7.105.3.1 `virtual enum nodeType Couenne::exprVar::Type() const [inline], [virtual]`

[Node](#) type.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprUpperBound](#), [Couenne::exprAux](#), and [Couenne::exprLowerBound](#).

Definition at line 55 of file CouenneExprVar.hpp.

7.105.3.2 `virtual exprVar* Couenne::exprVar::clone(Domain * d=NULL) const [inline], [virtual]`

Cloning method.

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprUpperBound](#), [Couenne::exprAux](#), [Couenne::exprLowerBound](#), and [Couenne::exprVar](#).

Definition at line 72 of file CouenneExprVar.hpp.

7.105.3.3 `int Couenne::exprVar::Index() const [inline], [virtual]`

Get variable index in problem.

Reimplemented from [Couenne::expression](#).

Definition at line 76 of file CouenneExprVar.hpp.

7.105.3.4 `virtual expression* Couenne::exprVar::Lb() [virtual]`

Get lower bound expression.

Reimplemented in [Couenne::exprAux](#).

7.105.3.5 `virtual expression* Couenne::exprVar::Ub() [virtual]`

Get upper bound expression.

Reimplemented in [Couenne::exprAux](#).

7.105.3.6 `virtual CouNumber& Couenne::exprVar::lb() [inline], [virtual]`

Get/set lower bound value.

Definition at line 84 of file CouenneExprVar.hpp.

7.105.3.7 **virtual CouNumber& Couenne::exprVar::ub() [inline], [virtual]**

Get/set upper bound value.

Definition at line 85 of file CouenneExprVar.hpp.

7.105.3.8 **virtual void Couenne::exprVar::print(std::ostream & out = std::cout, bool = false) const [inline], [virtual]**

print

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprUpperBound](#), [Couenne::exprAux](#), [Couenne::exprLowerBound](#), and [Couenne::exprVar](#).

Definition at line 88 of file CouenneExprVar.hpp.

7.105.3.9 **virtual CouNumber Couenne::exprVar::operator()() [inline], [virtual]**

return the value of the variable

Implements [Couenne::expression](#).

Reimplemented in [Couenne::exprUpperBound](#), [Couenne::exprAux](#), and [Couenne::exprLowerBound](#).

Definition at line 93 of file CouenneExprVar.hpp.

7.105.3.10 **virtual CouNumber Couenne::exprVar::gradientNorm(const double * x) [inline], [virtual]**

return l-2 norm of gradient at given point

Reimplemented from [Couenne::expression](#).

Definition at line 97 of file CouenneExprVar.hpp.

7.105.3.11 **virtual expression* Couenne::exprVar::differentiate(int index) [inline], [virtual]**

differentiation

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprUpperBound](#), and [Couenne::exprLowerBound](#).

Definition at line 101 of file CouenneExprVar.hpp.

7.105.3.12 **virtual int Couenne::exprVar::DepList(std::set< int > & depList, enum dig_type type = ORIG_ONLY) [inline], [virtual]**

fill in the set with all indices of variables appearing in the expression

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprAux](#).

Definition at line 106 of file CouenneExprVar.hpp.

7.105.3.13 **virtual void Couenne::exprVar::crossBounds() [inline], [virtual]**

set bounds depending on both branching rules and propagated bounds.

To be used after standardization

Reimplemented in [Couenne::exprAux](#).

Definition at line 118 of file CouenneExprVar.hpp.

7.105.3.14 virtual **expression*** Couenne::exprVar::simplify() [inline], [virtual]

simplify

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprAux](#).

Definition at line 121 of file CouenneExprVar.hpp.

7.105.3.15 virtual int Couenne::exprVar::Linearity() [inline], [virtual]

get a measure of "how linear" the expression is (see [CouenneTypes.hpp](#))

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprUpperBound](#), [Couenne::exprAux](#), and [Couenne::exprLowerBound](#).

Definition at line 125 of file CouenneExprVar.hpp.

7.105.3.16 virtual bool Couenne::exprVar::isDefinedInteger() [inline], [virtual]

is this expression defined as an integer?

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprAux](#), and [Couenne::exprVar](#).

Definition at line 129 of file CouenneExprVar.hpp.

7.105.3.17 virtual bool Couenne::exprVar::isInteger() [inline], [virtual]

is this variable integer?

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprAux](#), and [Couenne::exprVar](#).

Definition at line 133 of file CouenneExprVar.hpp.

7.105.3.18 virtual void Couenne::exprVar::getBounds(**expression*&**, **expression*&**) [virtual]

Get expressions of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.105.3.19 virtual void Couenne::exprVar::getBounds(**CouNumber & lb**, **CouNumber & ub**) [virtual]

Get value of lower and upper bound of an expression (if any)

Reimplemented from [Couenne::expression](#).

7.105.3.20 virtual void Couenne::exprVar::generateCuts(**OsiCuts &**, **const CouenneCutGenerator * , t_chg_bounds**
*** =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =COUENNE_INFINITY**)
[inline], [virtual]

Get values of lower and upper bound of an expression (if any)

generate cuts for expression associated with this auxiliary

Reimplemented in [Couenne::exprAux](#).

Definition at line 156 of file CouenneExprVar.hpp.

```
7.105.3.21 virtual void Couenne::exprVar::generateCuts( expression * w, OsiCuts & cs, const CouenneCutGenerator  
* cg, t_chg_bounds * =NULL, int =-1, CouNumber =-COUENNE_INFINITY, CouNumber =  
COUENNE_INFINITY ) [virtual]
```

generate convexification cut for constraint w = this

Reimplemented from [Couenne::expression](#).

```
7.105.3.22 virtual enum expr_type Couenne::exprVar::code( ) [inline], [virtual]
```

code for comparison

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprUpperBound](#), and [Couenne::exprLowerBound](#).

Definition at line 171 of file CouenneExprVar.hpp.

```
7.105.3.23 virtual bool Couenne::exprVar::impliedBound( int , CouNumber * , CouNumber * , t_chg_bounds * , enum  
auxSign = expression::AUX_EQ ) [virtual]
```

implied bound processing

Reimplemented from [Couenne::expression](#).

```
7.105.3.24 virtual int Couenne::exprVar::rank( ) [inline], [virtual]
```

rank of an original variable is always one

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprAux](#).

Definition at line 178 of file CouenneExprVar.hpp.

```
7.105.3.25 virtual void Couenne::exprVar::fillDepSet( std::set< DepNode *, compNode *> * , DepGraph * )  
[virtual]
```

update dependence set with index of this variable

Reimplemented from [Couenne::expression](#).

```
7.105.3.26 virtual bool Couenne::exprVar::isFixed( ) [inline], [virtual]
```

is this variable fixed?

Definition at line 185 of file CouenneExprVar.hpp.

```
7.105.3.27 virtual void Couenne::exprVar::linkDomain( Domain * d ) [inline], [virtual]
```

link this variable to a domain

Reimplemented from [Couenne::expression](#).

Reimplemented in [Couenne::exprAux](#).

Definition at line 189 of file CouenneExprVar.hpp.

```
7.105.3.28 virtual Domain* Couenne::exprVar::domain( ) [inline], [virtual]
```

return pointer to variable domain

Definition at line 193 of file CouenneExprVar.hpp.

7.105.3.29 `virtual void Couenne::exprVar::decreaseMult() [inline], [virtual]`

Reimplemented in [Couenne::exprAux](#).

Definition at line 197 of file CouenneExprVar.hpp.

7.105.3.30 `virtual void Couenne::exprVar::zeroMult() [inline], [virtual]`

Disable variable (empty for compatibility with [exprAux](#))

Reimplemented in [Couenne::exprAux](#).

Definition at line 200 of file CouenneExprVar.hpp.

7.105.3.31 `virtual void Couenne::exprVar::setInteger(bool value) [inline], [virtual]`

Set this variable as integer (empty for compatibility with [exprAux](#))

Reimplemented in [Couenne::exprAux](#).

Definition at line 203 of file CouenneExprVar.hpp.

7.105.3.32 `virtual enum convexity Couenne::exprVar::convexity() const [inline], [virtual]`

either CONVEX, CONCAVE, AFFINE, or NONCONVEX

Reimplemented from [Couenne::expression](#).

Definition at line 206 of file CouenneExprVar.hpp.

7.105.3.33 `virtual CouenneObject* Couenne::exprVar::properObject(CouenneCutGenerator * c, CouenneProblem * p, Bonmin::BabSetupBase * base, JnlstPtr jnlst_) [virtual]`

return proper object to handle expression associated with this variable (NULL if this is not an auxiliary)

Reimplemented in [Couenne::exprAux](#).

7.105.3.34 `virtual enum auxSign Couenne::exprVar::sign() const [inline], [virtual]`

return its sign in the definition constraint

Reimplemented in [Couenne::exprAux](#).

Definition at line 217 of file CouenneExprVar.hpp.

7.105.4 Member Data Documentation

7.105.4.1 `int Couenne::exprVar::varIndex_ [protected]`

The index of the variable.

Definition at line 49 of file CouenneExprVar.hpp.

7.105.4.2 `Domain* Couenne::exprVar::domain_ [protected]`

Pointer to a descriptor of the current point/bounds.

Definition at line 50 of file CouenneExprVar.hpp.

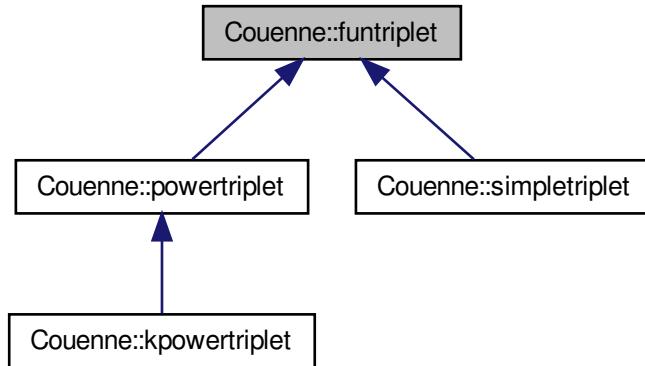
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/[CouenneExprVar.hpp](#)

7.106 Couenne::funtriplet Class Reference

```
#include <CouenneFunTriplets.hpp>
```

Inheritance diagram for Couenne::funtriplet:



Public Member Functions

- [funtriplet \(\)](#)
Basic constructor.
- [virtual ~funtriplet \(\)](#)
Destructor.
- [virtual CouNumber F \(CouNumber x\)=0](#)
- [virtual CouNumber Fp \(CouNumber x\)=0](#)
- [virtual CouNumber Fpp \(CouNumber x\)=0](#)
- [virtual CouNumber FpInv \(CouNumber x\)=0](#)

7.106.1 Detailed Description

Definition at line 22 of file CouenneFunTriplets.hpp.

7.106.2 Constructor & Destructor Documentation

7.106.2.1 Couenne::funtriplet::funtriplet() [inline]

Basic constructor.

Definition at line 27 of file CouenneFunTriplets.hpp.

7.106.2.2 virtual Couenne::funtriplet::~funtriplet() [inline], [virtual]

Destructor.

Definition at line 30 of file CouenneFunTriplets.hpp.

7.106.3 Member Function Documentation

7.106.3.1 `virtual CouNumber Couenne::funtriplet::F(CouNumber x) [pure virtual]`

Implemented in [Couenne::kpowertriplet](#), [Couenne::powertriplet](#), and [Couenne::simpletriplet](#).

7.106.3.2 `virtual CouNumber Couenne::funtriplet::Fp(CouNumber x) [pure virtual]`

Implemented in [Couenne::kpowertriplet](#), [Couenne::powertriplet](#), and [Couenne::simpletriplet](#).

7.106.3.3 `virtual CouNumber Couenne::funtriplet::Fpp(CouNumber x) [pure virtual]`

Implemented in [Couenne::kpowertriplet](#), [Couenne::powertriplet](#), and [Couenne::simpletriplet](#).

7.106.3.4 `virtual CouNumber Couenne::funtriplet::FpInv(CouNumber x) [pure virtual]`

Implemented in [Couenne::kpowertriplet](#), [Couenne::powertriplet](#), and [Couenne::simpletriplet](#).

The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Couenne/src/util/CouenneFunTriplets.hpp](#)

7.107 Couenne::GlobalCutOff Class Reference

```
#include <CouenneGlobalCutOff.hpp>
```

Public Member Functions

- [GlobalCutOff\(\)](#)
- [GlobalCutOff\(double c, const double *s=NULL, int n=0\)](#)
- [~GlobalCutOff\(\)](#)
- [void setCutOff\(const CouenneProblem *p, double cutoff, const double *s=NULL\)](#)
- [double getCutOff\(\) const](#)
- [double * getCutOffSol\(\) const](#)

7.107.1 Detailed Description

Definition at line 19 of file [CouenneGlobalCutOff.hpp](#).

7.107.2 Constructor & Destructor Documentation

7.107.2.1 `Couenne::GlobalCutOff::GlobalCutOff()`

7.107.2.2 `Couenne::GlobalCutOff::GlobalCutOff(double c, const double * s = NULL, int n = 0)`

7.107.2.3 `Couenne::GlobalCutOff::~GlobalCutOff()`

7.107.3 Member Function Documentation

7.107.3.1 `void Couenne::GlobalCutOff::setCutOff(const CouenneProblem * p, double cutoff, const double * s = NULL)`

7.107.3.2 double Couenne::GlobalCutOff::getCutOff() const [inline]

Definition at line 38 of file CouenneGlobalCutOff.hpp.

7.107.3.3 double* Couenne::GlobalCutOff::getCutOffSol() const [inline]

Definition at line 39 of file CouenneGlobalCutOff.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/problem/CouenneGlobalCutOff.hpp

7.108 Couenne::InitHeuristic Class Reference

A heuristic that stores the initial solution of the NLP.

```
#include <BonInitHeuristic.hpp>
```

Public Member Functions

- **InitHeuristic** (double objValue, const double *sol, [CouenneProblem &cp](#))

Constructor with model and Ipopt problems.
- **InitHeuristic** (const [InitHeuristic](#) &other)

Copy constructor.
- virtual **~InitHeuristic** ()

Destructor.
- virtual CbcHeuristic * **clone** () const

Clone.
- **InitHeuristic** & **operator=** (const [InitHeuristic](#) &rhs)

Assignment operator.
- virtual void **resetModel** (CbcModel *model)
- virtual int **solution** (double &objectiveValue, double *newSolution)

Run heuristic, return 1 if a better solution than the one passed is found and 0 otherwise.

7.108.1 Detailed Description

A heuristic that stores the initial solution of the NLP.

This is computed before Cbc is started, and in this way we can tell Cbc about this.

Definition at line 24 of file BonInitHeuristic.hpp.

7.108.2 Constructor & Destructor Documentation

7.108.2.1 Couenne::InitHeuristic::InitHeuristic (double *objValue*, const double * *sol*, CouenneProblem & *cp*)

Constructor with model and Ipopt problems.

7.108.2.2 Couenne::InitHeuristic::InitHeuristic (const [InitHeuristic](#) & *other*)

Copy constructor.

7.108.2.3 virtual Couenne::InitHeuristic::~InitHeuristic() [virtual]

Destructor.

7.108.3 Member Function Documentation

7.108.3.1 virtual CbcHeuristic* Couenne::InitHeuristic::clone() const [virtual]

Clone.

7.108.3.2 InitHeuristic& Couenne::InitHeuristic::operator= (const InitHeuristic & rhs)

Assignment operator.

7.108.3.3 virtual void Couenne::InitHeuristic::resetModel (CbcModel * model) [inline], [virtual]

Definition at line 42 of file BonInitHeuristic.hpp.

7.108.3.4 virtual int Couenne::InitHeuristic::solution (double & objectiveValue, double * newSolution) [virtual]

Run heuristic, return 1 if a better solution than the one passed is found and 0 otherwise.

objectiveValue Best known solution in input and value of solution found in output newSolution Solution found by heuristic.

Todo Find a quicker way to get to [Couenne](#) objects, store them or something

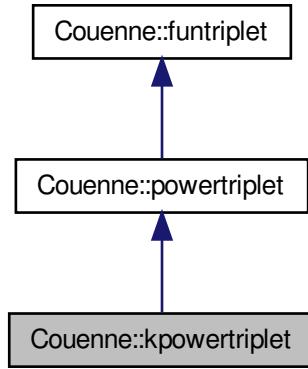
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/heuristics/[BonInitHeuristic.hpp](#)

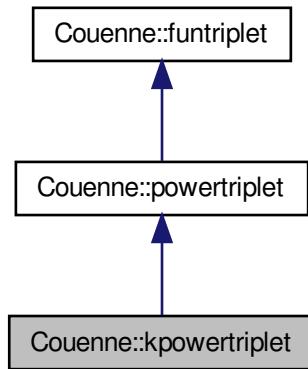
7.109 Couenne::kpowertriplet Class Reference

```
#include <CouenneFunTriplets.hpp>
```

Inheritance diagram for Couenne::kpowertriplet:



Collaboration diagram for Couenne::kpowertriplet:



Public Member Functions

- [kpowertriplet \(CouNumber exponent, CouNumber k\)](#)
Basic constructor.
- [virtual ~kpowertriplet \(\)](#)
Destructor.
- [virtual CouNumber F \(CouNumber x\)](#)
- [virtual CouNumber Fp \(CouNumber x\)](#)

- virtual [CouNumber Fpp \(CouNumber x\)](#)
- virtual [CouNumber FpInv \(CouNumber x\)](#)

Protected Attributes

- [CouNumber mult_](#)

7.109.1 Detailed Description

Definition at line 103 of file CouenneFunTriplets.hpp.

7.109.2 Constructor & Destructor Documentation

7.109.2.1 Couenne::kpowertriplet::kpowertriplet (CouNumber exponent, CouNumber k) [inline]

Basic constructor.

Definition at line 112 of file CouenneFunTriplets.hpp.

7.109.2.2 virtual Couenne::kpowertriplet::~kpowertriplet () [inline], [virtual]

Destructor.

Definition at line 117 of file CouenneFunTriplets.hpp.

7.109.3 Member Function Documentation

7.109.3.1 virtual CouNumber Couenne::kpowertriplet::F (CouNumber x) [inline], [virtual]

Reimplemented from [Couenne::powertriplet](#).

Definition at line 119 of file CouenneFunTriplets.hpp.

7.109.3.2 virtual CouNumber Couenne::kpowertriplet::Fp (CouNumber x) [inline], [virtual]

Reimplemented from [Couenne::powertriplet](#).

Definition at line 122 of file CouenneFunTriplets.hpp.

7.109.3.3 virtual CouNumber Couenne::kpowertriplet::Fpp (CouNumber x) [inline], [virtual]

Reimplemented from [Couenne::powertriplet](#).

Definition at line 125 of file CouenneFunTriplets.hpp.

7.109.3.4 virtual CouNumber Couenne::kpowertriplet::FpInv (CouNumber x) [inline], [virtual]

Reimplemented from [Couenne::powertriplet](#).

Definition at line 128 of file CouenneFunTriplets.hpp.

7.109.4 Member Data Documentation

7.109.4.1 CouNumber Couenne::kpowertriplet::mult_ [protected]

Definition at line 107 of file CouenneFunTriplets.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/util/CouenneFunTriplets.hpp

7.110 less_than_str Struct Reference

```
#include <CouenneProblem.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (register const char *a, register const char *b) const

7.110.1 Detailed Description

Definition at line 128 of file CouenneProblem.hpp.

7.110.2 Member Function Documentation

7.110.2.1 bool less_than_str::operator() (register const char * a, register const char * b) const [inline]

Definition at line 129 of file CouenneProblem.hpp.

The documentation for this struct was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/problem/CouenneProblem.hpp

7.111 Couenne::LinMap Class Reference

```
#include <CouenneLQelems.hpp>
```

Public Member Functions

- std::map< int, CouNumber > & [Map](#) ()
public access
- void [insert](#) (int index, CouNumber coe)
insert a pair <int,CouNumber> into a map for linear terms

7.111.1 Detailed Description

Definition at line 48 of file CouenneLQelems.hpp.

7.111.2 Member Function Documentation

7.111.2.1 `std::map<int, CouNumber>& Couenne::LinMap::Map() [inline]`

public access

Definition at line 56 of file CouenneLQelems.hpp.

7.111.2.2 `void Couenne::LinMap::insert(int index, CouNumber coe) [inline]`

insert a pair `<int,CouNumber>` into a map for linear terms

Definition at line 60 of file CouenneLQelems.hpp.

The documentation for this class was generated from the following file:

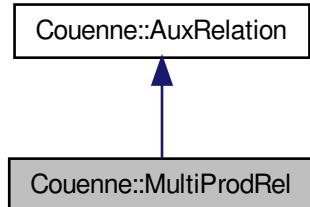
- /home/ted/COIN/trunk/Couenne/src/standardize/CouenneLQelems.hpp

7.112 Couenne::MultiProdRel Class Reference

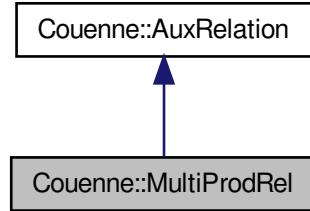
Identifies 5-ples of variables of the form.

```
#include <CouenneCrossConv.hpp>
```

Inheritance diagram for Couenne::MultiProdRel:



Collaboration diagram for Couenne::MultiProdRel:



Public Member Functions

- virtual int [findRelations \(\)](#)
- virtual void [generateCuts](#) (const OsiSolverInterface &, OsiCuts &, const CglTreeInfo=CglTreeInfo()) const

7.112.1 Detailed Description

Identifies 5-ples of variables of the form.

$x_k := x_i x_j x_l := x_i x_p$
 $x_q := x_k x_p$ OR $x_q := x_k / x_j$ $x_r := x_k x_j x_r := x_l / x_p$

and generates, ONLY ONCE, a cut

$x_q = x_r$ (in both cases).

Definition at line 82 of file CouenneCrossConv.hpp.

7.112.2 Member Function Documentation

7.112.2.1 virtual int Couenne::MultiProdRel::findRelations () [virtual]

Implements [Couenne::AuxRelation](#).

7.112.2.2 virtual void Couenne::MultiProdRel::generateCuts (const OsiSolverInterface & , OsiCuts & , const CglTreeInfo = CglTreeInfo()) const [virtual]

Reimplemented from [Couenne::AuxRelation](#).

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/cut/crossconv/CouenneCrossConv.hpp

7.113 myclass Struct Reference

```
#include <CouenneProblem.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (register const [Node](#) &a, register const [Node](#) &b)

7.113.1 Detailed Description

Definition at line 122 of file CouenneProblem.hpp.

7.113.2 Member Function Documentation

7.113.2.1 bool myclass::operator() (register const Node & a, register const Node & b) [inline]

Definition at line 123 of file CouenneProblem.hpp.

The documentation for this struct was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/problem/[CouenneProblem.hpp](#)

7.114 myclass0 Struct Reference

```
#include <CouenneProblem.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (register const [Node](#) &a, register const [Node](#) &b)

7.114.1 Detailed Description

Definition at line 76 of file CouenneProblem.hpp.

7.114.2 Member Function Documentation

7.114.2.1 bool myclass0::operator() (register const Node & a, register const Node & b) [inline]

Definition at line 77 of file CouenneProblem.hpp.

The documentation for this struct was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/problem/[CouenneProblem.hpp](#)

7.115 Nauty Class Reference

```
#include <Nauty.h>
```

Public Types

- enum [VarStatus](#) { [FIX_AT_ZERO](#), [FIX_AT_ONE](#), [FREE](#) }

Public Member Functions

- `Nauty (int n_)`
- `~Nauty ()`
- `void addElement (int ix, int jx)`
- `void clearPartitions ()`
- `void computeAuto ()`
- `void deleteElement (int ix, int jx)`
- `void color_node (int ix, int color)`
- `void insertRHS (int rhs, int cons)`
- `double getGroupSize () const`
- `int getNautyCalls () const`
- `double getNautyTime () const`
- `int getN () const`
- `int getNumGenerators () const`
- `int getNumOrbits () const`
- `std::vector< std::vector< int > > * getOrbits () const`
Returns the orbits in a "convenient" form.
- `void getVstat (double *v, int nv)`
- `void setWriteAutoms (const std::string &filename)`
Methods to classify orbits.
- `void unsetWriteAutoms ()`

7.115.1 Detailed Description

Definition at line 23 of file Nauty.h.

7.115.2 Member Enumeration Documentation

7.115.2.1 enum Nauty::VarStatus

Enumerator:

- `FIX_AT_ZERO`
- `FIX_AT_ONE`
- `FREE`

Definition at line 27 of file Nauty.h.

7.115.3 Constructor & Destructor Documentation

7.115.3.1 Nauty::Nauty (int n_)

7.115.3.2 Nauty::~Nauty ()

7.115.4 Member Function Documentation

7.115.4.1 void Nauty::addElement (int ix, int jx)

7.115.4.2 void Nauty::clearPartitions ()

7.115.4.3 void Nauty::computeAuto()

7.115.4.4 void Nauty::deleteElement(int ix, int jx)

7.115.4.5 void Nauty::color_node(int ix, int color) [inline]

Definition at line 36 of file Nauty.h.

7.115.4.6 void Nauty::insertRHS(int rhs, int cons) [inline]

Definition at line 37 of file Nauty.h.

7.115.4.7 double Nauty::getGroupSize() const

7.115.4.8 int Nauty::getNautyCalls() const [inline]

Definition at line 40 of file Nauty.h.

7.115.4.9 double Nauty::getNautyTime() const [inline]

Definition at line 41 of file Nauty.h.

7.115.4.10 int Nauty::getN() const [inline]

Definition at line 43 of file Nauty.h.

7.115.4.11 int Nauty::getNumGenerators() const

7.115.4.12 int Nauty::getNumOrbits() const

7.115.4.13 std::vector<std::vector<int>>* Nauty::getOrbits() const

Returns the orbits in a "convenient" form.

7.115.4.14 void Nauty::getVstat(double *v, int nv)

7.115.4.15 void Nauty::setWriteAutoms(const std::string &filename)

Methods to classify orbits.

Not horribly efficient, but gets the job done

7.115.4.16 void Nauty::unsetWriteAutoms()

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/branch/[Nauty.h](#)

7.116 Couenne::CouenneInfo::NlpSolution Class Reference

Class for storing an Nlp Solution.

```
#include <BonCouenneInfo.hpp>
```

Public Member Functions

- [NlpSolution](#) (int n, const double *sol, double objval)

- [~NlpSolution \(\)](#)

Accessor methods

- const double * [solution \(\) const](#)
- double [objVal \(\) const](#)
- int [nVars \(\) const](#)

7.116.1 Detailed Description

Class for storing an Nlp Solution.

Definition at line 26 of file BonCouenneInfo.hpp.

7.116.2 Constructor & Destructor Documentation

7.116.2.1 [Couenne::CouenneInfo::NlpSolution::NlpSolution \(int n, const double * sol, double objval \)](#)

7.116.2.2 [Couenne::CouenneInfo::NlpSolution::~NlpSolution \(\)](#)

7.116.3 Member Function Documentation

7.116.3.1 [const double* Couenne::CouenneInfo::NlpSolution::solution \(\) const \[inline\]](#)

Definition at line 37 of file BonCouenneInfo.hpp.

7.116.3.2 [double Couenne::CouenneInfo::NlpSolution::objVal \(\) const \[inline\]](#)

Definition at line 41 of file BonCouenneInfo.hpp.

7.116.3.3 [int Couenne::CouenneInfo::NlpSolution::nVars \(\) const \[inline\]](#)

Definition at line 45 of file BonCouenneInfo.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/main/[BonCouenneInfo.hpp](#)

7.117 Couenne::NlpSolveHeuristic Class Reference

```
#include <BonNlpHeuristic.hpp>
```

Public Member Functions

- [NlpSolveHeuristic \(\)](#)
Default constructor.
- [NlpSolveHeuristic \(CbcModel &mip, Bonmin::OsiTMINLPInterface &nlp, bool cloneNlp=false, CouenneProblem *couenne=NULL\)](#)
Constructor with model and [Ipopt](#) problems.
- [NlpSolveHeuristic \(const NlpSolveHeuristic &other\)](#)
Copy constructor.

- virtual `~NlpSolveHeuristic ()`
Destructor.
- virtual `CbcHeuristic * clone () const`
Clone.
- `NlpSolveHeuristic & operator= (const NlpSolveHeuristic &rhs)`
Assignment operator.
- void `setNlp (Bonmin::OsiTMINLPInterface &nlp, bool cloneNlp=true)`
Set the nlp solver.
- void `setCouenneProblem (CouenneProblem *)`
set the couenne problem to use.
- virtual void `resetModel (CbcModel *model)`
Does nothing.
- virtual int `solution (double &objectiveValue, double *newSolution)`
Run heuristic, return 1 if a better solution than the one passed is found and 0 otherwise.
- void `setMaxNlpInf (double value)`
set maxNlpInf.
- void `setNumberSolvePerLevel (int value)`
set number of nlp's solved for each given level of the tree

Static Public Member Functions

- static void `registerOptions (Ipopt::SmartPtr< Bonmin::RegisteredOptions >)`
initialize options

7.117.1 Detailed Description

Definition at line 28 of file BonNlpHeuristic.hpp.

7.117.2 Constructor & Destructor Documentation

7.117.2.1 Couenne::NlpSolveHeuristic::NlpSolveHeuristic ()

Default constructor.

7.117.2.2 Couenne::NlpSolveHeuristic::NlpSolveHeuristic (CbcModel & *mip*, Bonmin::OsiTMINLPInterface & *nlp*, bool *cloneNlp* = false, CouenneProblem * *couenne* = NULL)

Constructor with model and `Ipopt` problems.

7.117.2.3 Couenne::NlpSolveHeuristic::NlpSolveHeuristic (const NlpSolveHeuristic & *other*)

Copy constructor.

7.117.2.4 virtual Couenne::NlpSolveHeuristic::~NlpSolveHeuristic () [virtual]

Destructor.

7.117.3 Member Function Documentation

7.117.3.1 `virtual CbcHeuristic* Couenne::NlpSolveHeuristic::clone() const [virtual]`

Clone.

7.117.3.2 `NlpSolveHeuristic& Couenne::NlpSolveHeuristic::operator=(const NlpSolveHeuristic & rhs)`

Assignment operator.

7.117.3.3 `void Couenne::NlpSolveHeuristic::setNlp(Bonmin::OsiTMINLPInterface & nlp, bool cloneNlp = true)`

Set the nlp solver.

7.117.3.4 `void Couenne::NlpSolveHeuristic::setCouenneProblem(CouenneProblem *)`

set the couenne problem to use.

7.117.3.5 `virtual void Couenne::NlpSolveHeuristic::resetModel(CbcModel * model) [inline], [virtual]`

Does nothing.

Definition at line 53 of file BonNlpHeuristic.hpp.

7.117.3.6 `virtual int Couenne::NlpSolveHeuristic::solution(double & objectiveValue, double * newSolution) [virtual]`

Run heuristic, return 1 if a better solution than the one passed is found and 0 otherwise.

objectiveValue Best known solution in input and value of solution found in output newSolution Solution found by heuristic.

Todo Find a quicker way to get to [Couenne](#) objects, store them or something

7.117.3.7 `void Couenne::NlpSolveHeuristic::setMaxNlpInf(double value) [inline]`

set maxNlpInf.

Definition at line 61 of file BonNlpHeuristic.hpp.

7.117.3.8 `void Couenne::NlpSolveHeuristic::setNumberSolvePerLevel(int value) [inline]`

set number of nlp's solved for each given level of the tree

Definition at line 64 of file BonNlpHeuristic.hpp.

7.117.3.9 `static void Couenne::NlpSolveHeuristic::registerOptions(Ipopt::SmartPtr< Bonmin::RegisteredOptions >) [static]`

initialize options

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/heuristics/[BonNlpHeuristic.hpp](#)

7.118 Node Class Reference

```
#include <CouenneProblem.hpp>
```

Public Member Functions

- void `node` (int, double, double, double, int, int)
- void `color_vertex` (register int k)
- int `get_index` () const
- double `get_coeff` () const
- double `get_lb` () const
- double `get_ub` () const
- int `get_color` () const
- int `get_code` () const
- int `get_sign` () const
- void `bounds` (register double a, register double b)

7.118.1 Detailed Description

Definition at line 53 of file CouenneProblem.hpp.

7.118.2 Member Function Documentation

7.118.2.1 void `Node::node` (int , double , double , double , int , int)

7.118.2.2 void `Node::color_vertex` (register int k) [inline]

Definition at line 63 of file CouenneProblem.hpp.

7.118.2.3 int `Node::get_index` () const [inline]

Definition at line 64 of file CouenneProblem.hpp.

7.118.2.4 double `Node::get_coeff` () const [inline]

Definition at line 65 of file CouenneProblem.hpp.

7.118.2.5 double `Node::get_lb` () const [inline]

Definition at line 66 of file CouenneProblem.hpp.

7.118.2.6 double `Node::get_ub` () const [inline]

Definition at line 67 of file CouenneProblem.hpp.

7.118.2.7 int `Node::get_color` () const [inline]

Definition at line 68 of file CouenneProblem.hpp.

7.118.2.8 int `Node::get_code` () const [inline]

Definition at line 69 of file CouenneProblem.hpp.

7.118.2.9 int `Node::get_sign` () const [inline]

Definition at line 70 of file CouenneProblem.hpp.

7.118.2.10 void Node::bounds (register double *a*, register double *b*) [inline]

Definition at line 71 of file CouenneProblem.hpp.

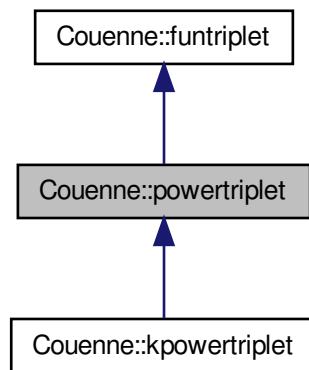
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/problem/CouenneProblem.hpp

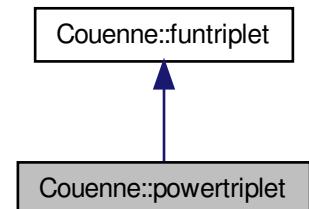
7.119 Couenne::powertriplet Class Reference

```
#include <CouenneFunTriplets.hpp>
```

Inheritance diagram for Couenne::powertriplet:



Collaboration diagram for Couenne::powertriplet:



Public Member Functions

- [powertriplet \(CouNumber exponent, bool signpower=false\)](#)
Basic constructor.
- [virtual ~powertriplet \(\)](#)
Destructor.
- [virtual CouNumber F \(CouNumber x\)](#)
- [virtual CouNumber Fp \(CouNumber x\)](#)
- [virtual CouNumber Fpp \(CouNumber x\)](#)
- [virtual CouNumber FpInv \(CouNumber x\)](#)

Protected Attributes

- [CouNumber exponent_](#)
- [bool issignpower_](#)

7.119.1 Detailed Description

Definition at line 72 of file CouenneFunTriplets.hpp.

7.119.2 Constructor & Destructor Documentation

7.119.2.1 Couenne::powertriplet::powertriplet (CouNumber exponent, bool signpower = false) [inline]

Basic constructor.

Definition at line 82 of file CouenneFunTriplets.hpp.

7.119.2.2 virtual Couenne::powertriplet::~powertriplet () [inline], [virtual]

Destructor.

Definition at line 86 of file CouenneFunTriplets.hpp.

7.119.3 Member Function Documentation

7.119.3.1 virtual CouNumber Couenne::powertriplet::F (CouNumber x) [inline], [virtual]

Implements [Couenne::funtriplet](#).

Reimplemented in [Couenne::kpowertriplet](#).

Definition at line 88 of file CouenneFunTriplets.hpp.

7.119.3.2 virtual CouNumber Couenne::powertriplet::Fp (CouNumber x) [inline], [virtual]

Implements [Couenne::funtriplet](#).

Reimplemented in [Couenne::kpowertriplet](#).

Definition at line 91 of file CouenneFunTriplets.hpp.

7.119.3.3 virtual CouNumber Couenne::powertriplet::Fpp (CouNumber x) [inline], [virtual]

Implements Couenne::funtriplet.

Reimplemented in Couenne::kpowertriplet.

Definition at line 94 of file CouenneFunTriplets.hpp.

7.119.3.4 virtual CouNumber Couenne::powertriplet::FpInv (CouNumber x) [inline], [virtual]

Implements Couenne::funtriplet.

Reimplemented in Couenne::kpowertriplet.

Definition at line 97 of file CouenneFunTriplets.hpp.

7.119.4 Member Data Documentation

7.119.4.1 CouNumber Couenne::powertriplet::exponent_ [protected]

Definition at line 76 of file CouenneFunTriplets.hpp.

7.119.4.2 bool Couenne::powertriplet::issignpower_ [protected]

Definition at line 77 of file CouenneFunTriplets.hpp.

The documentation for this class was generated from the following file:

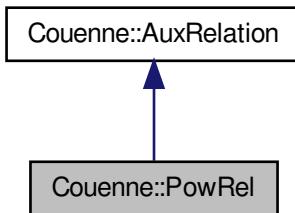
- /home/ted/COIN/trunk/Couenne/src/util/CouenneFunTriplets.hpp

7.120 Couenne::PowRel Class Reference

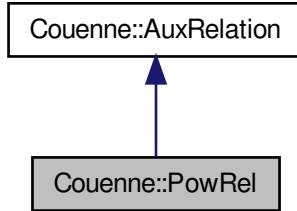
Identifies 5-tuple of the form.

```
#include <CouenneCrossConv.hpp>
```

Inheritance diagram for Couenne::PowRel:



Collaboration diagram for Couenne::PowRel:



Public Member Functions

- virtual int [findRelations \(\)](#)
- virtual void [generateCuts](#) (const OsiSolverInterface &, OsiCuts &, const CglTreeInfo=CglTreeInfo()) const

7.120.1 Detailed Description

Identifies 5-tuple of the form.

$x_j := x_i \wedge \alpha$ $x_p := x_i \wedge \beta$

and generates cuts based on the relation

$x_p = x_j \wedge \{\beta/\alpha\}$

Definition at line 125 of file CouenneCrossConv.hpp.

7.120.2 Member Function Documentation

7.120.2.1 virtual int Couenne::PowRel::findRelations () [virtual]

Implements [Couenne::AuxRelation](#).

7.120.2.2 virtual void Couenne::PowRel::generateCuts (const OsiSolverInterface & , OsiCuts & , const CglTreeInfo = CglTreeInfo ()) const [virtual]

Reimplemented from [Couenne::AuxRelation](#).

The documentation for this class was generated from the following file:

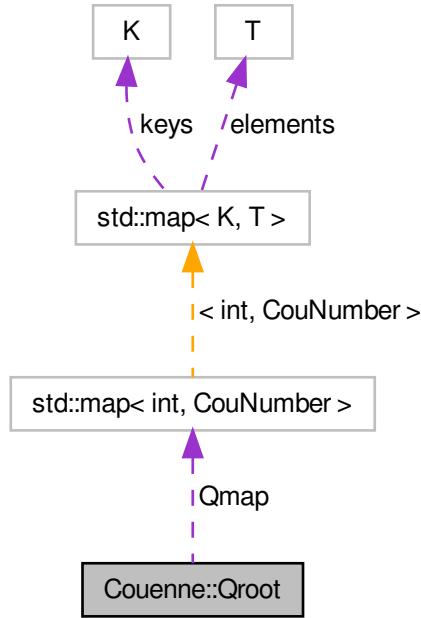
- /home/ted/COIN/trunk/Couenne/src/cut/crossconv/CouenneCrossConv.hpp

7.121 Couenne::Qroot Class Reference

class that stores result of previous calls to rootQ into a map for faster access

```
#include <CouenneRootQ.hpp>
```

Collaboration diagram for Couenne::Qroot:



Public Member Functions

- [Qroot \(\)](#)
Empty constructor – we only need the method to work on the static structure.
- [~Qroot \(\)](#)
Empty destructor.
- [CouNumber operator\(\) \(int k\)](#)
Retrieve root of Q with order = k.

Static Protected Attributes

- static std::map< int, CouNumber > Qmap
Maps an integer k with the root of $Q^k(x)$.

7.121.1 Detailed Description

class that stores result of previous calls to rootQ into a map for faster access

Definition at line 29 of file CouenneRootQ.hpp.

7.121.2 Constructor & Destructor Documentation

7.121.2.1 Couenne::Qroot::Qroot() [inline]

Empty constructor – we only need the method to work on the static structure.

Definition at line 41 of file CouenneRootQ.hpp.

7.121.2.2 Couenne::Qroot::~Qroot() [inline]

Empty destructor.

Definition at line 44 of file CouenneRootQ.hpp.

7.121.3 Member Function Documentation

7.121.3.1 CouNumber Couenne::Qroot::operator()(int k) [inline]

Retrieve root of Q with order = k.

If no such computation has been performed yet, do it here

Definition at line 49 of file CouenneRootQ.hpp.

7.121.4 Member Data Documentation

7.121.4.1 std::map<int, CouNumber> Couenne::Qroot::Qmap [static], [protected]

Maps an integer k with the root of $Q^k(x)$.

Definition at line 35 of file CouenneRootQ.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/util/CouenneRootQ.hpp

7.122 Couenne::quadElem Class Reference

```
#include <CouenneLQelems.hpp>
```

Public Member Functions

- quadElem (exprVar *i, exprVar *j, CouNumber c)
- quadElem (const quadElem &src)
- quadElem * clone ()
- exprVar * varI ()
- exprVar * varJ ()
- CouNumber coeff ()

7.122.1 Detailed Description

Definition at line 20 of file CouenneLQelems.hpp.

7.122.2 Constructor & Destructor Documentation

7.122.2.1 `Couenne::quadElem::quadElem (exprVar * i, exprVar * j, CouNumber c)` [inline]

Definition at line 29 of file CouenneLQelems.hpp.

7.122.2.2 `Couenne::quadElem::quadElem (const quadElem & src)` [inline]

Definition at line 34 of file CouenneLQelems.hpp.

7.122.3 Member Function Documentation

7.122.3.1 `quadElem* Couenne::quadElem::clone ()` [inline]

Definition at line 39 of file CouenneLQelems.hpp.

7.122.3.2 `exprVar* Couenne::quadElem::varl ()` [inline]

Definition at line 42 of file CouenneLQelems.hpp.

7.122.3.3 `exprVar* Couenne::quadElem::varJ ()` [inline]

Definition at line 43 of file CouenneLQelems.hpp.

7.122.3.4 `CouNumber Couenne::quadElem::coeff ()` [inline]

Definition at line 44 of file CouenneLQelems.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/standardize/[CouenneLQelems.hpp](#)

7.123 Couenne::QuadMap Class Reference

```
#include <CouenneLQelems.hpp>
```

Public Member Functions

- `std::map< std::pair< int, int >, CouNumber > & Map ()`
public access
- `void insert (int indI, int indJ, CouNumber coe)`
insert a pair <<int,int>, CouNumber> into a map for quadratic terms

7.123.1 Detailed Description

Definition at line 75 of file CouenneLQelems.hpp.

7.123.2 Member Function Documentation

7.123.2.1 `std::map<std::pair <int, int>, CouNumber>& Couenne::QuadMap::Map() [inline]`

public access

Definition at line 83 of file CouenneLQelems.hpp.

7.123.2.2 `void Couenne::QuadMap::insert(int indI, int indJ, CouNumber coe) [inline]`

insert a pair <<int,int>,CouNumber> into a map for quadratic terms

Definition at line 87 of file CouenneLQelems.hpp.

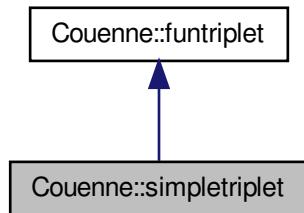
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/standardize/CouenneLQelems.hpp

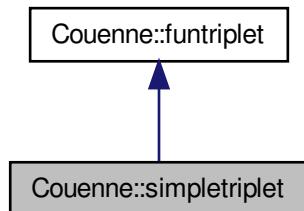
7.124 Couenne::simpletriplet Class Reference

#include <CouenneFunTriplets.hpp>

Inheritance diagram for Couenne::simpletriplet:



Collaboration diagram for Couenne::simpletriplet:



Public Member Functions

- `simpletriplet (unary_function f=NULL, unary_function fp=NULL, unary_function fpp=NULL, unary_function fpl=NULL)`
Basic constructor.
- `virtual ~simpletriplet ()`
Destructor.
- `virtual CouNumber F (CouNumber x)`
- `virtual CouNumber Fp (CouNumber x)`
- `virtual CouNumber Fpp (CouNumber x)`
- `virtual CouNumber FpInv (CouNumber x)`

Protected Attributes

- `unary_function f_`
- `unary_function fp_`
- `unary_function fpp_`
- `unary_function fpl_`

7.124.1 Detailed Description

Definition at line 40 of file CouenneFunTriplets.hpp.

7.124.2 Constructor & Destructor Documentation

7.124.2.1 Couenne::simpletriplet::simpletriplet (`unary_function f=NULL, unary_function fp=NULL, unary_function fpp=NULL, unary_function fpl=NULL`) [inline]

Basic constructor.

Definition at line 52 of file CouenneFunTriplets.hpp.

7.124.2.2 virtual Couenne::simpletriplet::~simpletriplet () [inline], [virtual]

Destructor.

Definition at line 62 of file CouenneFunTriplets.hpp.

7.124.3 Member Function Documentation

7.124.3.1 virtual CouNumber Couenne::simpletriplet::F (`CouNumber x`) [inline], [virtual]

Implements `Couenne::funtriplet`.

Definition at line 64 of file CouenneFunTriplets.hpp.

7.124.3.2 virtual CouNumber Couenne::simpletriplet::Fp (`CouNumber x`) [inline], [virtual]

Implements `Couenne::funtriplet`.

Definition at line 65 of file CouenneFunTriplets.hpp.

7.124.3.3 virtual CouNumber Couenne::simpletriplet::Fpp (CouNumber x) [inline], [virtual]

Implements [Couenne::funtriplet](#).

Definition at line 66 of file CouenneFunTriplets.hpp.

7.124.3.4 virtual CouNumber Couenne::simpletriplet::FpInv (CouNumber x) [inline], [virtual]

Implements [Couenne::funtriplet](#).

Definition at line 67 of file CouenneFunTriplets.hpp.

7.124.4 Member Data Documentation

7.124.4.1 unary_function Couenne::simpletriplet::f_ [protected]

Definition at line 44 of file CouenneFunTriplets.hpp.

7.124.4.2 unary_function Couenne::simpletriplet::fp_ [protected]

Definition at line 45 of file CouenneFunTriplets.hpp.

7.124.4.3 unary_function Couenne::simpletriplet::fpp_ [protected]

Definition at line 46 of file CouenneFunTriplets.hpp.

7.124.4.4 unary_function Couenne::simpletriplet::fpl_ [protected]

Definition at line 47 of file CouenneFunTriplets.hpp.

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/util/CouenneFunTriplets.hpp

7.125 Couenne::SmartAsl Class Reference

```
#include <BonCouenneSetup.hpp>
```

Public Member Functions

- [SmartAsl \(\)](#)
- virtual [~SmartAsl \(\)](#)

Public Attributes

- ASL * [asl](#)

7.125.1 Detailed Description

Definition at line 33 of file BonCouenneSetup.hpp.

7.125.2 Constructor & Destructor Documentation

7.125.2.1 Couenne::SmartAsl::SmartAsl() [inline]

Definition at line 36 of file BonCouenneSetup.hpp.

7.125.2.2 virtual Couenne::SmartAsl::~SmartAsl() [virtual]

7.125.3 Member Data Documentation

7.125.3.1 ASL* Couenne::SmartAsl::asl

Definition at line 35 of file BonCouenneSetup.hpp.

The documentation for this class was generated from the following file:

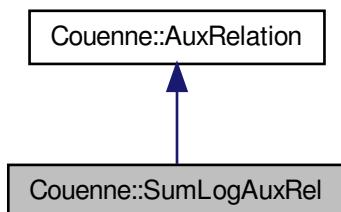
- /home/ted/COIN/trunk/Couenne/src/main/BonCouenneSetup.hpp

7.126 Couenne::SumLogAuxRel Class Reference

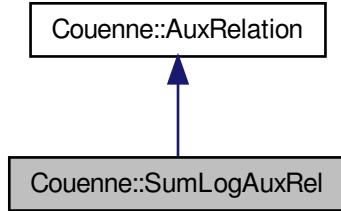
Identifies 5-ples of variables of the form.

```
#include <CouenneCrossConv.hpp>
```

Inheritance diagram for Couenne::SumLogAuxRel:



Collaboration diagram for Couenne::SumLogAuxRel:



Public Member Functions

- virtual int [findRelations \(\)](#)
- virtual void [generateCuts](#) (const OsiSolverInterface &, OsiCuts &, const CglTreeInfo=CglTreeInfo()) const

7.126.1 Detailed Description

Identifies 5-ples of variables of the form.

$x_3 := \log x_1$ $x_4 := \log x_2$ $x_5 := x_1 x_2$ in $[l, u]$

and generates a cut

$x_3 + x_4$ in $[\max\{0, \log l\}, \max\{0, \log u\}]$.

This has to be repeatedly generated, even when $l=u$ (l and/or u could change in other nodes).

Definition at line 58 of file CouenneCrossConv.hpp.

7.126.2 Member Function Documentation

7.126.2.1 virtual int Couenne::SumLogAuxRel::findRelations () [virtual]

Implements [Couenne::AuxRelation](#).

7.126.2.2 virtual void Couenne::SumLogAuxRel::generateCuts (const OsiSolverInterface & , OsiCuts & , const CglTreeInfo = CglTreeInfo()) const [virtual]

Reimplemented from [Couenne::AuxRelation](#).

The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/cut/crossconv/CouenneCrossConv.hpp

7.127 Couenne::t_chg_bounds Class Reference

status of lower/upper bound of a variable, to be checked/modified in bound tightening

```
#include <CouenneTypes.hpp>
```

Public Types

- enum **ChangeStatus** { **UNCHANGED** = 0, **CHANGED** = 1, **EXACT** = 2 }

Public Member Functions

- **t_chg_bounds ()**
- **t_chg_bounds (const t_chg_bounds &src)**
- const char & **lower () const**
- const char & **upper () const**
- void **setLower (ChangeStatus lower)**
- void **setUpper (ChangeStatus upper)**
- void **setLowerBits (char lower)**
- void **setUpperBits (char upper)**
- **t_chg_bounds operator= (const t_chg_bounds &src)**

7.127.1 Detailed Description

status of lower/upper bound of a variable, to be checked/modified in bound tightening

Definition at line 66 of file CouenneTypes.hpp.

7.127.2 Member Enumeration Documentation

7.127.2.1 enum Couenne::t_chg_bounds::ChangeStatus

Enumerator:

- UNCHANGED**
- CHANGED**
- EXACT**

Definition at line 69 of file CouenneTypes.hpp.

7.127.3 Constructor & Destructor Documentation

7.127.3.1 Couenne::t_chg_bounds::t_chg_bounds () [inline]

Definition at line 75 of file CouenneTypes.hpp.

7.127.3.2 Couenne::t_chg_bounds::t_chg_bounds (const t_chg_bounds & src) [inline]

Definition at line 79 of file CouenneTypes.hpp.

7.127.4 Member Function Documentation

7.127.4.1 const char& Couenne::t_chg_bounds::lower () const [inline]

Definition at line 83 of file CouenneTypes.hpp.

7.127.4.2 const char& Couenne::t_chg_bounds::upper() const [inline]

Definition at line 84 of file CouenneTypes.hpp.

7.127.4.3 void Couenne::t_chg_bounds::setLower(ChangeStatus lower) [inline]

Definition at line 85 of file CouenneTypes.hpp.

7.127.4.4 void Couenne::t_chg_bounds::setUp(ChangeStatus upper) [inline]

Definition at line 86 of file CouenneTypes.hpp.

7.127.4.5 void Couenne::t_chg_bounds::setLowerBits(char lower) [inline]

Definition at line 87 of file CouenneTypes.hpp.

7.127.4.6 void Couenne::t_chg_bounds::setUpBits(char upper) [inline]

Definition at line 88 of file CouenneTypes.hpp.

7.127.4.7 t_chg_bounds Couenne::t_chg_bounds::operator=(const t_chg_bounds & src) [inline]

Definition at line 89 of file CouenneTypes.hpp.

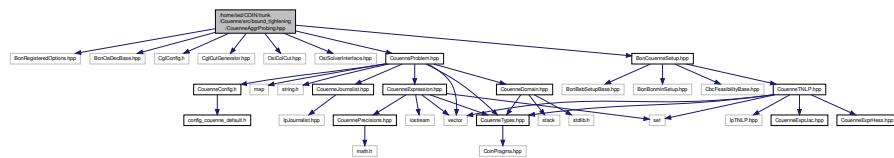
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Couenne/src/expression/CouenneTypes.hpp

8 File Documentation

8.1 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneAggrProbing.hpp File Reference

```
#include "BonRegisteredOptions.hpp"
#include "BonOaDecBase.hpp"
#include "CglConfig.h"
#include "CglCutGenerator.hpp"
#include "OsiColCut.hpp"
#include "OsiSolverInterface.hpp"
#include "CouenneProblem.hpp"
#include "BonCouenneSetup.hpp"
Include dependency graph for CouenneAggrProbing.hpp:
```



Classes

- class [Couenne::CouenneAggrProbing](#)

Cut Generator for aggressive BT; i.e., an aggressive probing.

Namespaces

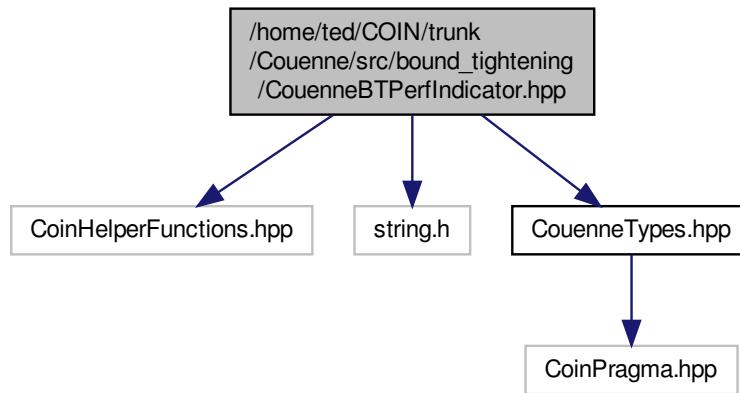
- namespace [Couenne](#)
general include file for different compilers

8.2 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneBTPerfIndicator.hpp File Reference

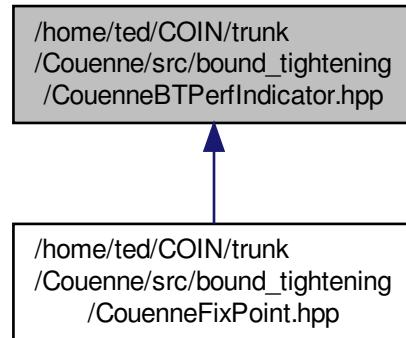
```
#include "CoinHelperFunctions.hpp"
#include <string.h>
```

```
#include "CouenneTypes.hpp"
```

Include dependency graph for CouenneBTPerfIndicator.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::CouenneBTPerfIndicator](#)

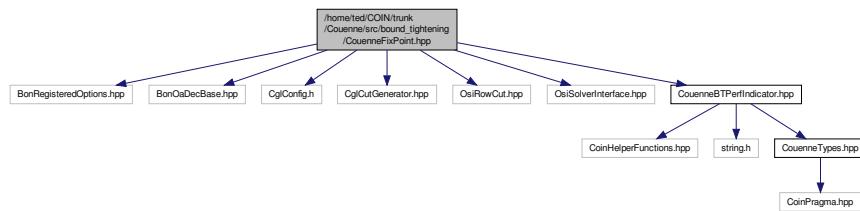
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.3 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneFixPoint.hpp File Reference

```
#include "BonRegisteredOptions.hpp"
#include "BonOaDecBase.hpp"
#include "CglConfig.h"
#include "CglCutGenerator.hpp"
#include "OsiRowCut.hpp"
#include "OsiSolverInterface.hpp"
#include "CouenneBTPerfIndicator.hpp"
```

Include dependency graph for CouenneFixPoint.hpp:



Classes

- class [Couenne::CouenneFixPoint](#)
Cut Generator for FBBT fixpoint.

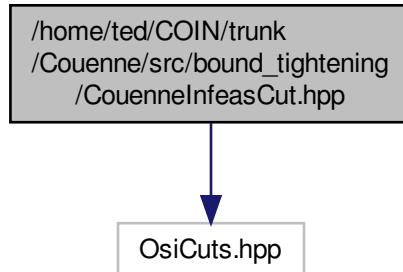
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.4 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneInfeasCut.hpp File Reference

```
#include "OsiCuts.hpp"
```

Include dependency graph for CouenneInfeasCut.hpp:



Functions

- void [WipeMakeInfeas](#) (OsiCuts &cs)
Add a fictitious cut $1 \leq x_0 \leq -1$ as a signal to the node solver that this node is deemed infeasible by this cut generator (most likely a bound tightener).
- bool [isWiped](#) (OsiCuts &cs)
Check whether the previous cut generators have added an infeasible cut.

8.4.1 Function Documentation

8.4.1.1 void WipeMakeInfeas (OsiCuts & cs)

Add a fictitious cut $1 \leq x_0 \leq -1$ as a signal to the node solver that this node is deemed infeasible by this cut generator (most likely a bound tightener).

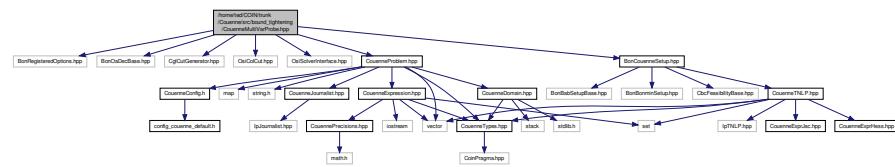
8.4.1.2 bool isWiped (OsiCuts & cs)

Check whether the previous cut generators have added an infeasible cut.

8.5 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneMultiVarProbe.hpp File Reference

```
#include "BonRegisteredOptions.hpp"
#include "BonOaDecBase.hpp"
#include "CglCutGenerator.hpp"
#include "OsiColCut.hpp"
#include "OsiSolverInterface.hpp"
#include "CouenneProblem.hpp"
#include "BonCouenneSetup.hpp"
```

Include dependency graph for CouenneMultiVarProbe.hpp:



Classes

- class [Couenne::CouenneMultiVarProbe](#)

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.6 /home/ted/COIN/trunk/Couenne/src/bound_tightening/CouenneSparseBndVec.hpp File Reference

Classes

- class [Couenne::CouenneSparseBndVec< T >](#)

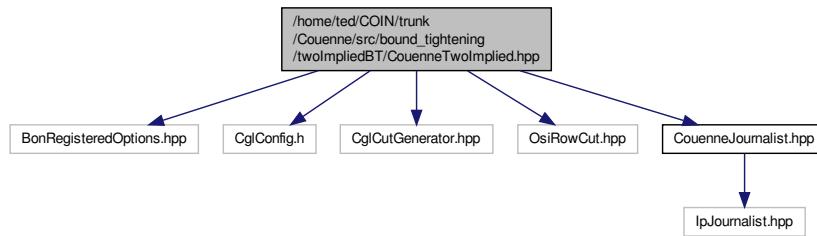
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.7 /home/ted/COIN/trunk/Couenne/src/bound_tightening/twoImpliedBT/CouenneTwoImplied.hpp File Reference

```
#include "BonRegisteredOptions.hpp"
#include "CglConfig.h"
#include "CglCutGenerator.hpp"
#include "OsiRowCut.hpp"
#include "CouenneJournalist.hpp"
```

Include dependency graph for CouenneTwoImplied.hpp:



Classes

- class [Couenne::CouenneTwoImplied](#)
Cut Generator for implied bounds derived from pairs of linear (in)equalities.

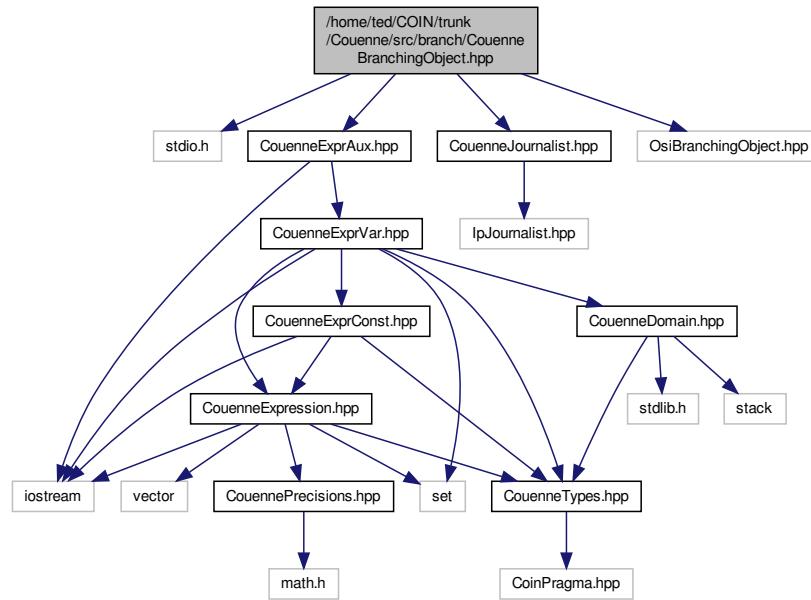
Namespaces

- namespace [Ipopt](#)
- namespace [Couenne](#)
general include file for different compilers

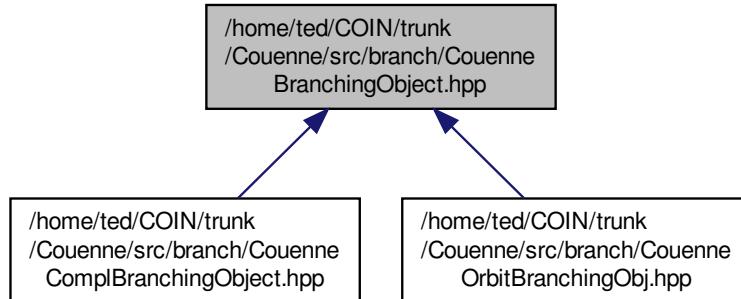
8.8 /home/ted/COIN/trunk/Couenne/src/branch/CouenneBranchingObject.hpp File Reference

```
#include "stdio.h"
#include "CouenneExprAux.hpp"
#include "CouenneJournalist.hpp"
#include "OsiBranchingObject.hpp"
```

Include dependency graph for CouenneBranchingObject.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::CouenneBranchingObject](#)
"Spatial" branching object.

Namespaces

- namespace **Couenne**
general include file for different compilers

Macros

- #define COUENNE_CROP 1
- #define COUENNE_LCROP (1e2*COUENNE_CROP)
- #define COUENNE_LARGE_INTERVAL 1e4
- #define COUENNE_NEAR_BOUND 1e-2

8.8.1 Macro Definition Documentation

8.8.1.1 #define COUENNE_CROP 1

Definition at line 25 of file CouenneBranchingObject.hpp.

8.8.1.2 #define COUENNE_LCROP (1e2*COUENNE_CROP)

Definition at line 26 of file CouenneBranchingObject.hpp.

8.8.1.3 #define COUENNE_LARGE_INTERVAL 1e4

Definition at line 28 of file CouenneBranchingObject.hpp.

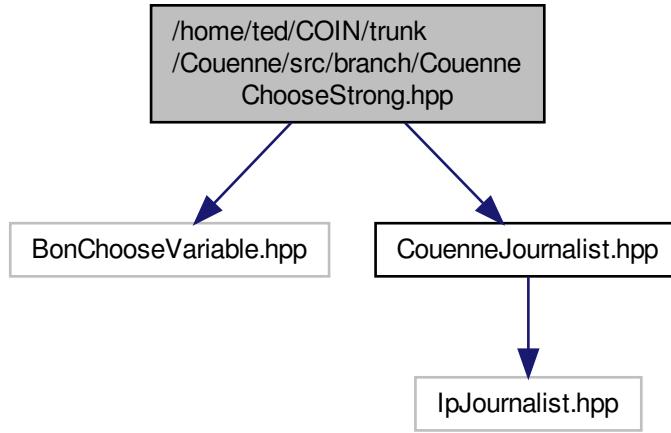
8.8.1.4 #define COUENNE_NEAR_BOUND 1e-2

Definition at line 29 of file CouenneBranchingObject.hpp.

8.9 /home/ted/COIN/trunk/Couenne/src/branch/CouenneChooseStrong.hpp File Reference

```
#include "BonChooseVariable.hpp"
#include "CouenneJournalist.hpp"
```

Include dependency graph for CouenneChooseStrong.hpp:



Classes

- class [Couenne::CouenneChooseStrong](#)

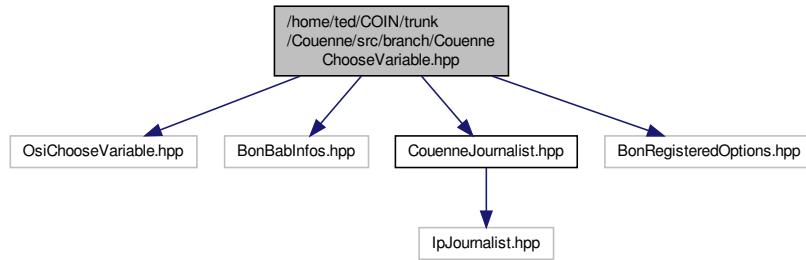
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.10 /home/ted/COIN/trunk/Couenne/src/branch/CouenneChooseVariable.hpp File Reference

```
#include "OsiChooseVariable.hpp"
#include "BonBabInfos.hpp"
#include "CouenneJournalist.hpp"
#include "BonRegisteredOptions.hpp"
```

Include dependency graph for CouenneChooseVariable.hpp:



Classes

- class [Couenne::CouenneChooseVariable](#)
Choose a variable for branching.

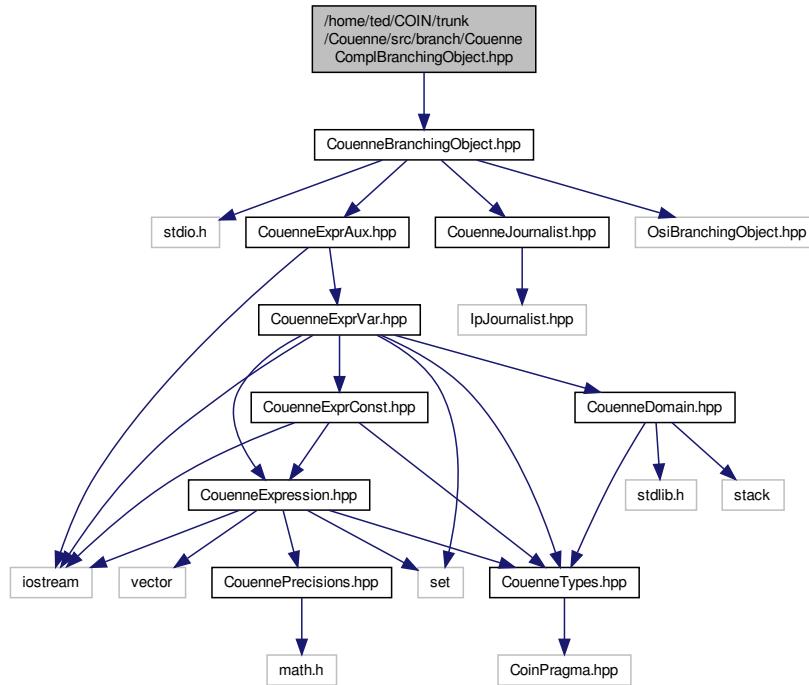
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.11 /home/ted/COIN/trunk/Couenne/src/branch/CouenneComplBranchingObject.hpp File Reference

```
#include "CouenneBranchingObject.hpp"
```

Include dependency graph for CouenneComplBranchingObject.hpp:



Classes

- class [Couenne::CouenneComplBranchingObject](#)
"Spatial" branching object for complementarity constraints.

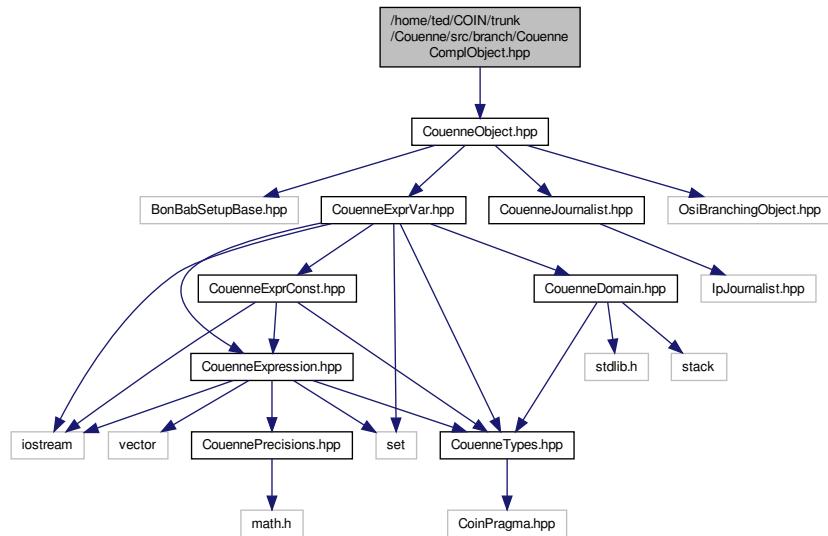
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.12 /home/ted/COIN/trunk/Couenne/src/branch/CouenneComplObject.hpp File Reference

```
#include "CouenneObject.hpp"
```

Include dependency graph for CouenneComplObject.hpp:



Classes

- class [Couenne::CouenneComplObject](#)
OsiObject for complementarity constraints $x_1x_2 \geq, \leq, = 0$.

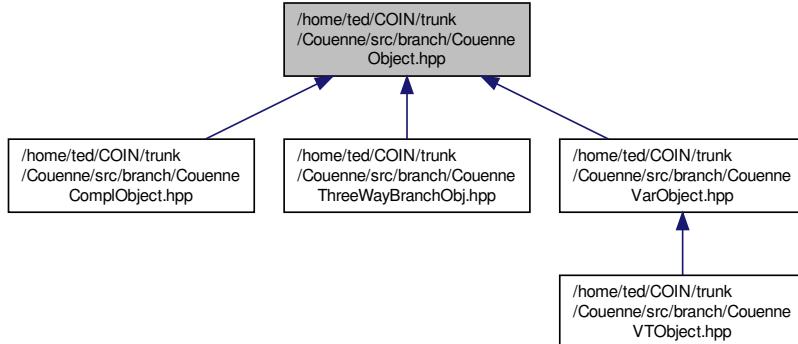
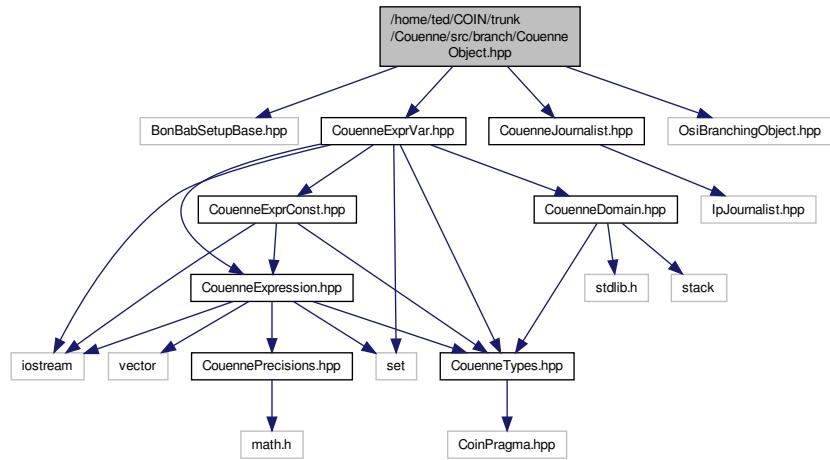
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.13 /home/ted/COIN/trunk/Couenne/src/branch/CouenneObject.hpp File Reference

```
#include "BonBabSetupBase.hpp"
#include "CouenneExprVar.hpp"
#include "CouenneJournalist.hpp"
#include "OsiBranchingObject.hpp"
```

Include dependency graph for CouenneObject.hpp:



Classes

- class [Couenne::CouenneObject](#)
OsiObject for auxiliary variables \$w=f(x)\$.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Macros

- `#define AGGR_MUL 2`
- `#define THRES_ZERO_SYMM 0.8`

Enumerations

- `enum { Couenne::TWO_LEFT, Couenne::TWO_RIGHT, Couenne::TWO RAND, Couenne::THREE_LEFT, Couenne::THREE_CENTER, Couenne::THREE_RIGHT, Couenne::THREE RAND, Couenne::BRANCH_NONE }`

Define what kind of branching (two- or three-way) and where to start from: left, (center,) or right.

Functions

- `CouNumber Couenne::minMaxDelta (funtriplet *ft, CouNumber lb, CouNumber ub)`
- `CouNumber Couenne::maxHeight (funtriplet *ft, CouNumber lb, CouNumber ub)`

Variables

- `const CouNumber Couenne::default_alpha = 0.25`
- `const CouNumber Couenne::default_clamp = 0.2`
- `const CouNumber Couenne::max_pseudocost = 1000.`
- `const double Couenne::large_bound = 1e9`
if |branching point| > this, change it
- `const CouNumber Couenne::closeToBounds = .05`

8.13.1 Macro Definition Documentation

8.13.1.1 `#define AGGR_MUL 2`

Definition at line 30 of file CouenneObject.hpp.

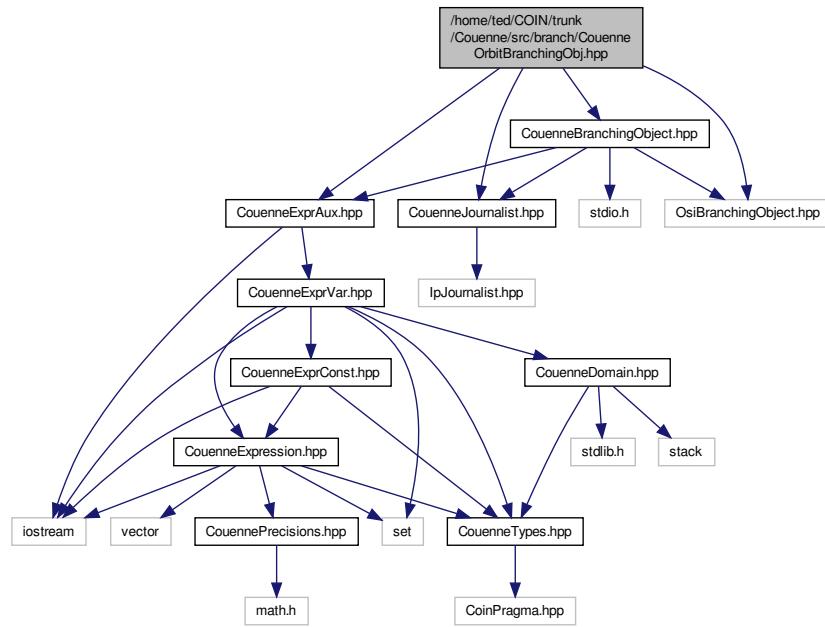
8.13.1.2 `#define THRES_ZERO_SYMM 0.8`

Definition at line 31 of file CouenneObject.hpp.

8.14 /home/ted/COIN/trunk/Couenne/src/branch/CouenneOrbitBranchingObj.hpp File Reference

```
#include "CouenneExprAux.hpp"
#include "CouenneJournalist.hpp"
#include "OsiBranchingObject.hpp"
#include "CouenneBranchingObject.hpp"
```

Include dependency graph for CouenneOrbitBranchingObj.hpp:



Classes

- class [Couenne::CouenneOrbitBranchingObj](#)
"Spatial" branching object.

Namespaces

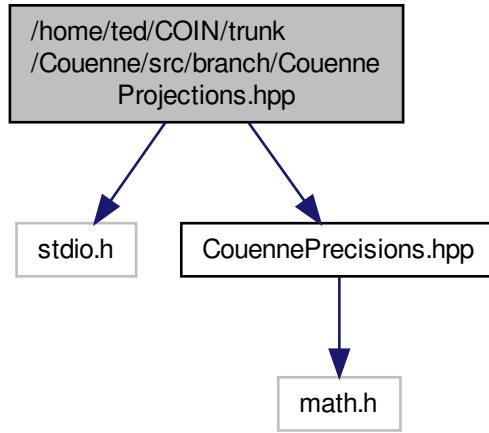
- namespace [Couenne](#)
general include file for different compilers

8.15 /home/ted/COIN/trunk/Couenne/src/branch/CouenneOrbitObj.hpp File Reference

8.16 /home/ted/COIN/trunk/Couenne/src/branch/CouenneProjections.hpp File Reference

```
#include <stdio.h>
#include "CouennePrecisions.hpp"
```

Include dependency graph for CouenneProjections.hpp:



Namespaces

- namespace [Couenne](#)
general include file for different compilers

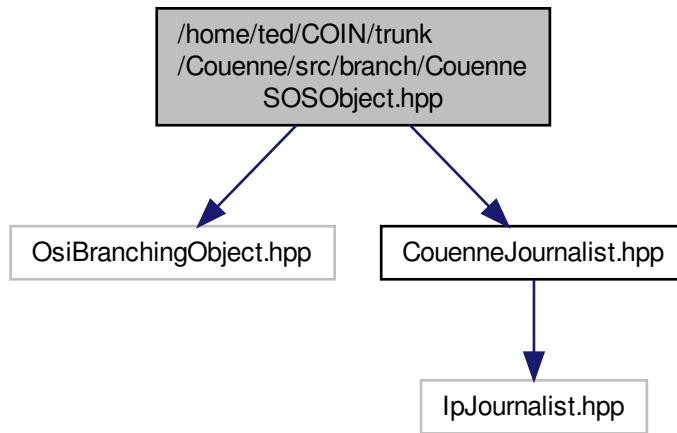
Functions

- CouNumber [Couenne::project](#) (CouNumber a, CouNumber b, CouNumber c, CouNumber x0, CouNumber y0, CouNumber lb, CouNumber ub, int sign, CouNumber *xp=NULL, CouNumber *yp=NULL)
Compute projection of point (x0, y0) on the segment defined by line ax + by + c <= 0 (sign provided by parameter sign) and bounds [lb, ub] on x.
- CouNumber [Couenne::projectSeg](#) (CouNumber x0, CouNumber y0, CouNumber x1, CouNumber y1, CouNumber x2, CouNumber y2, int sign, CouNumber *xp=NULL, CouNumber *yp=NULL)
Compute projection of point (x0, y0) on the segment defined by two points (x1,y1), (x2, y2) – sign provided by parameter sign.

8.17 /home/ted/COIN/trunk/Couenne/src/branch/CouenneSOSObject.hpp File Reference

```
#include "OsiBranchingObject.hpp"
#include "CouenneJournalist.hpp"
```

Include dependency graph for CouenneSOSObject.hpp:



Classes

- class [Couenne::CouenneSOSBranchingObject](#)
- class [Couenne::CouenneSOSObject](#)

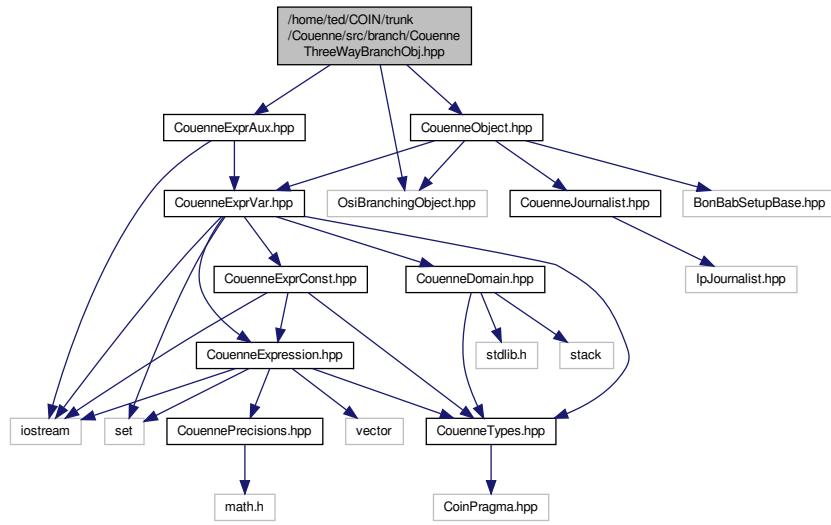
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.18 /home/ted/COIN/trunk/Couenne/src/branch/CouenneThreeWayBranchObj.hpp File Reference

```
#include "OsiBranchingObject.hpp"
#include "CouenneExprAux.hpp"
#include "CouenneObject.hpp"
```

Include dependency graph for CouenneThreeWayBranchObj.hpp:



Classes

- class [Couenne::CouenneThreeWayBranchObj](#)
Spatial, three-way branching object.

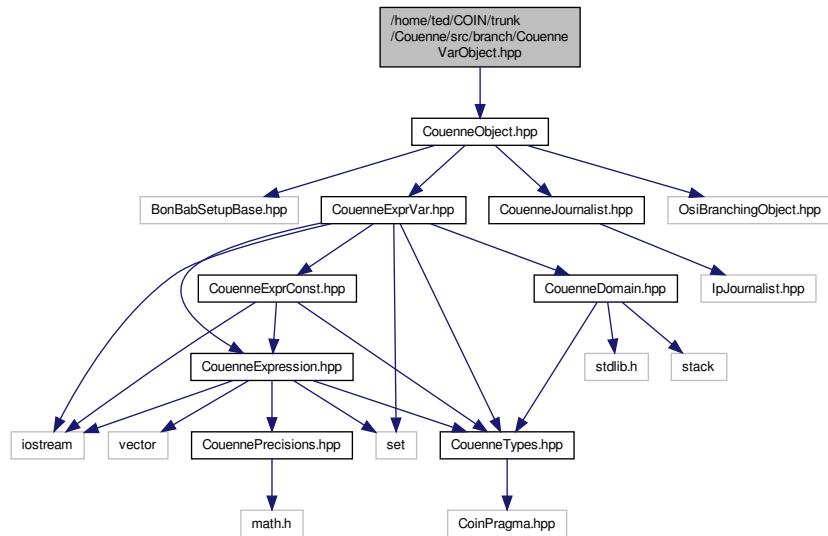
Namespaces

- namespace [Couenne](#)
general include file for different compilers

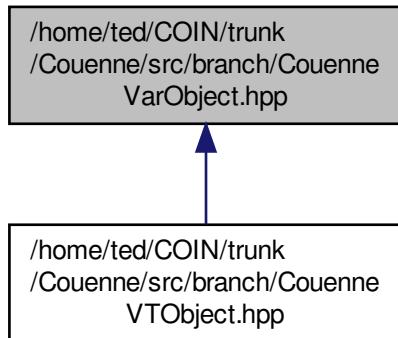
8.19 /home/ted/COIN/trunk/Couenne/src/branch/CouenneVarObject.hpp File Reference

```
#include "CouenneObject.hpp"
```

Include dependency graph for CouenneVarObject.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::CouenneVarObject](#)

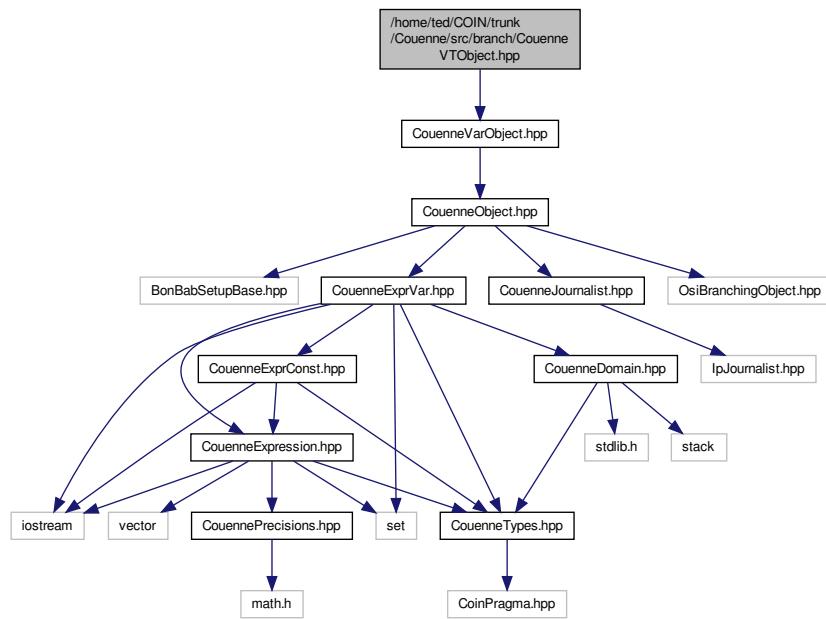
OsiObject for variables in a MINLP.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.20 /home/ted/COIN/trunk/Couenne/src/branch/CouenneVTOObject.hpp File Reference

```
#include "CouenneVarObject.hpp"
Include dependency graph for CouenneVTOObject.hpp:
```



Classes

- class [Couenne::CouenneVTOObject](#)
OsiObject for violation transfer on variables in a MINLP.

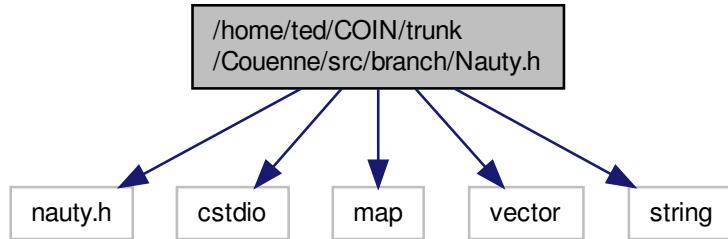
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.21 /home/ted/COIN/trunk/Couenne/src/branch/Nauty.h File Reference

```
#include "nauty.h"
```

```
#include <cstdio>
#include <map>
#include <vector>
#include <string>
Include dependency graph for Nauty.h:
```

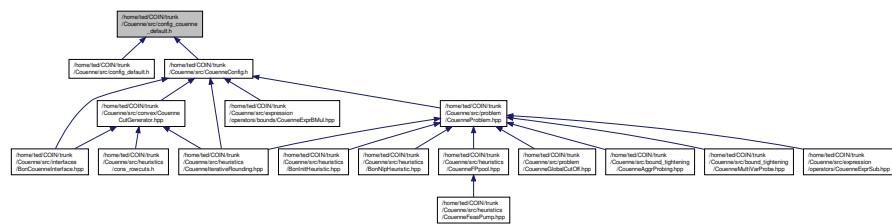


Classes

- class [Nauty](#)

8.22 /home/ted/COIN/trunk/Couenne/src/config_couenne_default.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define COUENNE_VERSION "trunk"`
- `#define COUENNE_VERSION_MAJOR 9999`
- `#define COUENNE_VERSION_MINOR 9999`
- `#define COUENNE_VERSION_RELEASE 9999`

8.22.1 Macro Definition Documentation

8.22.1.1 `#define COUENNE_VERSION "trunk"`

Definition at line 8 of file `config_couenne_default.h`.

8.22.1.2 #define COUENNE_VERSION_MAJOR 9999

Definition at line 11 of file config_couenne_default.h.

8.22.1.3 #define COUENNE_VERSION_MINOR 9999

Definition at line 14 of file config_couenne_default.h.

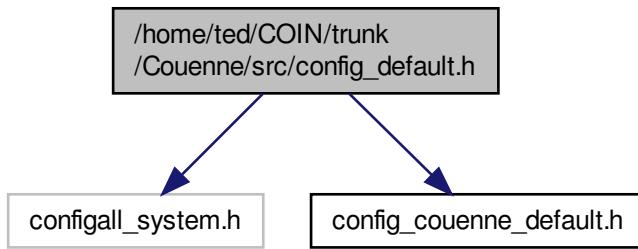
8.22.1.4 #define COUENNE_VERSION_RELEASE 9999

Definition at line 17 of file config_couenne_default.h.

8.23 /home/ted/COIN/trunk/Couenne/src/config_default.h File Reference

```
#include "configall_system.h"
#include "config_couenne_default.h"
```

Include dependency graph for config_default.h:



Macros

- #define COIN_COUENNE_CHECKLEVEL 0
- #define COIN_COUENNE_VERBOSITY 0

8.23.1 Macro Definition Documentation

8.23.1.1 #define COIN_COUENNE_CHECKLEVEL 0

Definition at line 14 of file config_default.h.

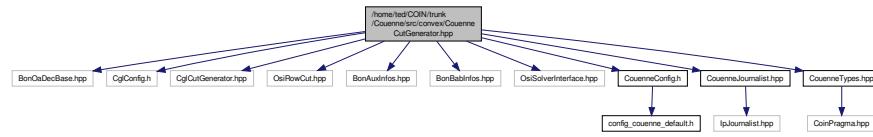
8.23.1.2 #define COIN_COUENNE_VERBOSITY 0

Definition at line 17 of file config_default.h.

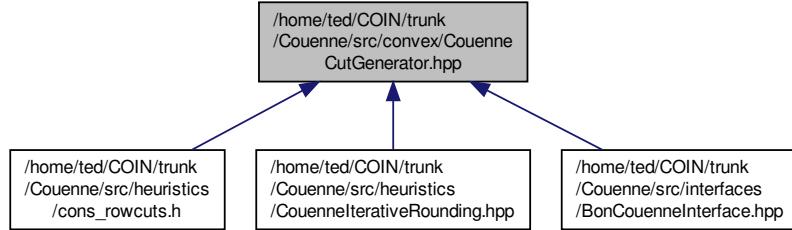
8.24 /home/ted/COIN/trunk/Couenne/src/convex/CouenneCutGenerator.hpp File Reference

```
#include "BonOaDecBase.hpp"
#include "CglConfig.h"
#include "CglCutGenerator.hpp"
#include "OsiRowCut.hpp"
#include "BonAuxInfos.hpp"
#include "BonBabInfos.hpp"
#include "OsiSolverInterface.hpp"
#include "CouenneConfig.h"
#include "CouenneJournalist.hpp"
#include "CouenneTypes.hpp"
```

Include dependency graph for CouenneCutGenerator.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::CouenneCutGenerator](#)
Cut Generator for linear convexifications.

Namespaces

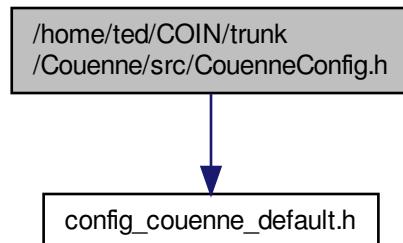
- namespace [Ipopt](#)
- namespace [Bonmin](#)
- namespace [Couenne](#)
general include file for different compilers

Functions

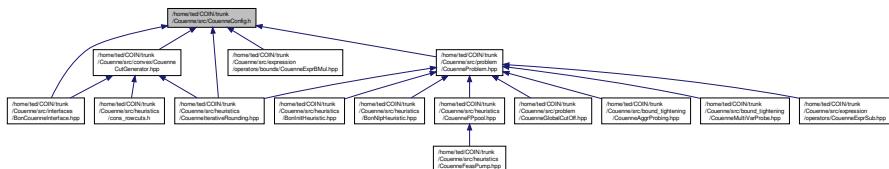
- void **Couenne::sparse2dense** (int ncols, t_chg_bounds *chg_bds, int *&changed, int &nchanged)
translate sparse to dense vector (should be replaced)

8.25 /home/ted/COIN/trunk/Couenne/src/CouenneConfig.h File Reference

```
#include "config_couenne_default.h"
Include dependency graph for CouenneConfig.h:
```



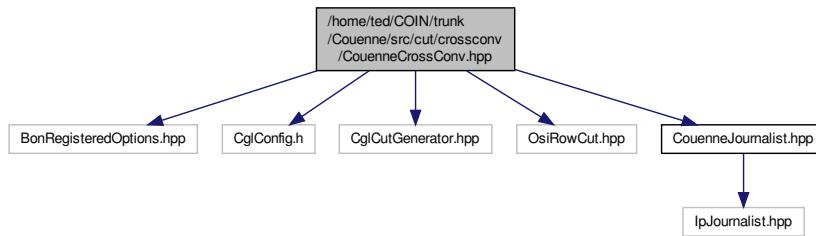
This graph shows which files directly or indirectly include this file:



8.26 /home/ted/COIN/trunk/Couenne/src/cut/crossconv/CouenneCrossConv.hpp File Reference

```
#include "BonRegisteredOptions.hpp"
#include "CglConfig.h"
#include "CglCutGenerator.hpp"
#include "OsiRowCut.hpp"
#include "CouenneJournalist.hpp"
```

Include dependency graph for CouenneCrossConv.hpp:



Classes

- class [Couenne::AuxRelation](#)
Base class definition for relations between auxiliaries.
- class [Couenne::SumLogAuxRel](#)
Identifies 5-ples of variables of the form.
- class [Couenne::MultiProdRel](#)
Identifies 5-ples of variables of the form.
- class [Couenne::BiProdDivRel](#)
Identifies 5-tuple of the form.
- class [Couenne::PowRel](#)
Identifies 5-tuple of the form.
- class [Couenne::CouenneCrossConv](#)
Cut Generator that uses relationships between auxiliaries.

Namespaces

- namespace [Ipopt](#)
- namespace [Couenne](#)
general include file for different compilers

8.27 /home/ted/COIN/trunk/Couenne/src/cut/ellipcuts/CouenneEllipCuts.hpp File Reference

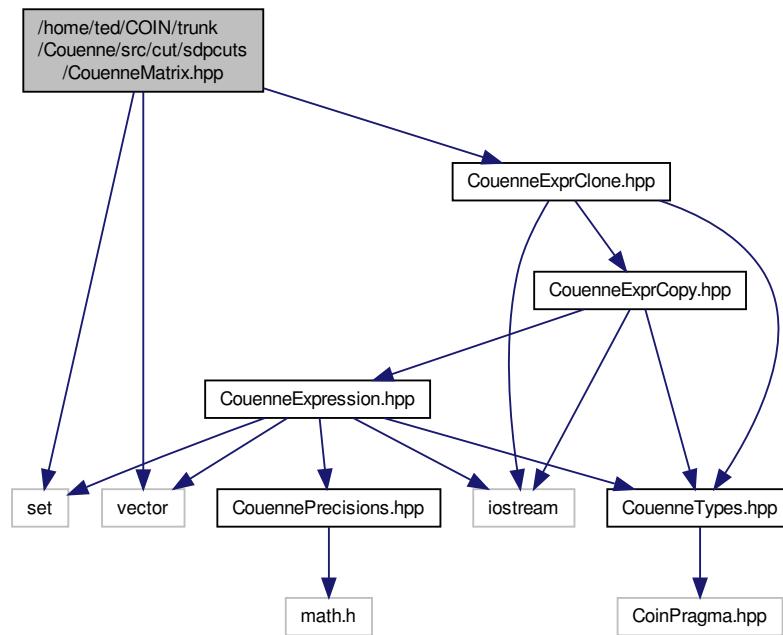
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.28 /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneMatrix.hpp File Reference

```
#include <set>
#include <vector>
#include "CouenneExprClone.hpp"
```

Include dependency graph for CouenneMatrix.hpp:



Classes

- class [Couenne::CouenneScalar](#)
- class [Couenne::CouenneSparseVector](#)
- struct [Couenne::CouenneSparseVector::compare_scalars](#)
- class [Couenne::CouenneExprMatrix](#)
- struct [Couenne::CouenneExprMatrix::compare_pair_ind](#)

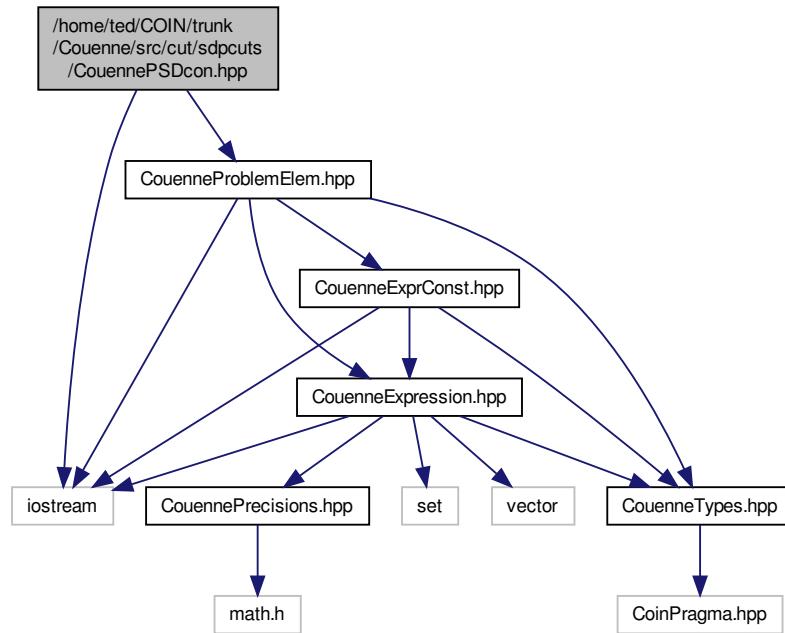
Namespaces

- namespace [Couenne](#)
general include file for different compilers

Functions

- bool [Couenne::operator<](#) (const CouenneScalar &first, const CouenneScalar &second)

Include dependency graph for CouennePSDcon.hpp:



Classes

- class [Couenne::CouennePSDcon](#)
Class to represent positive semidefinite constraints //////////////////.

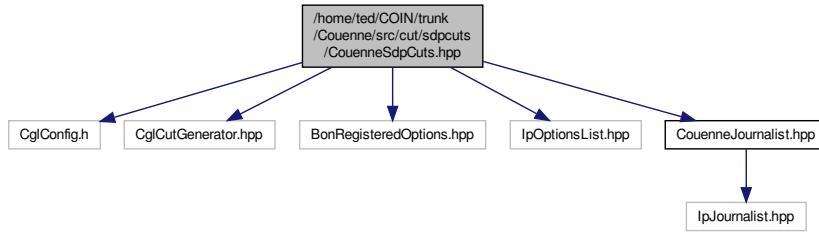
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.30 /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/CouenneSdpCuts.hpp File Reference

```
#include "CglConfig.h"
#include "CglCutGenerator.hpp"
#include "BonRegisteredOptions.hpp"
#include "IpOptionsList.hpp"
#include "CouenneJournalist.hpp"
```

Include dependency graph for CouenneSdpCuts.hpp:



Classes

- class `Couenne::CouenneSdpCuts`

These are cuts of the form.

Namespaces

- namespace **Couenne**
general include file for different compilers

8.31 /home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/dsyevx_wrapper.hpp File Reference

Functions

- int **dsyevx_interface** (int n, double *A, int &m, double *&w, double *&z, double tolerance, double lb_ev, double ub_ev, int firstidx, int lastidx)

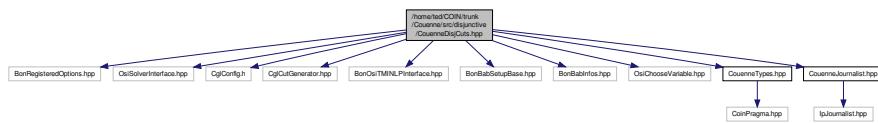
8.31.1 Function Documentation

8.31.1.1 int dsyevx_interface (int n , double * A , int & m , double *& w , double *&& z , double tolerance, double lb_ev , double ub_ev , int $firstidx$, int $lastidx$)

8.32 /home/ted/COIN/trunk/Couenne/src/disjunctive/CouenneDisjCuts.hpp File Reference

```
#include "BonRegisteredOptions.hpp"
#include "OsiSolverInterface.hpp"
#include "CglConfig.h"
#include "CglCutGenerator.hpp"
#include "BonOsiTMINLPInterface.hpp"
#include "BonBabSetupBase.hpp"
#include "BonBabInfos.hpp"
#include "OsiChooseVariable.hpp"
#include "CouenneTypes.hpp"
#include "CouenneJournalist.hpp"
```

Include dependency graph for CouenneDisjCuts.hpp:



Classes

- class [Couenne::CouenneDisjCuts](#)
Cut Generator for linear convexifications.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Enumerations

- enum { [Couenne::COUENNE_INFEASIBLE](#), [Couenne::COUENNE_TIGHTENED](#), [Couenne::COUENNE_FEASIBLE](#) }

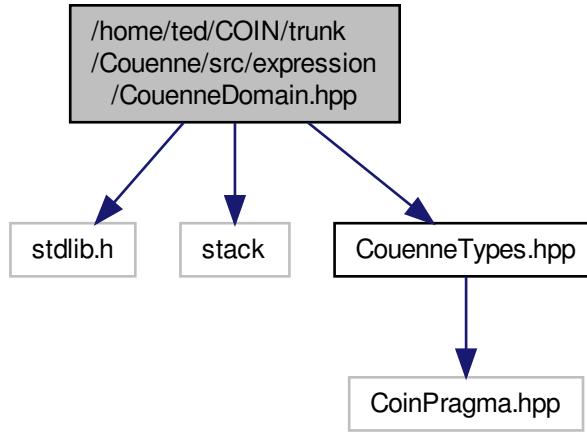
Functions

- void [Couenne::CoinInvN](#) (register const double *orig, register int n, register double *inverted)
invert all contents
- void [Couenne::CoinCopyDisp](#) (register const int *src, register int num, register int *dst, register int displacement)
a CoinCopyN with a += on each element

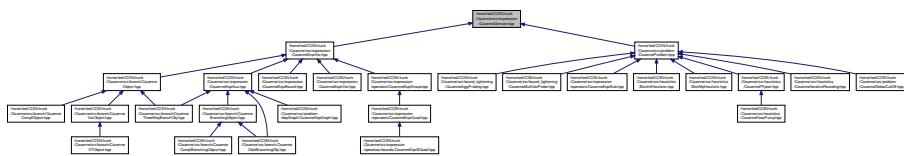
8.33 /home/ted/COIN/trunk/Couenne/src/expression/CouenneDomain.hpp File Reference

```
#include <stdlib.h>
#include <stack>
#include "CouenneTypes.hpp"
```

Include dependency graph for CouenneDomain.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::DomainPoint](#)
Define a point in the solution space and the bounds around it.
- class [Couenne::Domain](#)
Define a dynamic point+bounds, with a way to save and restore previous points+bounds through a LIFO structure.

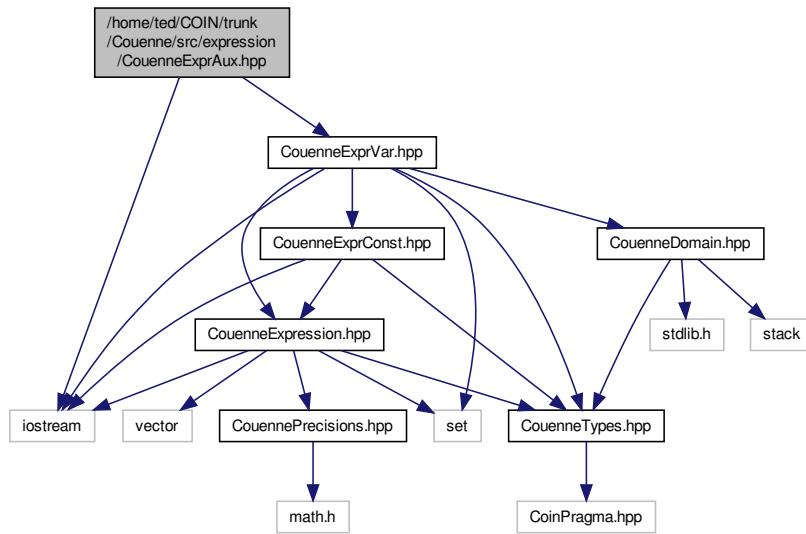
Namespaces

- namespace [Osi](#)
- namespace [Couenne](#)
general include file for different compilers

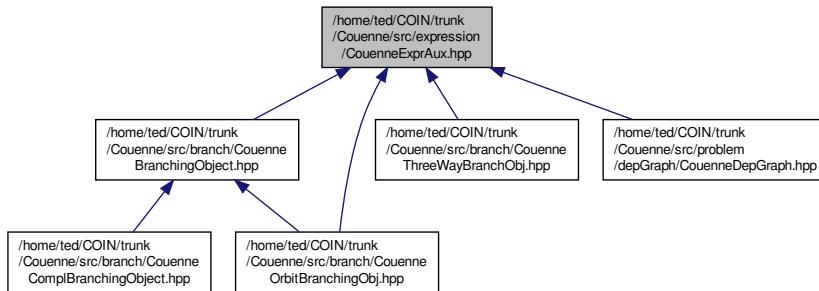
8.34 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprAux.hpp File Reference

```
#include <iostream>
#include "CouenneExprVar.hpp"
```

Include dependency graph for CouenneExprAux.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprAux](#)
Auxiliary variable.
- struct [Couenne::compExpr](#)
Structure for comparing expressions.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

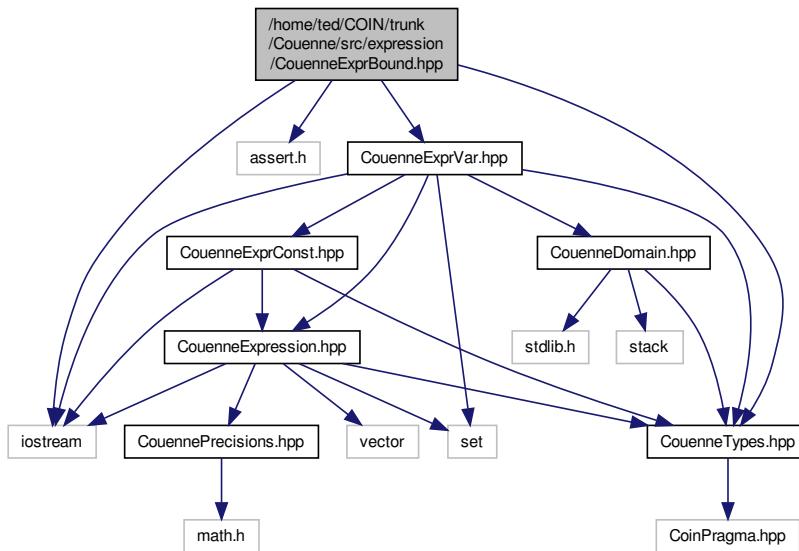
Functions

- void [Couenne::draw_cuts](#) (OsiCuts &, const CouenneCutGenerator *, int, expression *, expression *)

allow to draw function within intervals and cuts introduced

8.35 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprBound.hpp File Reference

```
#include <iostream>
#include <assert.h>
#include "CouenneTypes.hpp"
#include "CouenneExprVar.hpp"
Include dependency graph for CouenneExprBound.hpp:
```



Classes

- class [Couenne::exprLowerBound](#)

These are bound expression classes.
- class [Couenne::exprUpperBound](#)

upper bound

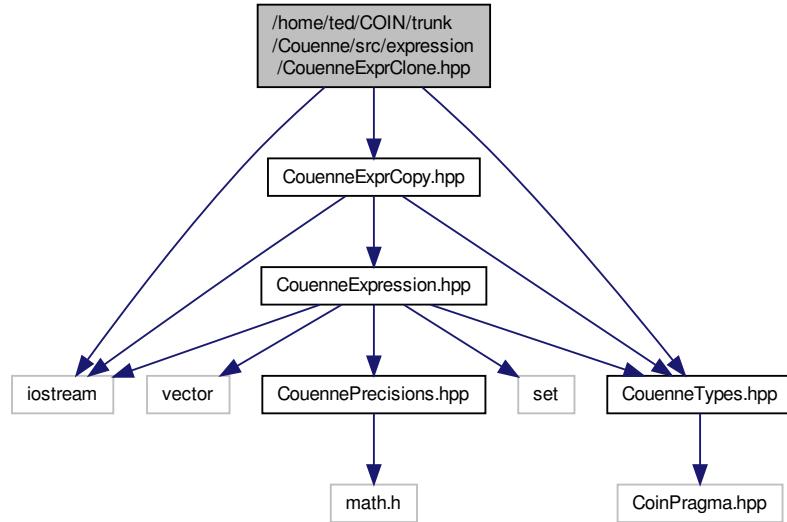
Namespaces

- namespace [Couenne](#)

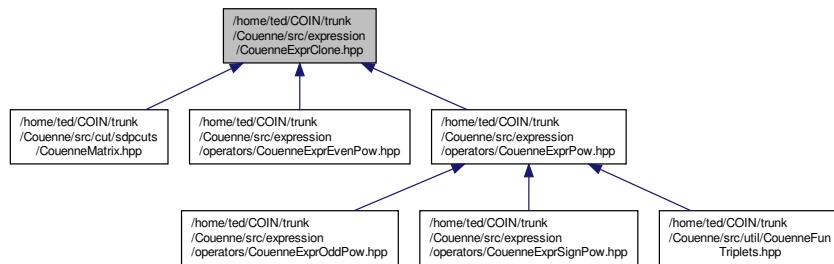
general include file for different compilers

8.36 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprClone.hpp File Reference

```
#include <iostream>
#include "CouenneTypes.hpp"
#include "CouenneExprCopy.hpp"
Include dependency graph for CouenneExprClone.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

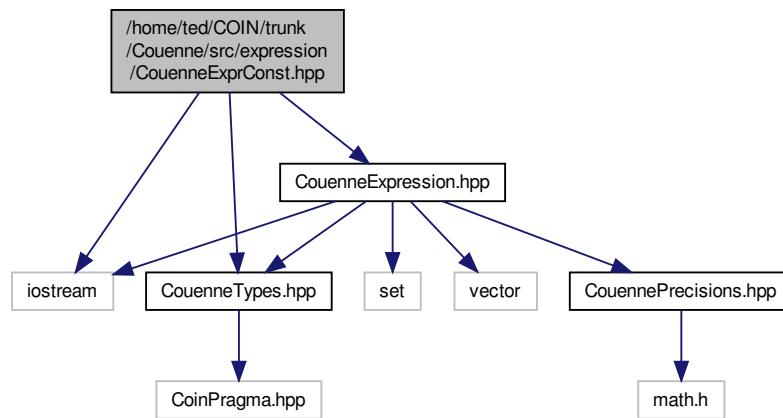
- class `Couenne::exprClone`
expression clone (points to another expression)

Namespaces

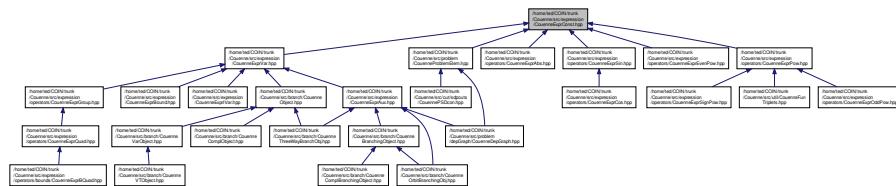
- namespace [Couenne](#)
general include file for different compilers

8.37 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprConst.hpp File Reference

```
#include <iostream>
#include "CouenneTypes.hpp"
#include "CouenneExpression.hpp"
Include dependency graph for CouenneExprConst.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

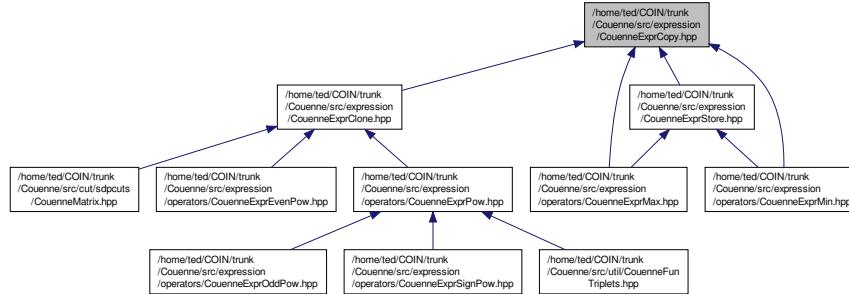
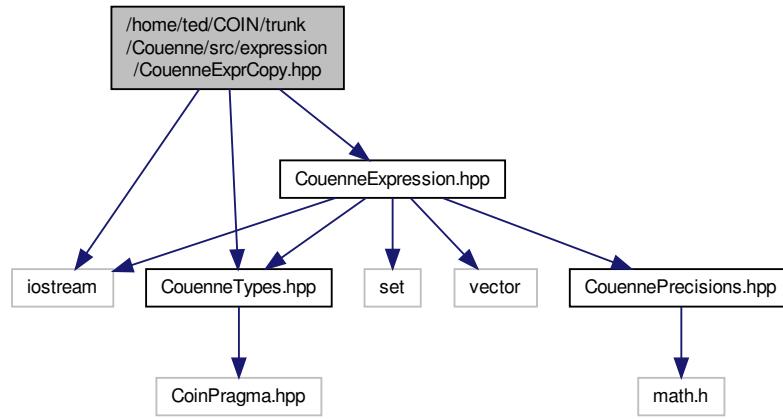
- class [Couenne::exprConst](#)
constant-type operator

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.38 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprCopy.hpp File Reference

```
#include <iostream>
#include "CouenneTypes.hpp"
#include "CouenneExpression.hpp"
Include dependency graph for CouenneExprCopy.hpp:
```



Classes

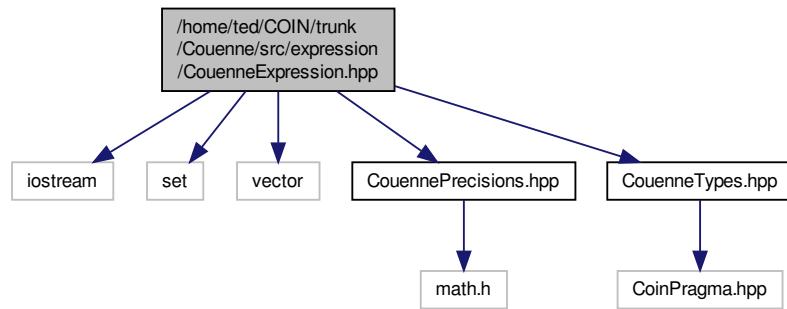
- class [Couenne::exprCopy](#)

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.39 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExpression.hpp File Reference

```
#include <iostream>
#include <set>
#include <vector>
#include "CouennePrecisions.hpp"
#include "CouenneTypes.hpp"
Include dependency graph for CouenneExpression.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::expression](#)

Expression base class.

Namespaces

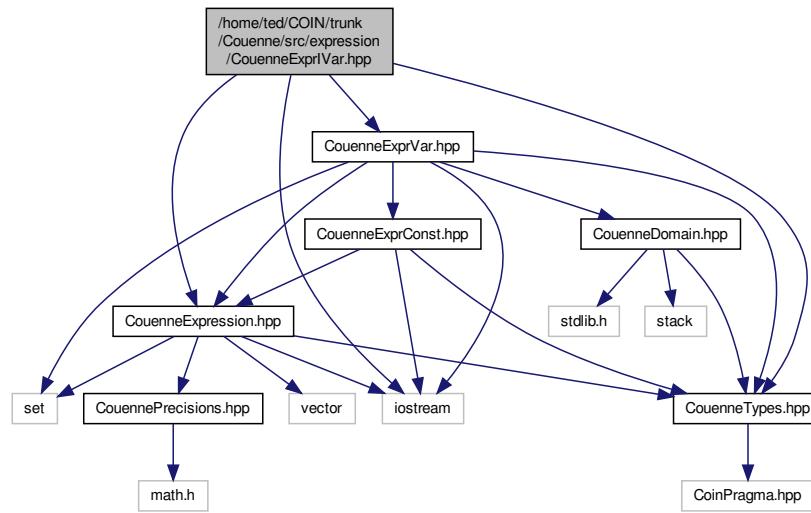
- namespace [Couenne](#)
- general include file for different compilers*

Functions

- bool [Couenne::updateBound](#) (register int sign, register CouNumber *dst, register CouNumber src)
updates maximum violation.
- int [Couenne::compareExpr](#) (const void *e0, const void *e1)
independent comparison
- bool [Couenne::isInteger](#) (CouNumber x)
is this number integer?
- expression * [Couenne::getOriginal](#) (expression *e)
get original expression (can't make it an expression method as I need a non-const, what "this" would return)

8.40 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprVar.hpp File Reference

```
#include <iostream>
#include "CouenneTypes.hpp"
#include "CouenneExpression.hpp"
#include "CouenneExprVar.hpp"
Include dependency graph for CouenneExprVar.hpp:
```



Classes

- class [Couenne::exprVar](#)
variable-type operator.

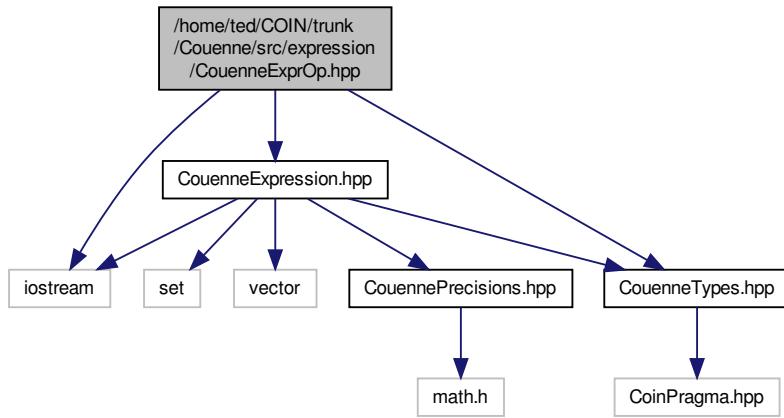
Namespaces

- namespace [Couenne](#)
general include file for different compilers

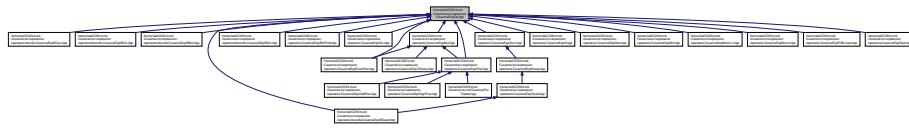
8.41 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprOp.hpp File Reference

```
#include <iostream>
#include "CouenneExpression.hpp"
#include "CouenneTypes.hpp"
```

Include dependency graph for CouenneExprOp.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprOp](#)
general n-ary operator-type expression: requires argument list.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Macros

- #define [MAX_ARG_LINE](#) 10

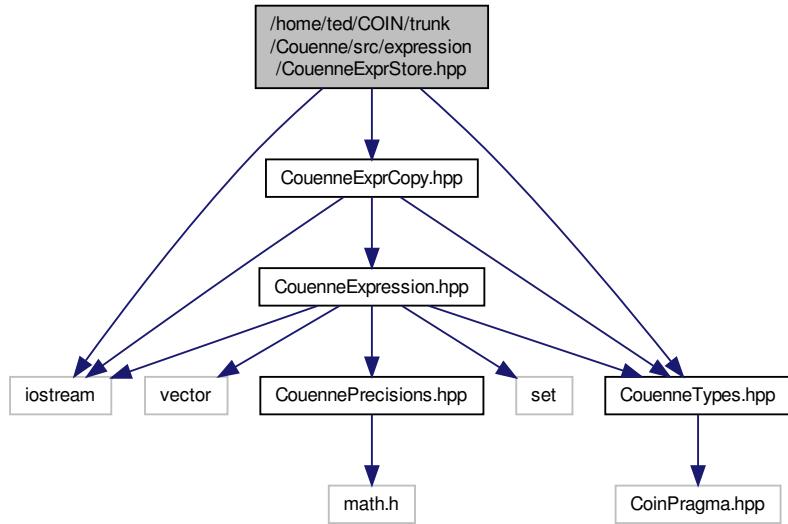
8.41.1 Macro Definition Documentation

8.41.1.1 #define MAX_ARG_LINE 10

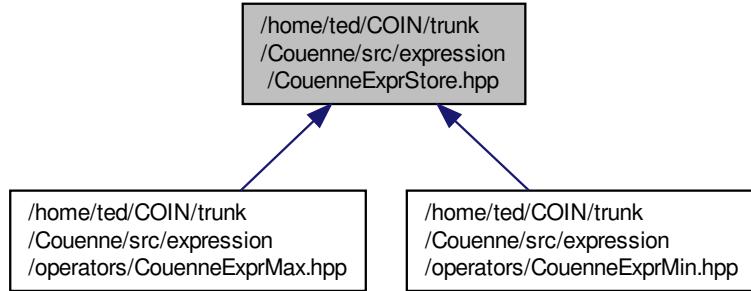
Definition at line 21 of file CouenneExprOp.hpp.

8.42 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprStore.hpp File Reference

```
#include <iostream>
#include "CouenneTypes.hpp"
#include "CouenneExprCopy.hpp"
Include dependency graph for CouenneExprStore.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprStore](#)
storage class for previously evaluated expressions

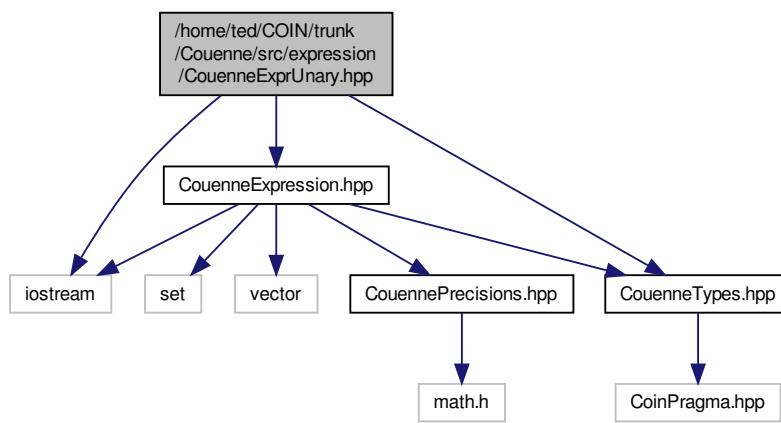
Namespaces

- namespace [Couenne](#)
general include file for different compilers

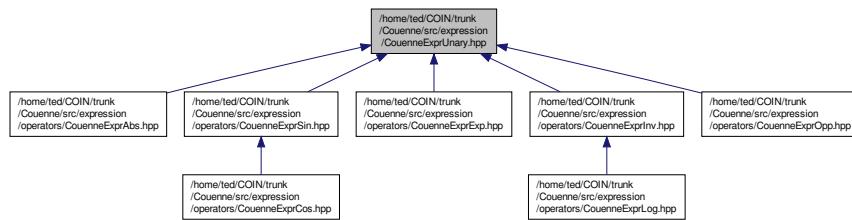
8.43 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprUnary.hpp File Reference

```
#include <iostream>
#include "CouenneExpression.hpp"
#include "CouenneTypes.hpp"
```

Include dependency graph for CouenneExprUnary.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprUnary](#)
expression class for unary functions (sin, log, etc.)

Namespaces

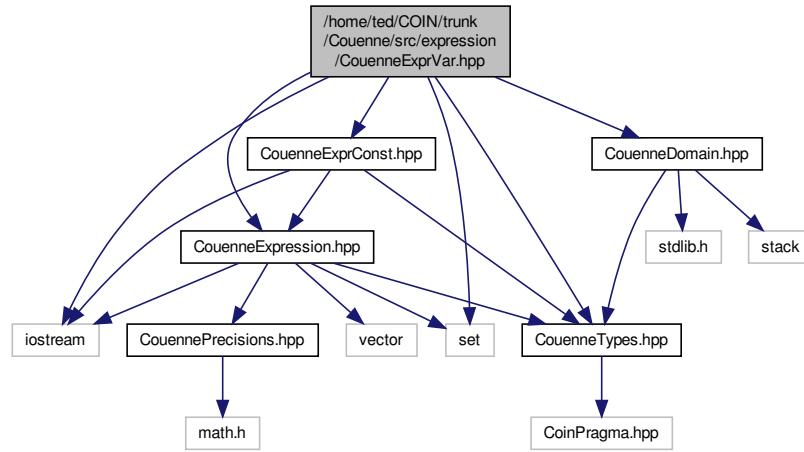
- namespace **Couenne**
general include file for different compilers

Functions

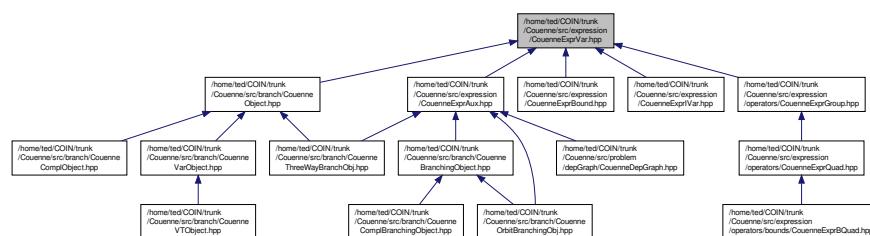
- CouNumber **Couenne::zero_fun** (CouNumber x)
zero function (used by default by exprUnary)

8.44 /home/ted/COIN/trunk/Couenne/src/expression/CouenneExprVar.hpp File Reference

```
#include <iostream>
#include <set>
#include "CouenneTypes.hpp"
#include "CouenneExpression.hpp"
#include "CouenneExprConst.hpp"
#include "CouenneDomain.hpp"
Include dependency graph for CouenneExprVar.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprVar](#)
variable-type operator

Namespaces

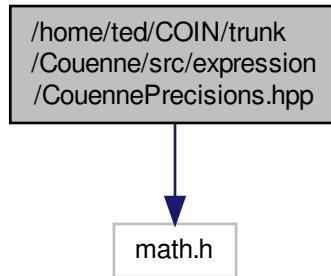
- namespace [Ipopt](#)
- namespace [Bonmin](#)
- namespace [Couenne](#)
general include file for different compilers

TypeDefs

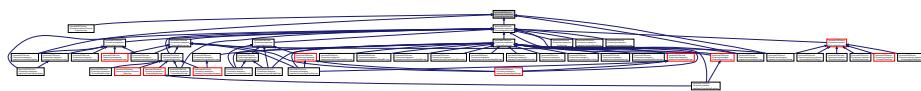
- typedef Ipopt::SmartPtr
 $< \text{Ipopt}::\text{Journalist} >$ [Couenne::JnlstPtr](#)
- typedef Ipopt::SmartPtr< const
 $\text{Ipopt}::\text{Journalist} >$ [Couenne::ConstJnlstPtr](#)

8.45 /home/ted/COIN/trunk/Couenne/src/expression/CouennePrecisions.hpp File Reference

```
#include <math.h>
Include dependency graph for CouennePrecisions.hpp:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **Couenne**
general include file for different compilers

Macros

- #define COUENNE_EPS 1.e-07
- #define COUENNE_BOUND_PREC 1.e-5
- #define COUENNE_EPS_INT 1.e-9
- #define COUENNE_EPS_SIMPL 1.e-20
- #define COUENNE_INFINITY 1.e+50
- #define COU_MAX_COEFF 1.e+9
- #define COU_MIN_COEFF 1.e-9
- #define COUENNE_round(x) ((int) (floor ((x) + 0.5)))
- #define COUENNE_sign(x) ((x) > 0.0 ? 1.0 : -1.0)
- #define MAX_BOUND 1.e45

Variables

- const double **Couenne::Couenne_large_bound** = 9.999e12
used to declare LP unbounded

8.45.1 Macro Definition Documentation

8.45.1.1 #define COUENNE_EPS 1.e-07

Definition at line 19 of file CouennePrecisions.hpp.

8.45.1.2 #define COUENNE_BOUND_PREC 1.e-5

Definition at line 22 of file CouennePrecisions.hpp.

8.45.1.3 #define COUENNE_EPS_INT 1.e-9

Definition at line 25 of file CouennePrecisions.hpp.

8.45.1.4 #define COUENNE_EPS_SIMPL 1.e-20

Definition at line 28 of file CouennePrecisions.hpp.

8.45.1.5 #define COUENNE_INFINITY 1.e+50

Definition at line 32 of file CouennePrecisions.hpp.

8.45.1.6 #define COU_MAX_COEFF 1.e+9

Definition at line 36 of file CouennePrecisions.hpp.

8.45.1.7 #define COU_MIN_COEFF 1.e-9

Definition at line 39 of file CouennePrecisions.hpp.

8.45.1.8 `#define COUENNE_round(x) ((int)(floor ((x) + 0.5)))`

Definition at line 42 of file CouennePrecisions.hpp.

8.45.1.9 `#define COUENNE_sign(x) ((x) > 0.0 ? 1.0 : -1.0)`

Definition at line 45 of file CouennePrecisions.hpp.

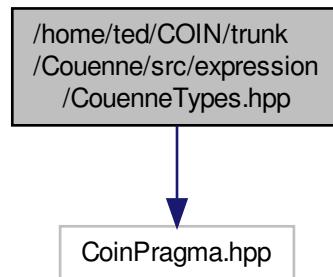
8.45.1.10 `#define MAX_BOUND 1.e45`

Definition at line 47 of file CouennePrecisions.hpp.

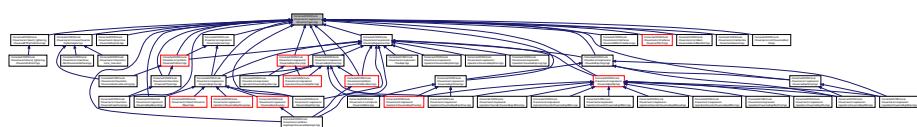
8.46 /home/ted/COIN/trunk/Couenne/src/expression/CouenneTypes.hpp File Reference

`#include "CoinPragma.hpp"`

Include dependency graph for CouenneTypes.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::t_chg_bounds](#)

status of lower/upper bound of a variable, to be checked/modified in bound tightening

Namespaces

- namespace [Couenne](#)

general include file for different compilers

Typedefs

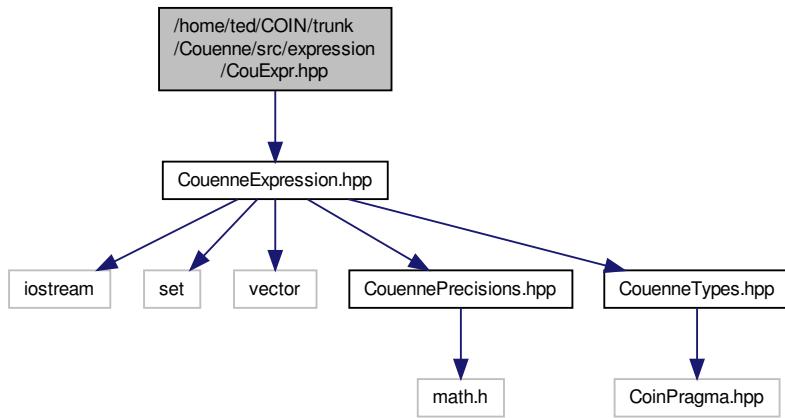
- `typedef double Couenne::CouNumber`
main number type in Couenne
- `typedef CouNumber(*) Couenne::unary_function)(CouNumber)`
unary function, used in all exprUnary

Enumerations

- `enum Couenne::nodeType { Couenne::CONST = 0, Couenne::VAR, Couenne::UNARY, Couenne::N_ARC, Couenne::COPY, Couenne::AUX, Couenne::EMPTY }`
type of a node in an expression tree
- `enum Couenne::linearity_type { Couenne::ZERO = 0, Couenne::CONSTANT, Couenne::LINEAR, Couenne::QUADRATIC, Couenne::NONLINEAR }`
linearity of an expression, as returned by the method Linearity()
- `enum Couenne::pos { Couenne::PRE = 0, Couenne::POST, Couenne::INSIDE, Couenne::NONE }`
position where the operator should be printed when printing the expression
- `enum Couenne::con_sign { Couenne::COUENNE_EQ, Couenne::COUENNE_LT, Couenne::COUENNE_GE, Couenne::COUENNE_RNG }`
sign of constraint
- `enum Couenne::conv_type { Couenne::CURRENT_ONLY, Couenne::UNIFORM_GRID, Couenne::AROUND_C-URPOINT }`
position and number of convexification cuts added for a lower convex (upper concave) envelope
- `enum Couenne::expr_type { Couenne::COU_EXPRESSION, Couenne::COU_EXPRCONST, Couenne::COU_EXPRVAR, Couenne::COU_E-XPRLBOUND, Couenne::COU_EXPRUBOUND, Couenne::COU_EXPROP, Couenne::COU_EXPRSUB, Couenne::COU_EXP-RSUM, Couenne::COU_EXPRGROUP, Couenne::COU_EXPRQUAD, Couenne::COU_EXPRMIN, Couenne::COU_EX-PRMUL, Couenne::COU_EXPRTRILINEAR, Couenne::COU_EXPRPOW, Couenne::COU_EXPRSIGNPOW, Couenne::COU_EXPRMAX, Couenne::COU_EXPRDIV, Couenne::COU_EXPRUNARY, Couenne::COU_EXPRCOS, Couenne::COU_EXPR-ABS, Couenne::COU_EXPREXP, Couenne::COU_EXPRINV, Couenne::COU_EXPRLOG, Couenne::COU_EXPROP-P, Couenne::COU_EXPRSIN, Couenne::COU_EXPRFLOOR, Couenne::COU_EXPRCEIL, Couenne::MAX_COU-EXPR_CODE }`
code returned by the method expression::code()
- `enum Couenne::convexity { Couenne::UNSET, Couenne::NONCONVEX, Couenne::CONVEX, Couenne::CONCAVE, Couenne::AFFINE, Couenne::CONV_LINEAR, Couenne::CONV_CONSTANT, Couenne::CONV_ZERO }`
convexity type of an expression
- `enum Couenne::monotonicity { Couenne::MON_UNSET, Couenne::NONMONOTONE, Couenne::NDECREAS, Couenne::NINCREAS, Couenne::INCLIN, Couenne::DECLIN, Couenne::MON_CONST, Couenne::MON_ZERO }`
monotonicity type of an expression
- `enum Couenne::dig_type { Couenne::ORIG_ONLY, Couenne::STOP_AT_AUX, Couenne::TAG_AND_RECURS-IVE, Couenne::COUNT }`
type of digging when filling the dependence list

8.47 /home/ted/COIN/trunk/Couenne/src/expression/CouExpr.hpp File Reference

```
#include "CouenneExpression.hpp"
Include dependency graph for CouExpr.hpp:
```



Classes

- class [Couenne::CouExpr](#)

Namespaces

- namespace [Couenne](#)

general include file for different compilers

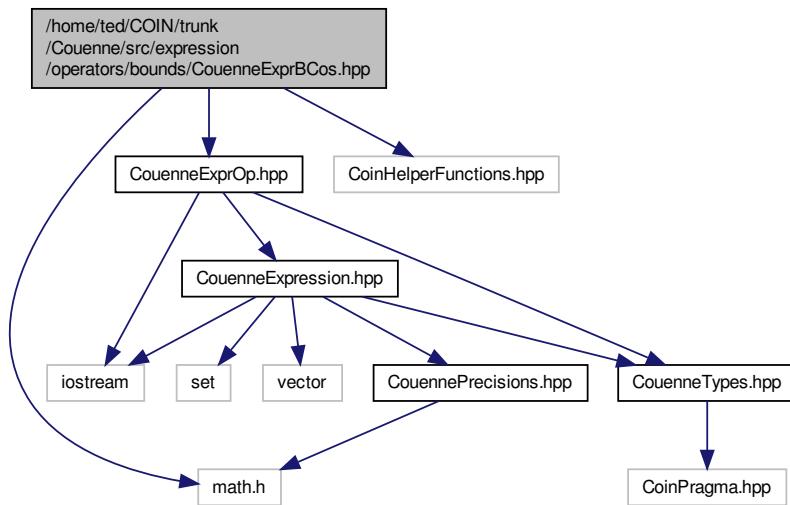
Functions

- CouExpr [Couenne::operator+](#) (CouExpr &e1, CouExpr &e2)
- CouExpr & [Couenne::operator/](#) (CouExpr &e1, CouExpr &e2)
- CouExpr & [Couenne::operator%](#) (CouExpr &e1, CouExpr &e2)
- CouExpr & [Couenne::operator-](#) (CouExpr &e1, CouExpr &e2)
- CouExpr & [Couenne::operator*](#) (CouExpr &e1, CouExpr &e2)
- CouExpr & [Couenne::operator^](#) (CouExpr &e1, CouExpr &e2)
- CouExpr & [Couenne::sin](#) (CouExpr &e)
- CouExpr & [Couenne::cos](#) (CouExpr &e)
- CouExpr & [Couenne::log](#) (CouExpr &e)
- CouExpr & [Couenne::exp](#) (CouExpr &e)
- CouExpr & [Couenne::operator+](#) (CouNumber &e1, CouExpr &e2)
- CouExpr & [Couenne::operator/](#) (CouNumber &e1, CouExpr &e2)
- CouExpr & [Couenne::operator%](#) (CouNumber &e1, CouExpr &e2)
- CouExpr & [Couenne::operator-](#) (CouNumber &e1, CouExpr &e2)
- CouExpr & [Couenne::operator*](#) (CouNumber &e1, CouExpr &e2)

- CouExpr & [Couenne::operator^](#) (CouNumber &e1, CouExpr &e2)
- CouExpr & [Couenne::sin](#) (CouNumber &e)
- CouExpr & [Couenne::cos](#) (CouNumber &e)
- CouExpr & [Couenne::log](#) (CouNumber &e)
- CouExpr & [Couenne::exp](#) (CouNumber &e)
- CouExpr & [Couenne::operator+](#) (CouExpr &e1, CouNumber &e2)
- CouExpr & [Couenne::operator/](#) (CouExpr &e1, CouNumber &e2)
- CouExpr & [Couenne::operator%](#) (CouExpr &e1, CouNumber &e2)
- CouExpr & [Couenne::operator-](#) (CouExpr &e1, CouNumber &e2)
- CouExpr & [Couenne::operator*](#) (CouExpr &e1, CouNumber &e2)
- CouExpr & [Couenne::operator^](#) (CouExpr &e1, CouNumber &e2)

8.48 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBCos.hpp File Reference

```
#include "CouenneExprOp.hpp"
#include "CoinHelperFunctions.hpp"
#include <math.h>
Include dependency graph for CouenneExprBCos.hpp:
```



Classes

- class [Couenne::exprLBCos](#)
class to compute lower bound of a cosine based on the bounds of its arguments
- class [Couenne::exprUBCos](#)
class to compute lower bound of a cosine based on the bounds of its arguments

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Macros

- #define [M_PI](#) 3.14159265358979323846

8.48.1 Macro Definition Documentation

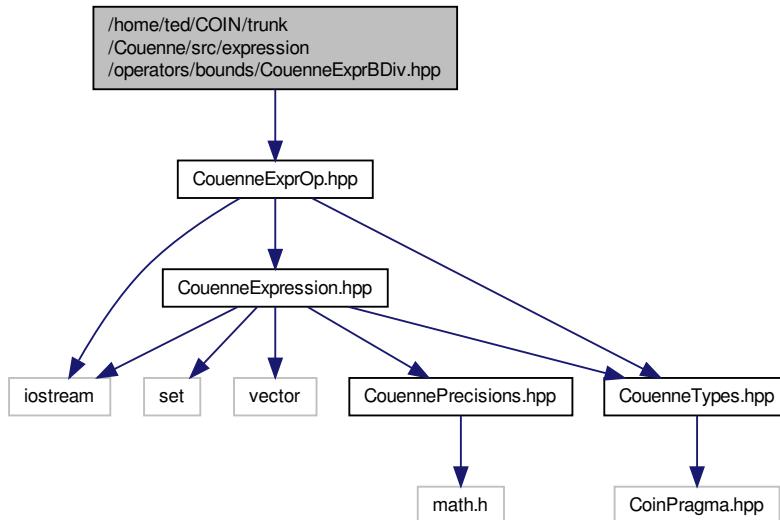
8.48.1.1 #define M_PI 3.14159265358979323846

Definition at line 19 of file CouenneExprBCos.hpp.

8.49 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBDiv.hpp File Reference

```
#include "CouenneExprOp.hpp"
```

Include dependency graph for CouenneExprBDiv.hpp:



Classes

- class [Couenne::exprLBDiv](#)
class to compute lower bound of a fraction based on the bounds of both numerator and denominator
- class [Couenne::exprUBDiv](#)
class to compute upper bound of a fraction based on the bounds of both numerator and denominator

Namespaces

- namespace [Couenne](#)
general include file for different compilers

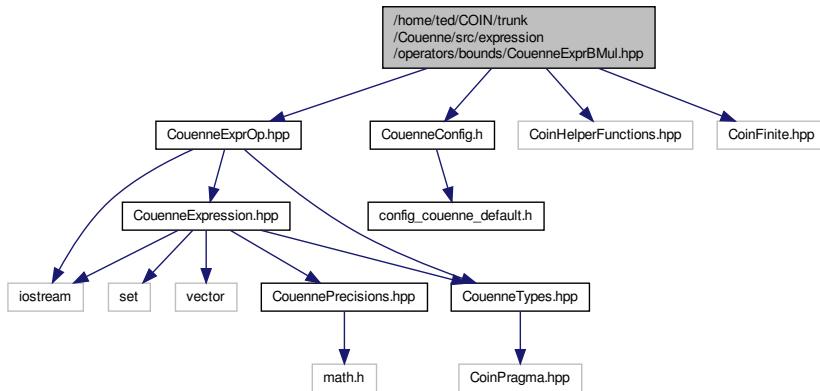
Functions

- static CouNumber [Couenne::safeDiv](#) (register CouNumber a, register CouNumber b, int sign)
division that avoids NaN's and considers a sign when returning infinity

8.50 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBMul.hpp File Reference

```
#include "CouenneExprOp.hpp"
#include "CouenneConfig.h"
#include "CoinHelperFunctions.hpp"
#include "CoinFinite.hpp"

Include dependency graph for CouenneExprBMul.hpp:
```



Classes

- class [Couenne::exprLBMul](#)
class to compute lower bound of a product based on the bounds of both factors
- class [Couenne::exprUBMul](#)
class to compute upper bound of a product based on the bounds of both factors

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Macros

- #define MUL_ZERO 1e-20
- #define MUL_INF sqrt (COIN_DBLE_MAX)

Functions

- CouNumber [Couenne::safeProd](#) (register CouNumber a, register CouNumber b)
product that avoids NaN's

8.50.1 Macro Definition Documentation

8.50.1.1 #define MUL_ZERO 1e-20

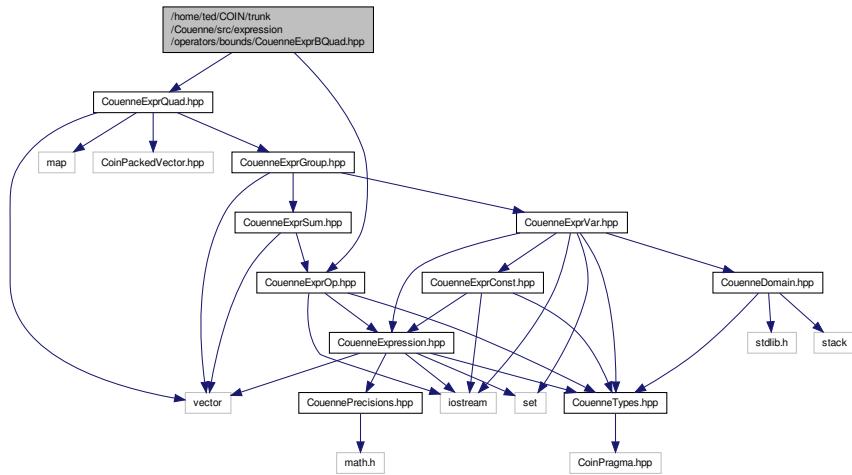
Definition at line 21 of file CouenneExprBMul.hpp.

8.50.1.2 #define MUL_INF sqrt (COIN_DBLE_MAX)

Definition at line 22 of file CouenneExprBMul.hpp.

8.51 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBQuad.hpp File Reference

```
#include "CouenneExprOp.hpp"
#include "CouenneExprQuad.hpp"
Include dependency graph for CouenneExprBQuad.hpp:
```



Classes

- class [Couenne::exprLBQuad](#)
class to compute lower bound of a fraction based on the bounds of both numerator and denominator
- class [Couenne::exprUBQuad](#)
class to compute upper bound of a fraction based on the bounds of both numerator and denominator

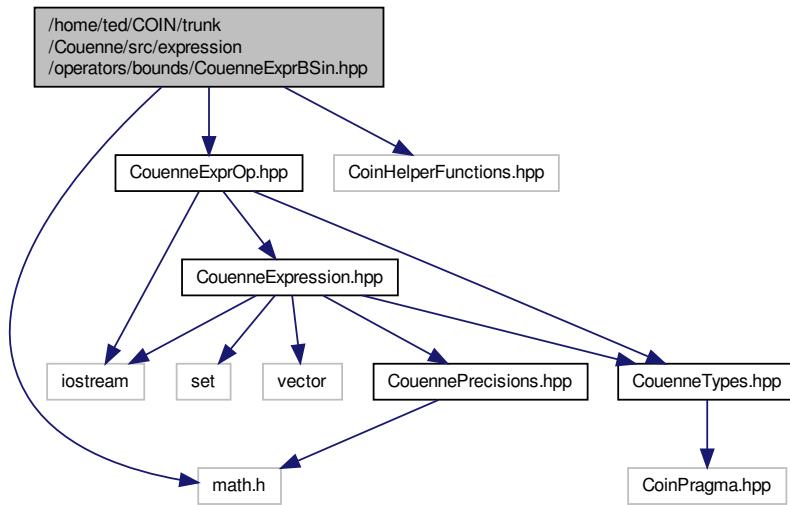
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.52 /home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/CouenneExprBSin.hpp File Reference

```
#include "CouenneExprOp.hpp"
#include "CoinHelperFunctions.hpp"
#include <math.h>
```

Include dependency graph for CouenneExprBSin.hpp:



Classes

- class [Couenne::exprLBSin](#)
class to compute lower bound of a sine based on the bounds on its arguments
- class [Couenne::exprUBSin](#)
class to compute lower bound of a sine based on the bounds on its arguments

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Macros

- #define [M_PI](#) 3.14159265358979323846

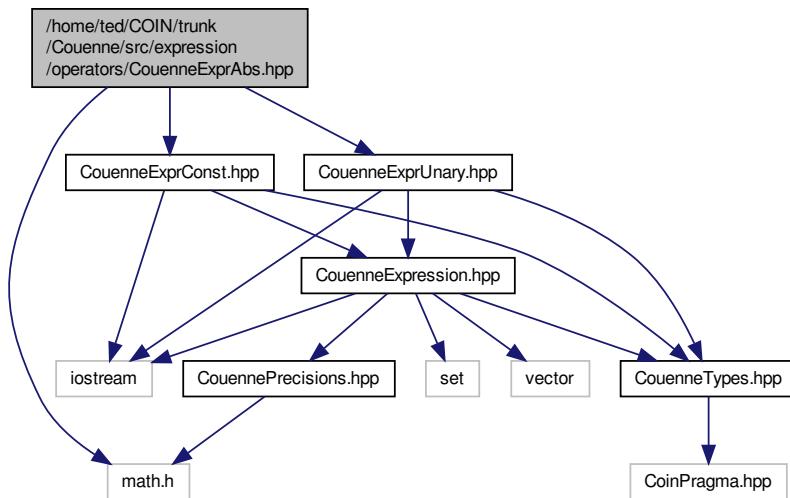
8.52.1 Macro Definition Documentation

8.52.1.1 #define M_PI 3.14159265358979323846

Definition at line 19 of file CouenneExprBSin.hpp.

8.53 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprAbs.hpp File Reference

```
#include <math.h>
#include "CouenneExprUnary.hpp"
#include "CouenneExprConst.hpp"
Include dependency graph for CouenneExprAbs.hpp:
```



Classes

- class `Couenne::exprAbs`

$$class \ for |f(x)|$$

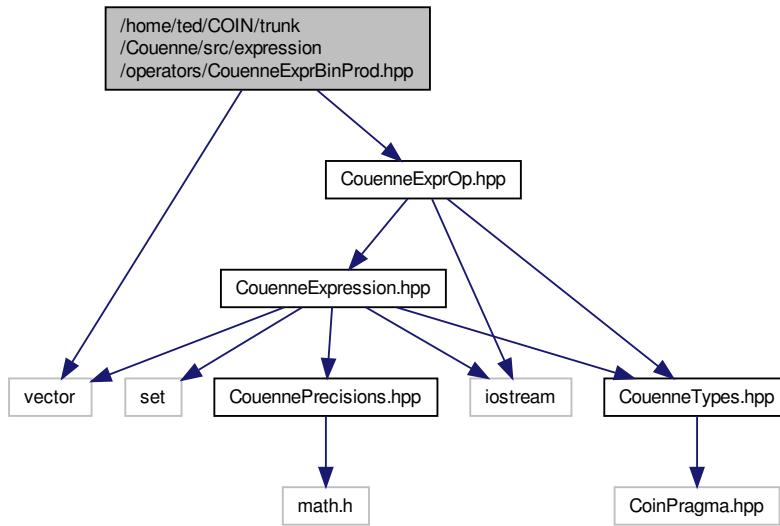
Namespaces

- namespace **Couenne**
general include file for different compilers

8.54 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprBinProd.hpp File Reference

```
#include <vector>
#include "CouenneExprOp.hpp"
```

Include dependency graph for CouenneExprBinProd.hpp:



Classes

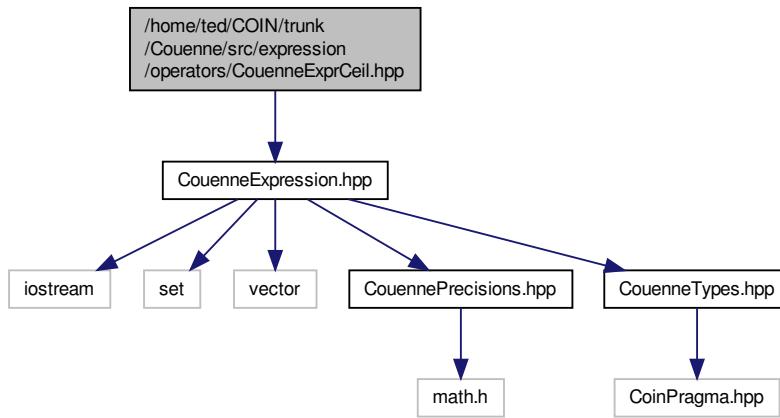
- class [Couenne::exprBinProd](#)
class for $\prod_{i=1}^n f_i(x)$ with $f_i(x)$ all binary

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.55 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprCeil.hpp File Reference

```
#include "CouenneExpression.hpp"
Include dependency graph for CouenneExprCeil.hpp:
```



Classes

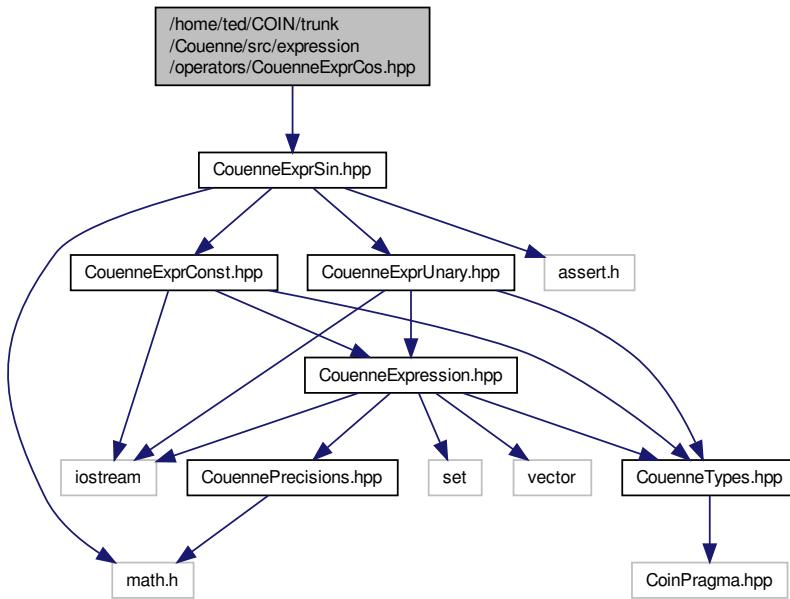
- class [Couenne::exprCeil](#)
class ceiling, $\lceil f(x) \rceil$

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.56 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprCos.hpp File Reference

```
#include "CouenneExprSin.hpp"
Include dependency graph for CouenneExprCos.hpp:
```



Classes

- class [Couenne::exprCos](#)
class cosine, cos f(x)

Namespaces

- namespace [Couenne](#)
general include file for different compilers

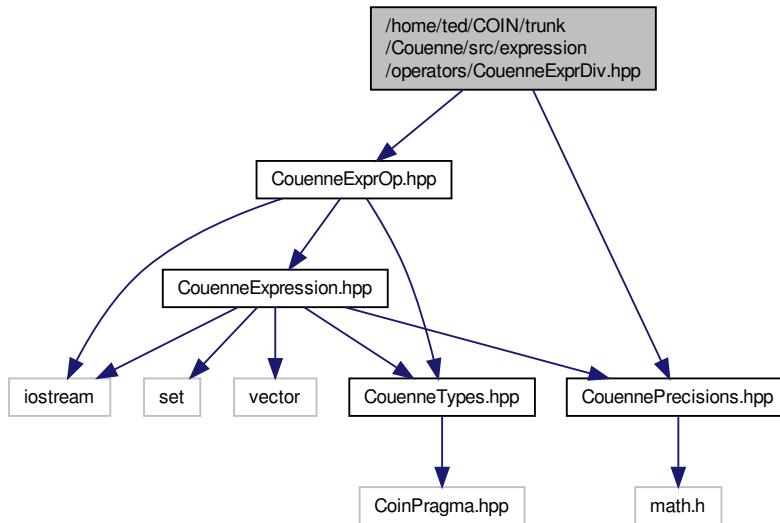
Functions

- CouNumber [Couenne::trigNewton](#) (CouNumber, CouNumber, CouNumber)
common convexification method used by both cos and sin

8.57 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprDiv.hpp File Reference

```
#include "CouenneExprOp.hpp"
#include "CouennePrecisions.hpp"
```

Include dependency graph for CouenneExprDiv.hpp:



Classes

- class [Couenne::exprDiv](#)
class for divisions, $\frac{f(x)}{g(x)}$

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Macros

- #define [BR_NEXT_ZERO](#) 1e-3
- #define [BR_MULT](#) 1e-3
- #define [SAFE_COEFFICIENT](#) 1e9

Functions

- bool [Couenne::is_boundbox_regular](#) (register CouNumber b1, register CouNumber b2)
check if bounding box is suitable for a multiplication/division convexification constraint

8.57.1 Macro Definition Documentation

8.57.1.1 #define BR_NEXT_ZERO 1e-3

Definition at line 19 of file CouenneExprDiv.hpp.

8.57.1.2 #define BR_MULT 1e-3

Definition at line 20 of file CouenneExprDiv.hpp.

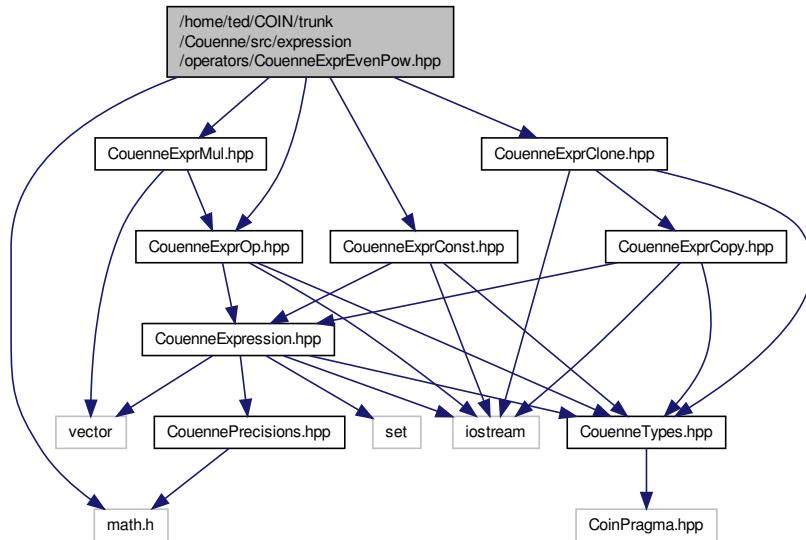
8.57.1.3 #define SAFE_COEFFICIENT 1e9

Definition at line 119 of file CouenneExprDiv.hpp.

8.58 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprEvenPow.hpp File Reference

```
#include <math.h>
#include "CouenneExprOp.hpp"
#include "CouenneExprMul.hpp"
#include "CouenneExprClone.hpp"
#include "CouenneExprConst.hpp"
```

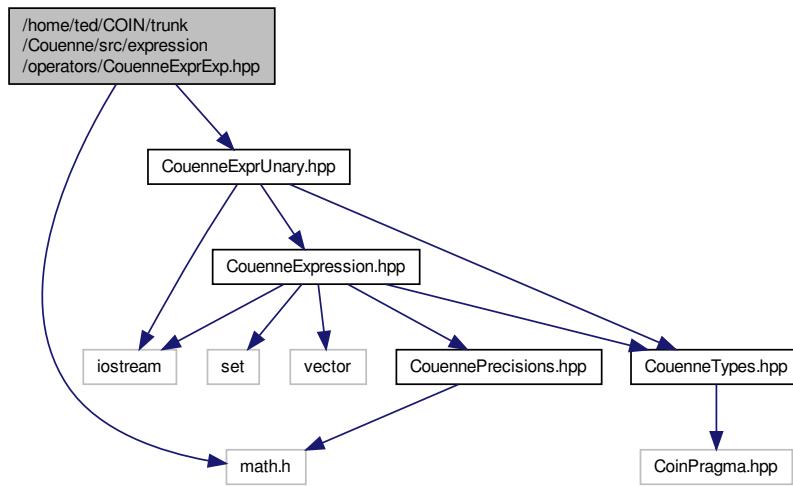
Include dependency graph for CouenneExprEvenPow.hpp:



8.59 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprExp.hpp File Reference

```
#include <math.h>
#include "CouenneExprUnary.hpp"
```

Include dependency graph for CouenneExprExp.hpp:



Classes

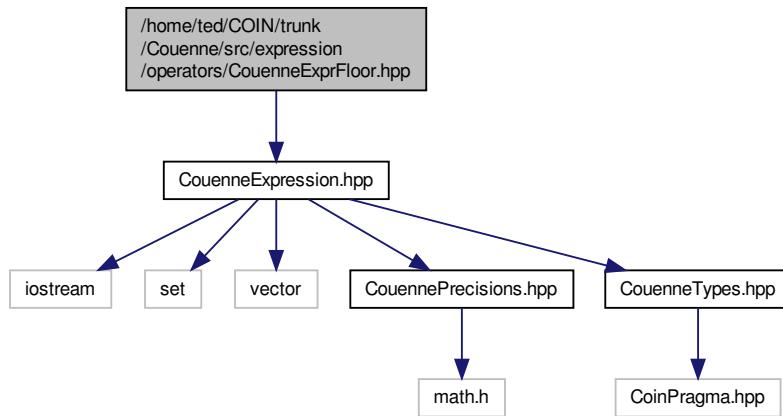
- class [Couenne::exprExp](#)
class for the exponential, $e^{f(x)}$

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.60 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprFloor.hpp File Reference

```
#include "CouenneExpression.hpp"
Include dependency graph for CouenneExprFloor.hpp:
```



Classes

- class [Couenne::exprFloor](#)
 $class floor, \lfloor f(x) \rfloor$

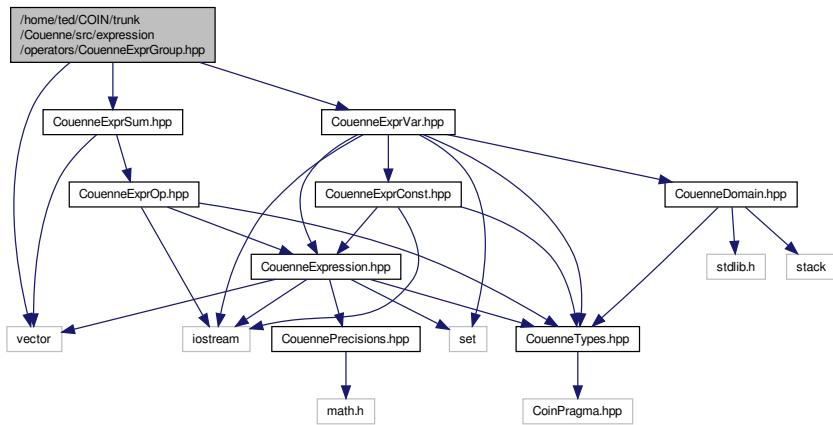
Namespaces

- namespace [Couenne](#)
general include file for different compilers

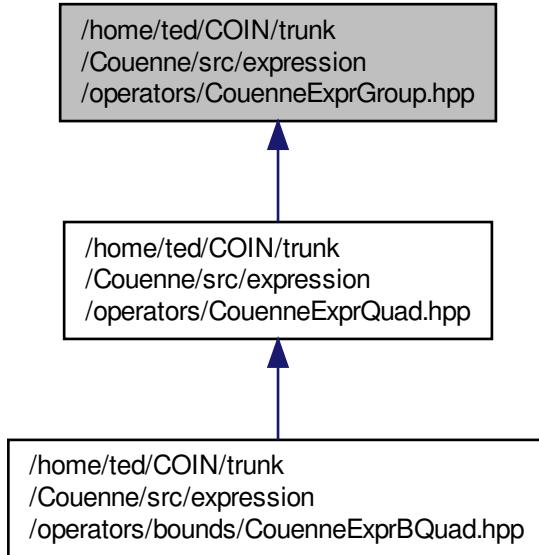
8.61 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprGroup.hpp File Reference

```
#include <vector>
#include "CouenneExprSum.hpp"
#include "CouenneExprVar.hpp"
```

Include dependency graph for CouenneExprGroup.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprGroup](#)
class Group, with constant, linear and nonlinear terms: $a_0 + \sum_{i=1}^n a_i x_i$

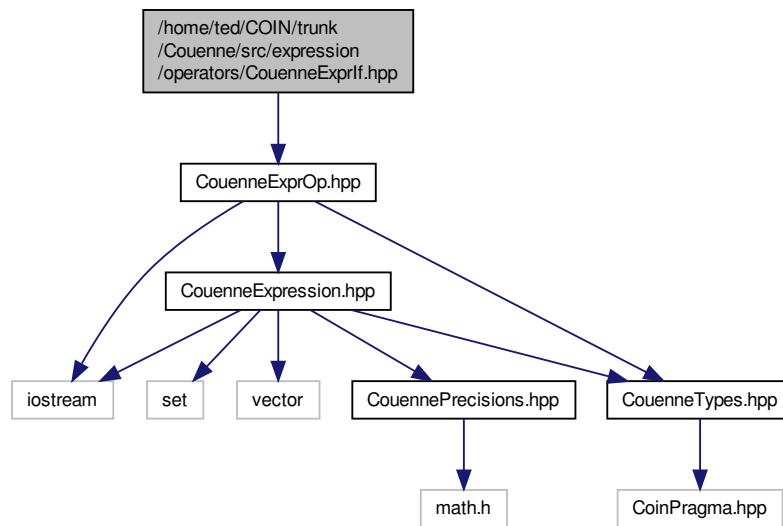
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.62 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprIf.hpp File Reference

```
#include "CouenneExprOp.hpp"
```

Include dependency graph for CouenneExprIf.hpp:



Classes

- class [Couenne::exprIf](#)

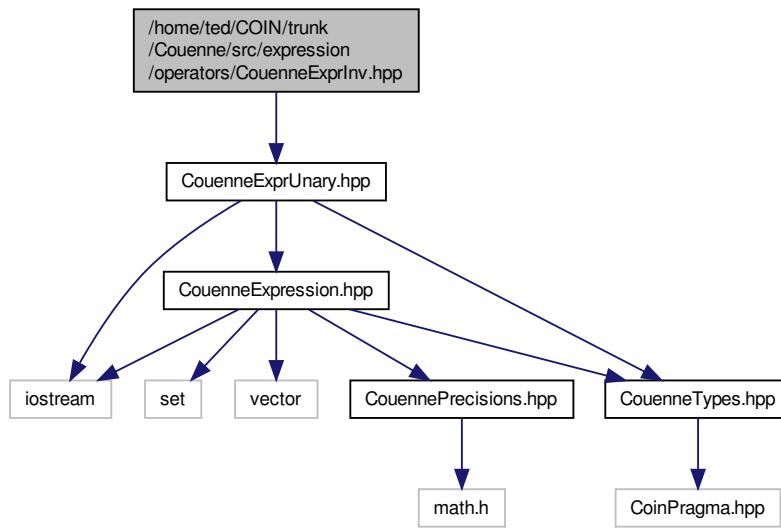
Namespaces

- namespace [Couenne](#)
general include file for different compilers

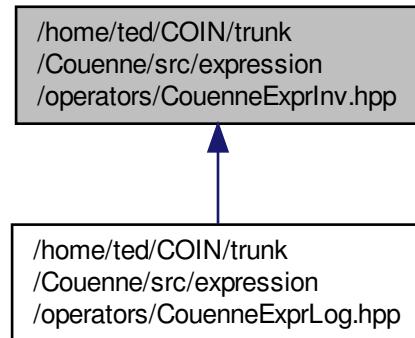
8.63 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprInv.hpp File Reference

```
#include "CouenneExprUnary.hpp"
```

Include dependency graph for CouenneExprInv.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprInv](#)
class inverse: $1/f(x)$

Namespaces

- namespace **Couenne**
general include file for different compilers

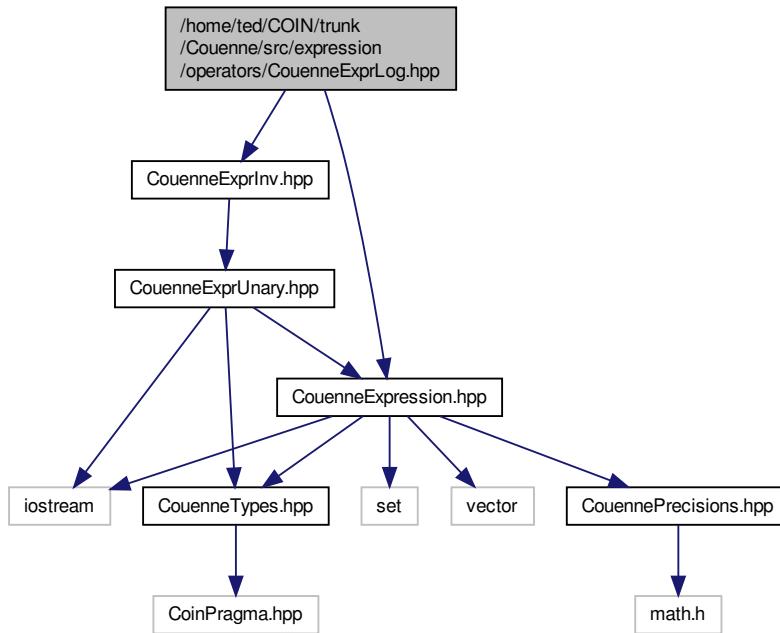
Functions

- CouNumber **Couenne::inv** (register CouNumber arg)
the operator itself
- CouNumber **Couenne::opplnInvSqr** (register CouNumber x)
derivative of inv (x)
- CouNumber **Couenne::inv_dblprime** (register CouNumber x)
inv_dblprime, second derivative of inv (x)

8.64 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprLog.hpp File Reference

```
#include "CouenneExprInv.hpp"
#include "CouenneExpression.hpp"
```

Include dependency graph for CouenneExprLog.hpp:



Classes

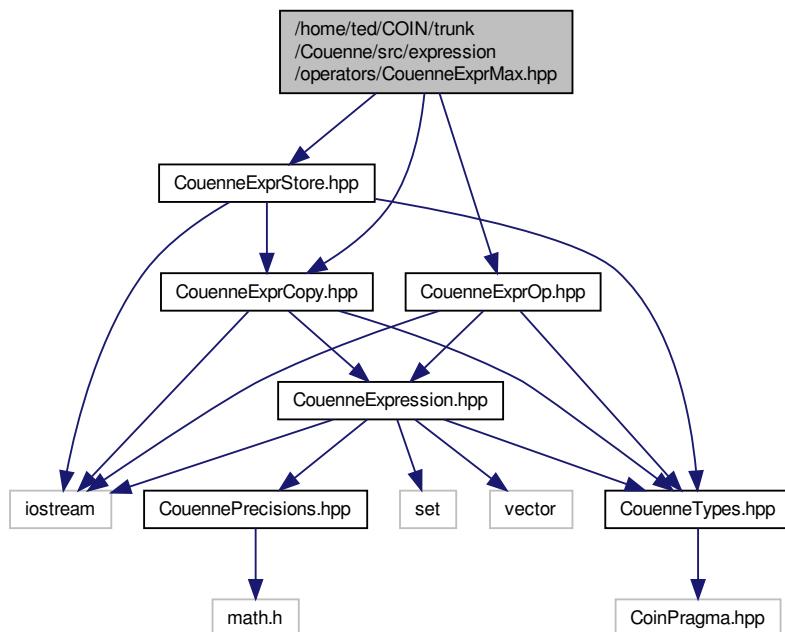
- class **Couenne::exprLog**
class logarithm, log f(x)

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.65 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprMax.hpp File Reference

```
#include "CouenneExprOp.hpp"
#include "CouenneExprCopy.hpp"
#include "CouenneExprStore.hpp"
Include dependency graph for CouenneExprMax.hpp:
```



Classes

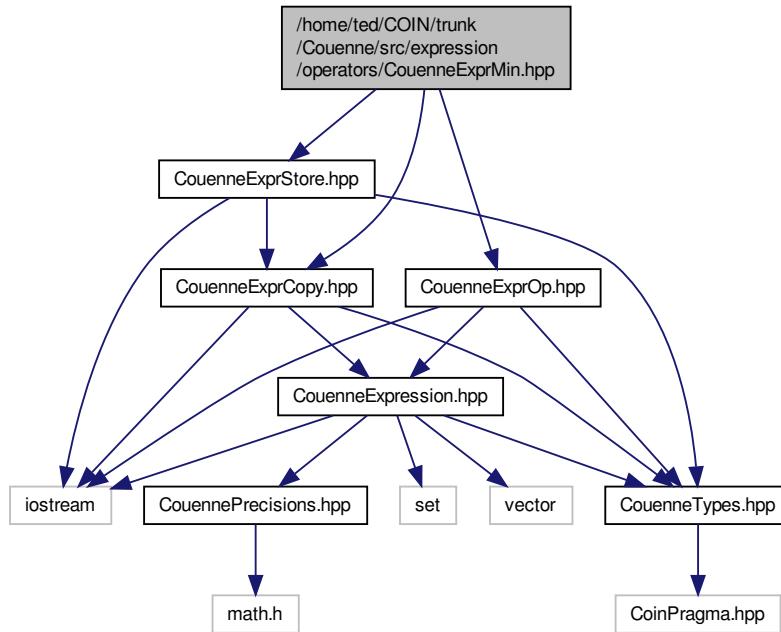
- class [Couenne::exprMax](#)
class for maxima

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.66 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprMin.hpp File Reference

```
#include "CouenneExprOp.hpp"
#include "CouenneExprCopy.hpp"
#include "CouenneExprStore.hpp"
Include dependency graph for CouenneExprMin.hpp:
```



Classes

- class [Couenne::exprMin](#)

class for minima

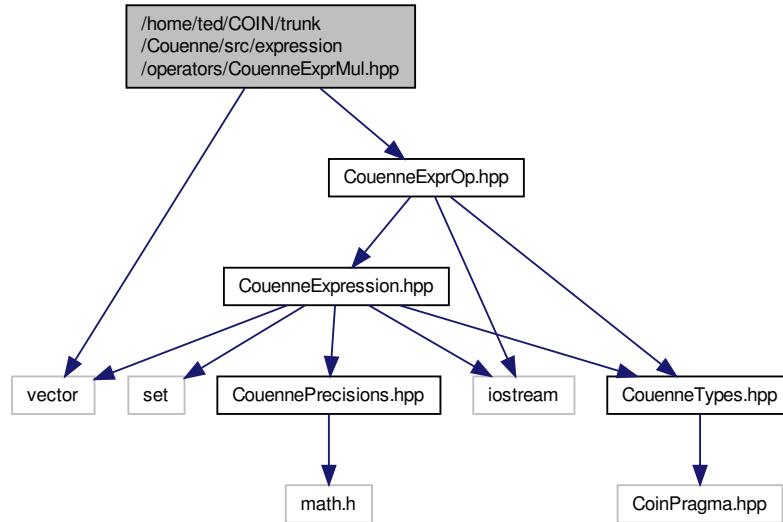
Namespaces

- namespace [Couenne](#)
- general include file for different compilers*

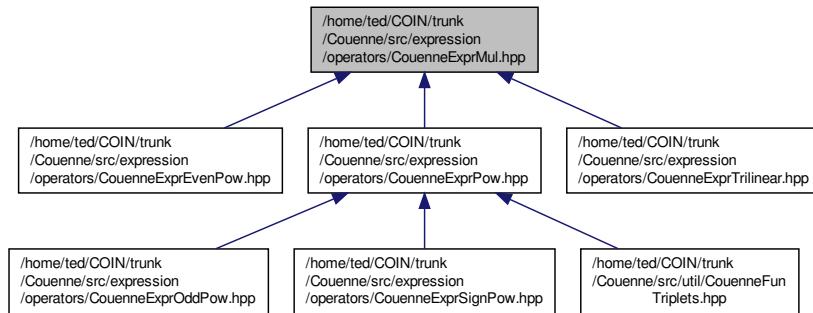
8.67 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprMul.hpp File Reference

```
#include <vector>
#include "CouenneExprOp.hpp"
```

Include dependency graph for CouenneExprMul.hpp:



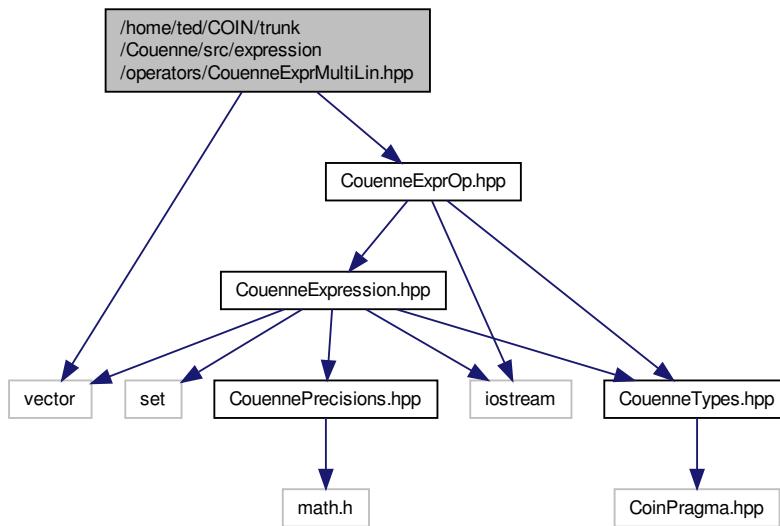
This graph shows which files directly or indirectly include this file:



8.68 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprMultiLin.hpp File Reference

```
#include <vector>
#include "CouenneExprOp.hpp"
```

Include dependency graph for CouenneExprMultiLin.hpp:



Classes

- class [Couenne::exprMultiLin](#)
another class for multiplications, $\prod_{i=1}^n f_i(x)$

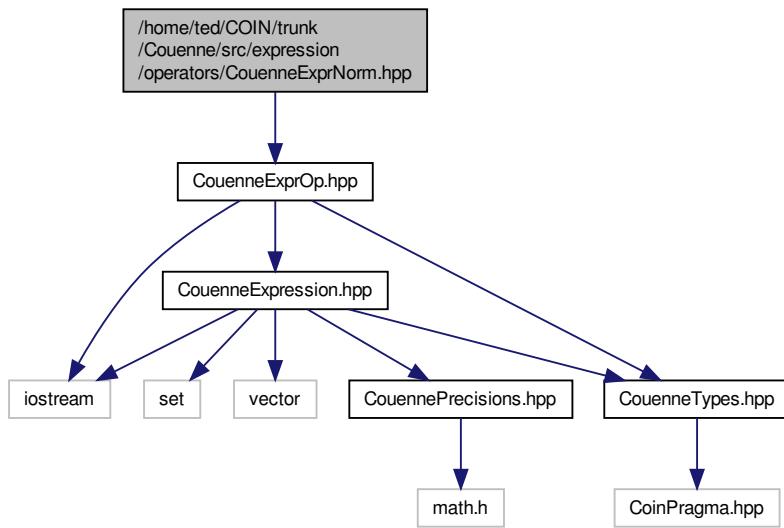
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.69 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprNorm.hpp File Reference

```
#include "CouenneExprOp.hpp"
```

Include dependency graph for CouenneExprNorm.hpp:



Classes

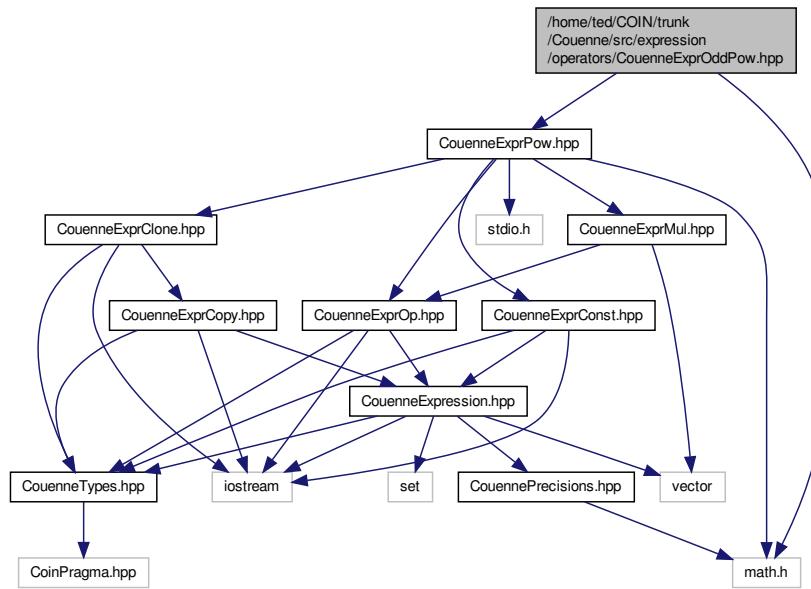
- class [Couenne::exprNorm](#)
Class for p-norms, $\|f(x)\|_p = \left(\sum_{i=1}^n f_i(x)^p\right)^{\frac{1}{p}}$.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

```
#include <math.h>
#include "CouenneExprNorm.hpp"
```

Include dependency graph for CouenneExprOddPow.hpp:



Classes

- class [Couenne::exprOddPow](#)
Power of an expression (binary operator), $f(x)^k$ with k constant.

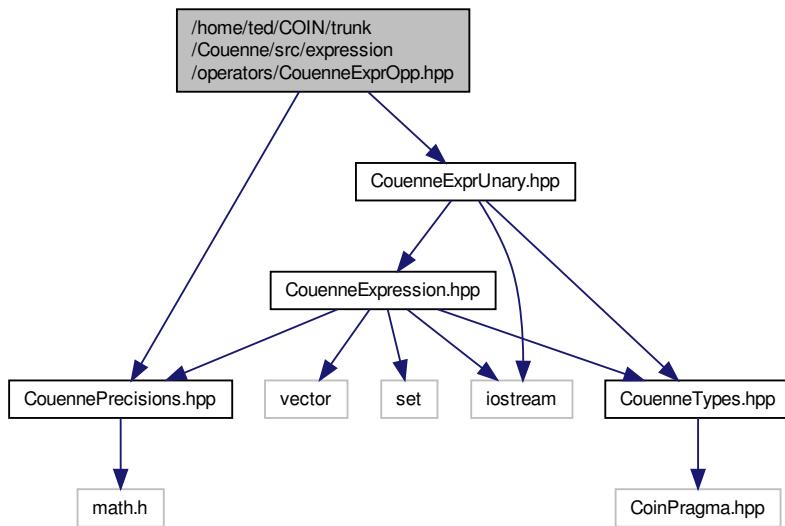
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.71 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprOpp.hpp File Reference

```
#include "CouennePrecisions.hpp"
#include "CouenneExprUnary.hpp"
```

Include dependency graph for CouenneExprOpp.hpp:



Classes

- class [Couenne::exprOpp](#)
class opposite, $-f(x)$

Namespaces

- namespace [Couenne](#)
general include file for different compilers

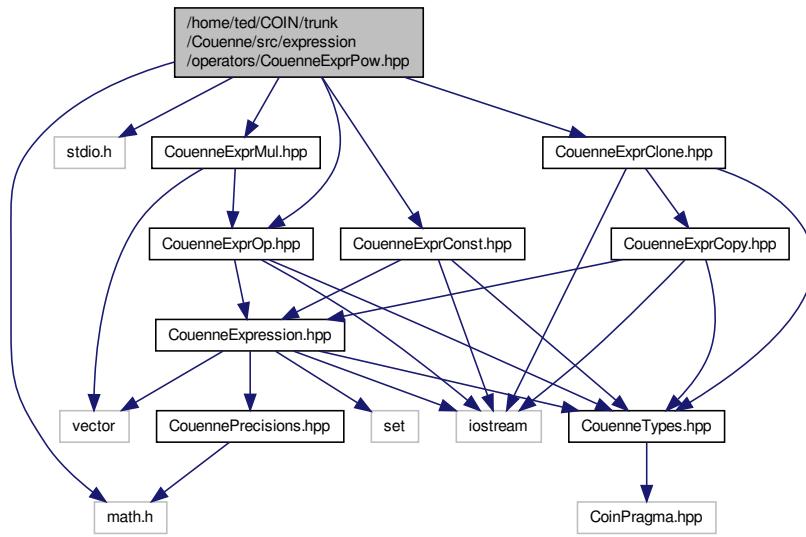
Functions

- CouNumber [Couenne::opp](#) (register CouNumber arg)
operator opp: returns the opposite of a number

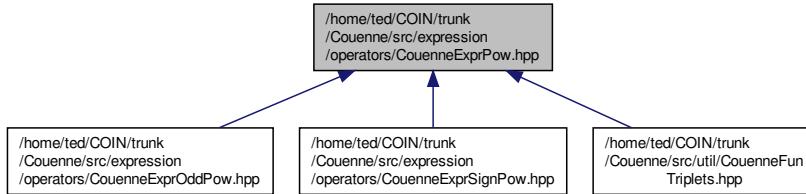
8.72 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprPow.hpp File Reference

```
#include <math.h>
#include <stdio.h>
#include "CouenneExprOp.hpp"
#include "CouenneExprMul.hpp"
#include "CouenneExprClone.hpp"
#include "CouenneExprConst.hpp"
```

Include dependency graph for CouenneExprPow.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprPow](#)
Power of an expression (binary operator), $f(x)^k$ with k constant.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

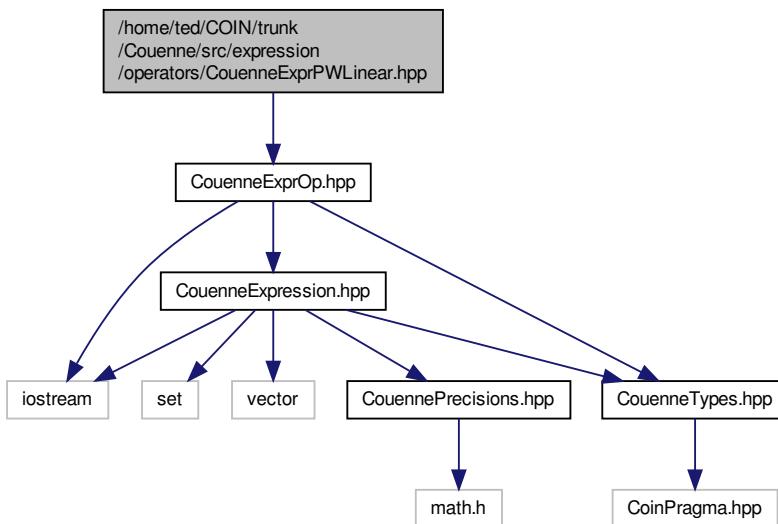
Functions

- CouNumber [Couenne::safe_pow](#) (CouNumber base, CouNumber exponent, bool signpower=false)

- compute power and check for integer-and-odd inverse exponent
- void [Couenne::addPowEnvelope](#) (const CouenneCutGenerator *, OsiCuts &, int, int, CouNumber, CouNumber, CouNumber, CouNumber, CouNumber, int, bool=false)
 - add upper/lower envelope to power in convex/concave areas*
- CouNumber [Couenne::powNewton](#) (CouNumber, CouNumber, unary_function, unary_function, unary_function)
 - find proper tangent point to add deepest tangent cut*
- CouNumber [Couenne::powNewton](#) (CouNumber, CouNumber, funtriplet *)
 - find proper tangent point to add deepest tangent cut*

8.73 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprPWLinear.hpp File Reference

```
#include "CouenneExprOp.hpp"
Include dependency graph for CouenneExprPWLinear.hpp:
```



Classes

- class [Couenne::exprPWLinear](#)

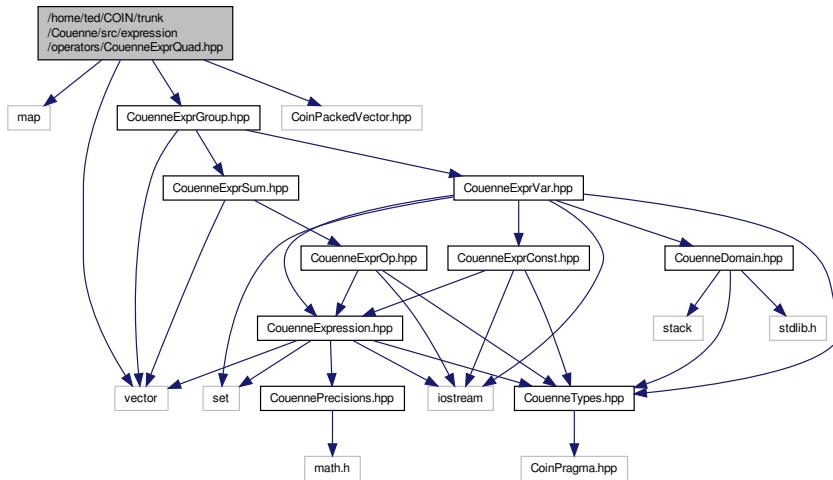
Namespaces

- namespace [Couenne](#)
 - general include file for different compilers*

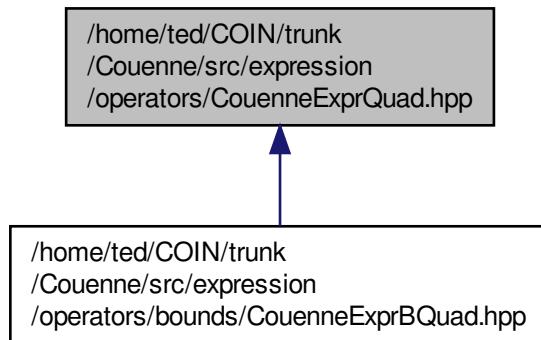
8.74 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprQuad.hpp File Reference

```
#include <map>
```

```
#include <vector>
#include "CoinPackedVector.hpp"
#include "CouenneExprGroup.hpp"
Include dependency graph for CouenneExprQuad.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

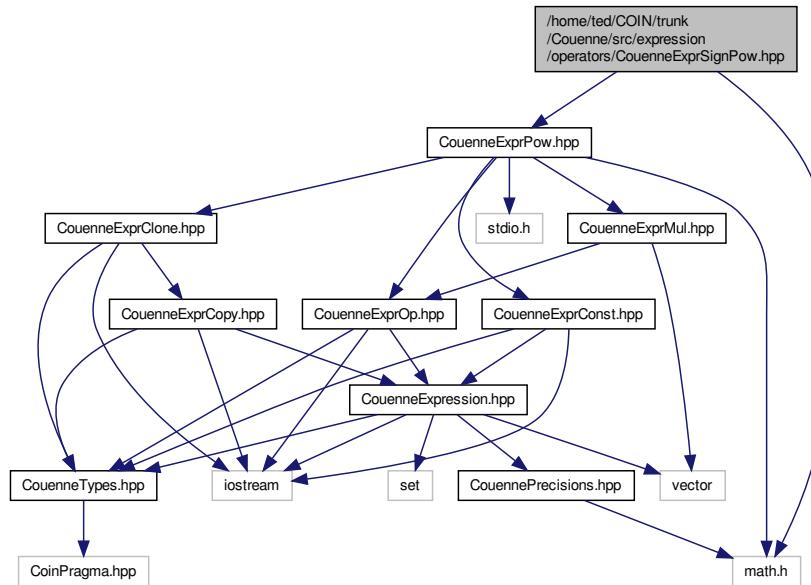
- class [Couenne::exprQuad](#)
class `exprQuad`, with constant, linear and quadratic terms

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.75 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprSignPow.hpp File Reference

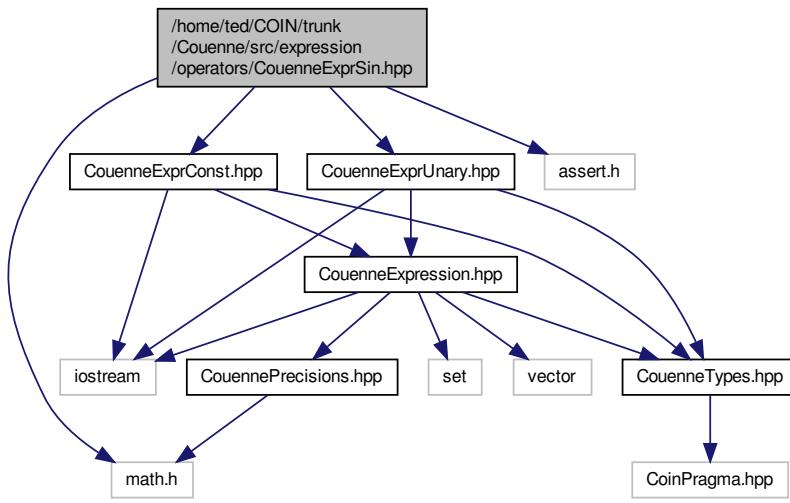
```
#include <math.h>
#include "CouenneExprPow.hpp"
Include dependency graph for CouenneExprSignPow.hpp:
```



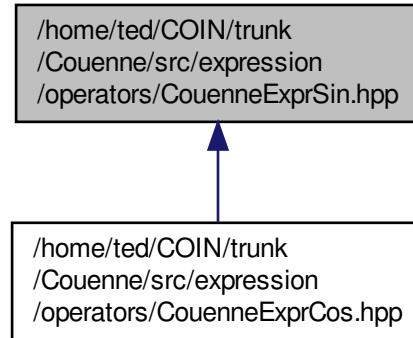
8.76 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprSin.hpp File Reference

```
#include <math.h>
#include <assert.h>
#include "CouenneExprUnary.hpp"
#include "CouenneExprConst.hpp"
```

Include dependency graph for CouenneExprSin.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprSin](#)
class for sin f(x)

Namespaces

- namespace [Couenne](#)

general include file for different compilers

Enumerations

- enum **Couenne::cou_trig** { **Couenne::COU_SINE**, **Couenne::COU_COSINE** }
specify which trigonometric function is dealt with in trigEnvelope

Functions

- CouNumber **Couenne::modulo** (register CouNumber a, register CouNumber b)
normalize angle within [0,b] (typically, pi or 2pi)
- CouNumber **Couenne::trigSelBranch** (const CouenneObject *obj, const OsiBranchingInformation *info, expression *&var, double *&brpts, double *&brDist, int &way, enum cou_trig type)
generalized procedure for both sine and cosine
- bool **Couenne::trigImpliedBound** (enum cou_trig, int, int, CouNumber *, CouNumber *, t_chg_bounds *)
generalized implied bound procedure for sine/cosine

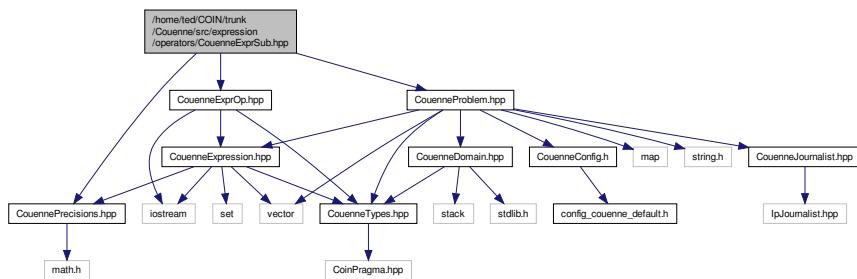
8.77 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprSub.hpp File Reference

```
#include "CouenneExprOp.hpp"
```

```
#include "CouennePrecisions.hpp"
```

```
#include "CouenneProblem.hpp"
```

Include dependency graph for CouenneExprSub.hpp:



Classes

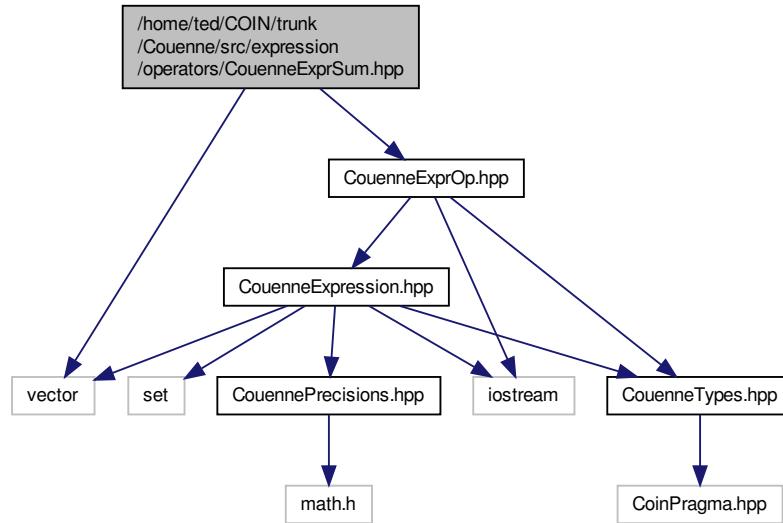
- class **Couenne::exprSub**
class for subtraction, $f(x) - g(x)$

Namespaces

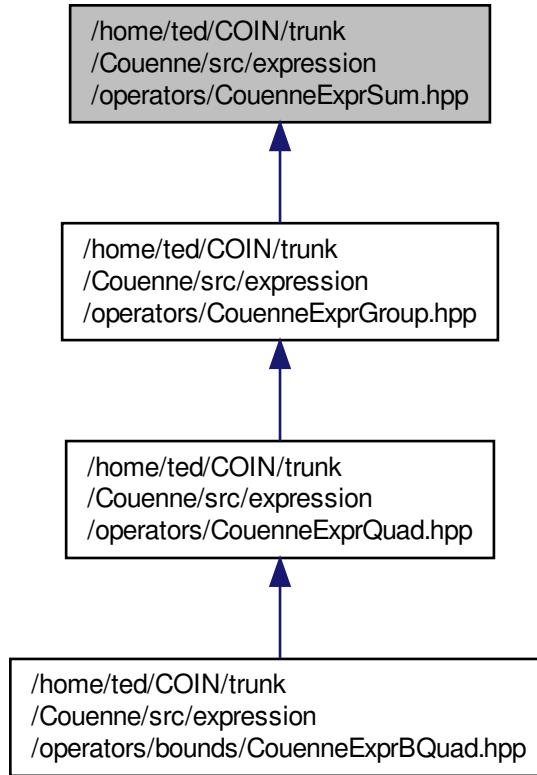
- namespace **Couenne**
general include file for different compilers

8.78 /home/ted/COIN/trunk/Couenne/src/expression/operators/CouenneExprSum.hpp File Reference

```
#include <vector>
#include "CouenneExprOp.hpp"
Include dependency graph for CouenneExprSum.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::exprSum](#)

class sum, $\sum_{i=1}^n f_i(x)$

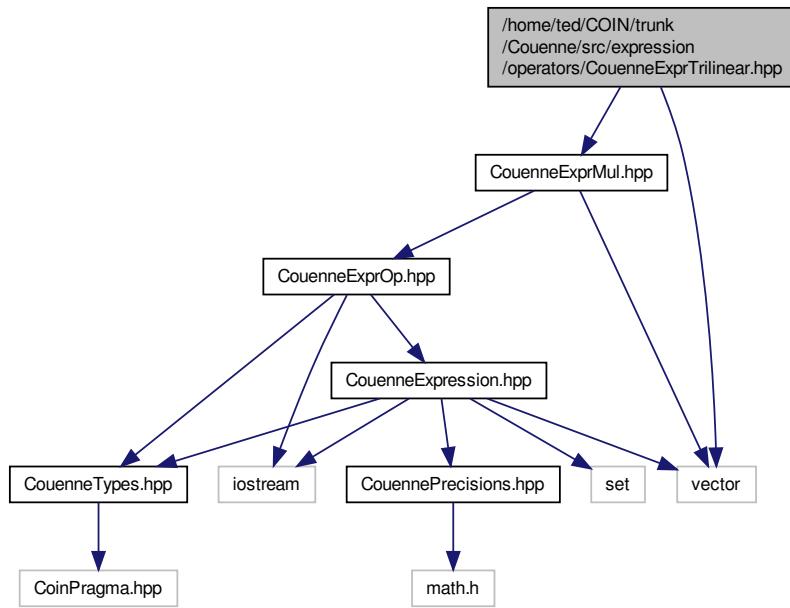
Namespaces

- namespace [Couenne](#)

general include file for different compilers

```
#include <vector>
#include "CouenneExprMul.hpp"
```

Include dependency graph for CouenneExprTrilinear.hpp:



Classes

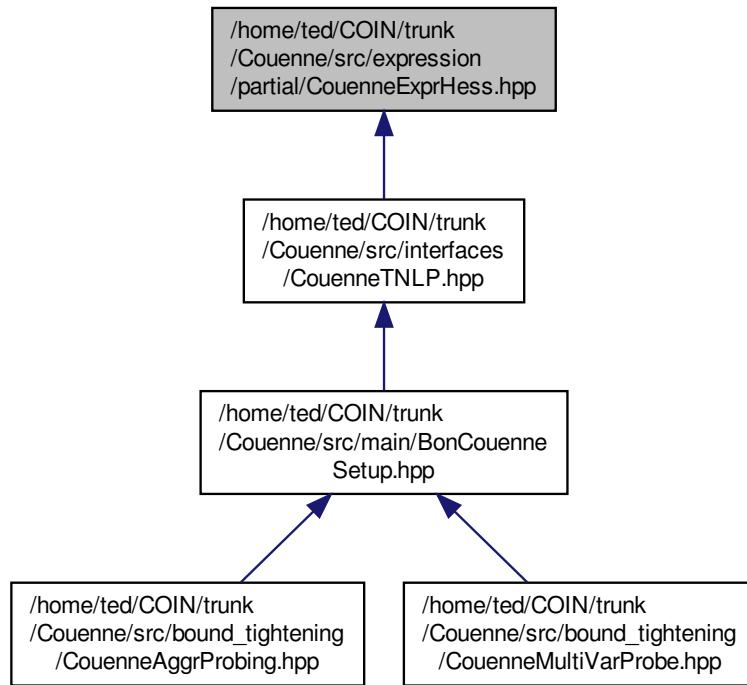
- class [Couenne::exprTrilinear](#)
class for multiplications

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.80 /home/ted/COIN/trunk/Couenne/src/expression/partial/CouenneExprHess.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

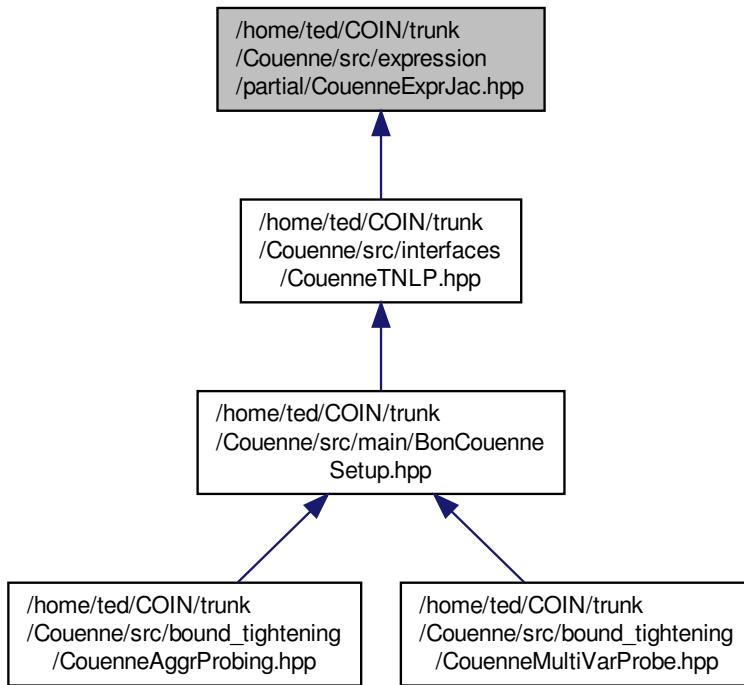
- class [Couenne::ExprHess](#)
expression matrices.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.81 /home/ted/COIN/trunk/Couenne/src/expression/partial/CouenneExprJac.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::ExprJac](#)
Jacobian of the problem (computed through [Couenne](#) expression classes).

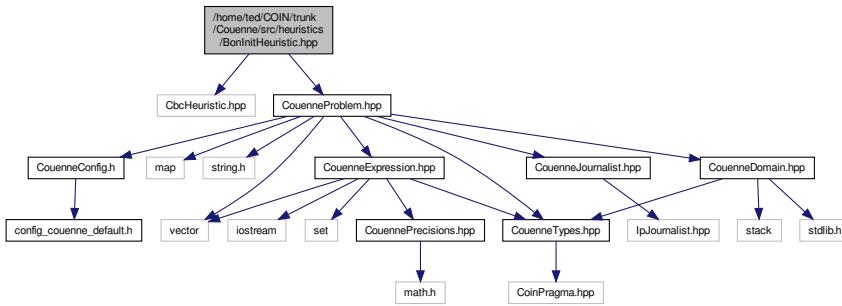
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.82 /home/ted/COIN/trunk/Couenne/src/heuristics/BonInitHeuristic.hpp File Reference

```
#include "CbcHeuristic.hpp"
#include "CouenneProblem.hpp"
```

Include dependency graph for BonInitHeuristic.hpp:



Classes

- class [Couenne::InitHeuristic](#)

A heuristic that stores the initial solution of the NLP.

Namespaces

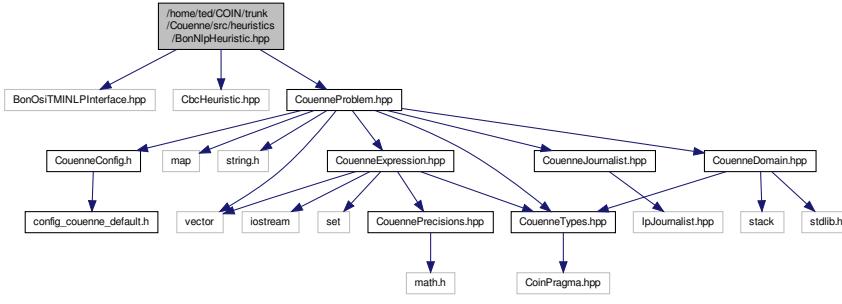
- namespace [Couenne](#)

general include file for different compilers

8.83 /home/ted/COIN/trunk/Couenne/src/heuristics/BonNlpHeuristic.hpp File Reference

```
#include "BonOsITMINLPInterface.hpp"
#include "CbcHeuristic.hpp"
#include "CouenneProblem.hpp"
```

Include dependency graph for BonNlpHeuristic.hpp:



Classes

- class [Couenne::NlpSolveHeuristic](#)

Namespaces

- namespace **Couenne**
general include file for different compilers

Variables

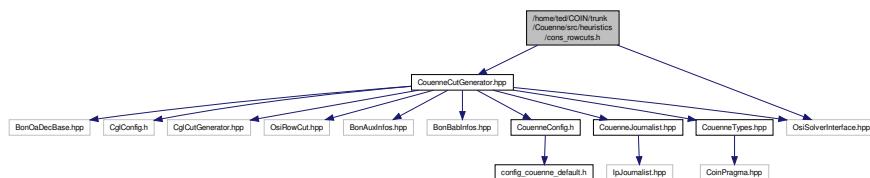
- const double **Couenne::maxNlpInf_0** = 1e-5

A heuristic to call an NlpSolver if all CouenneObjects are close to be satisfied (for other integer objects, rounding is performed, if SOS's are not satisfied it does not run).

8.84 /home/ted/COIN/trunk/Couenne/src/heuristics/cons_rowcuts.h File Reference

constraint handler for rowcuts constraints enables separation of convexification cuts during SCIP solution procedure

```
#include "CouenneCutGenerator.hpp"
#include "OsiSolverInterface.hpp"
Include dependency graph for cons_rowcuts.h:
```



8.84.1 Detailed Description

constraint handler for rowcuts constraints enables separation of convexification cuts during SCIP solution procedure

Id:

[cons_rowcuts.h](#) 688 2011-06-18 00:30:32Z pbelotti

Author

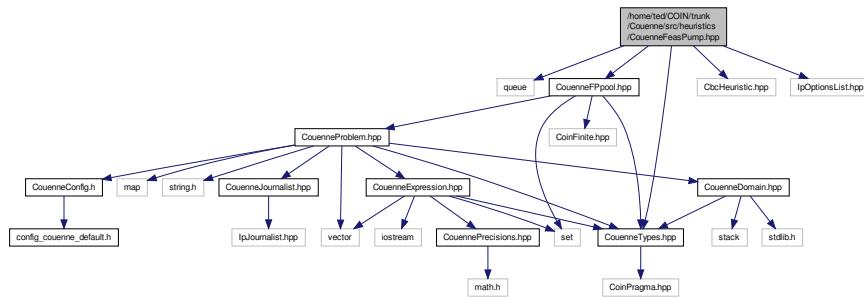
Pietro Belotti
 Timo Berthold This file is licensed under the Eclipse Public License (EPL)

Definition in file [cons_rowcuts.h](#).

8.85 /home/ted/COIN/trunk/Couenne/src/heuristics/CouenneFeasPump.hpp File Reference

```
#include <queue>
#include "CouenneTypes.hpp"
#include "CbcHeuristic.hpp"
#include "CouenneFPPool.hpp"
#include "IpOptionsList.hpp"
```

Include dependency graph for CouenneFeasPump.hpp:



Classes

- class [Couenne::CouenneFeasPump](#)

An implementation of the Feasibility pump that uses linearization and [Ipopt](#) to find the two sequences of points.

Namespaces

- namespace [Ipopt](#)
- namespace [Bonmin](#)
- namespace [Couenne](#)

general include file for different compilers

Functions

- double [fadingCoeff](#) (double a)

8.85.1 Function Documentation

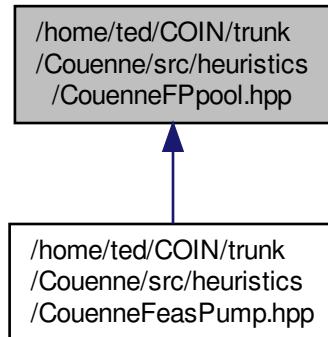
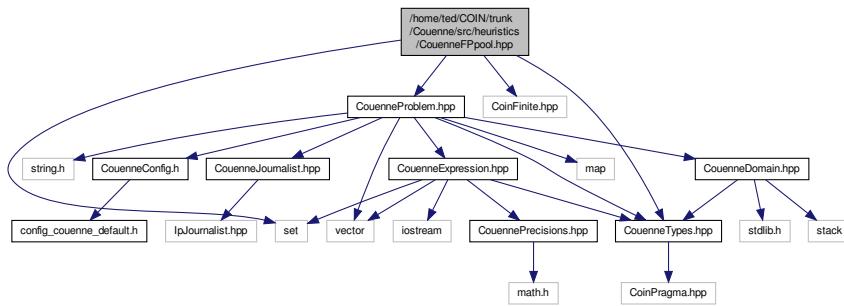
8.85.1.1 double fadingCoeff (double a) [inline]

Definition at line 35 of file CouenneFeasPump.hpp.

8.86 /home/ted/COIN/trunk/Couenne/src/heuristics/CouenneFPPool.hpp File Reference

```
#include <set>
#include "CouenneTypes.hpp"
#include "CoinFinite.hpp"
#include "CouenneProblem.hpp"
```

Include dependency graph for CouenneFPpool.hpp:



Classes

- class [Couenne::CouenneFPsolution](#)
Class containing a solution with infeasibility evaluation.
- class [Couenne::compareSol](#)
class for comparing solutions (used in tabu list)
- class [Couenne::CouenneFPpool](#)
Pool of solutions.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Enumerations

- enum [Couenne::what_to_compare](#) {
 [Couenne::SUM_NINF](#) = 0, [Couenne::SUM_INF](#), [Couenne::OBJVAL](#), [Couenne::ALL_VARS](#),
[Couenne::INTEGER_VARS](#) }
- what term to compare: the sum of infeasibilities, the sum of numbers of infeasible terms, or the objective function*

Functions

- bool [Couenne::operator<](#) (const CouenneFPsolution &one, const CouenneFPsolution &two)
compare, base version

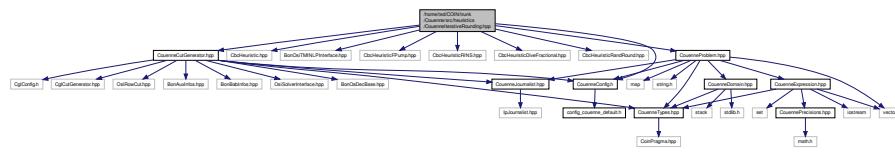
Variables

- static enum
[Couenne::what_to_compare](#) [Couenne::comparedTerm_](#)

8.87 /home/ted/COIN/trunk/Couenne/src/heuristics/CouennelterativeRounding.hpp File Reference

```
#include "CouenneConfig.h"
#include "CbcHeuristic.hpp"
#include "BonOsiTMINLPIterface.hpp"
#include "CbcHeuristicFPump.hpp"
#include "CbcHeuristicRINS.hpp"
#include "CbcHeuristicDiveFractional.hpp"
#include "CbcHeuristicRandRound.hpp"
#include "CouenneCutGenerator.hpp"
#include "CouenneProblem.hpp"
```

Include dependency graph for CouennelterativeRounding.hpp:



Classes

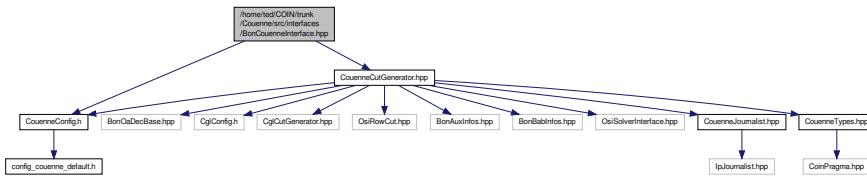
- class [Couenne::CouennelterativeRounding](#)
An iterative rounding heuristic, tailored for nonconvex MINLPs.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.88 /home/ted/COIN/trunk/Couenne/src/interfaces/BonCouenneInterface.hpp File Reference

```
#include "CouenneConfig.h"
#include "CouenneCutGenerator.hpp"
Include dependency graph for BonCouenneInterface.hpp:
```



Classes

- class [Couenne::CouenneInterface](#)

Namespaces

- namespace [Bonmin](#)
- namespace [Couenne](#)
general include file for different compilers

Macros

- #define [AmplInterface](#) OsiTMINLPInterface

8.88.1 Macro Definition Documentation

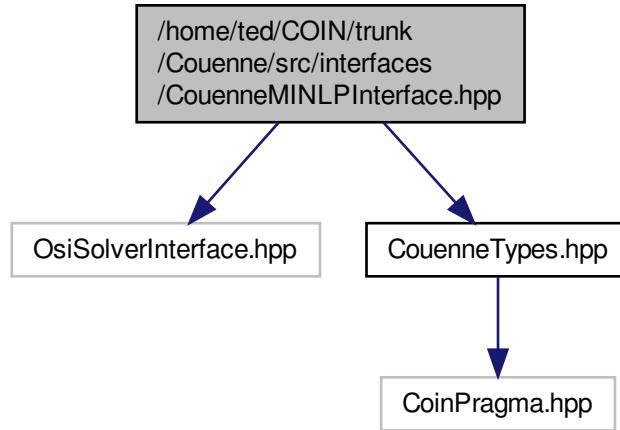
8.88.1.1 #define AmplInterface OsiTMINLPInterface

Definition at line 24 of file BonCouenneInterface.hpp.

8.89 /home/ted/COIN/trunk/Couenne/src/interfaces/CouenneMINLPInterface.hpp File Reference

```
#include "OsiSolverInterface.hpp"
#include "CouenneTypes.hpp"
```

Include dependency graph for CouenneMINLPIface.hpp:



Classes

- class [Couenne::CouenneMINLPIface](#)

This is class provides an [Osi](#) interface for a Mixed Integer Linear Program expressed as a TMINLP (so that we can use it for example as the continuous solver in Cbc).

Namespaces

- namespace [Ipopt](#)
- namespace [Couenne](#)
general include file for different compilers

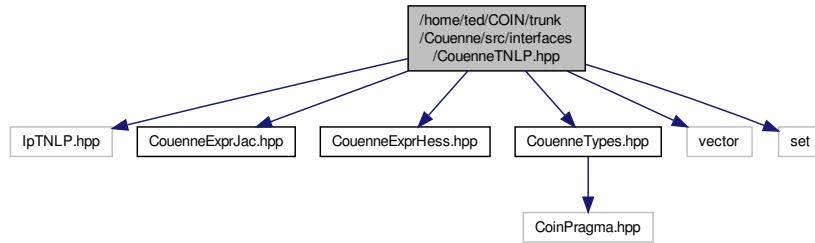
Enumerations

- enum [Couenne::Solver](#) { [Couenne::Elpopt](#) = 0, [Couenne::EFilterSQP](#), [Couenne::EAll](#) }
Solvers for solving nonlinear programs.

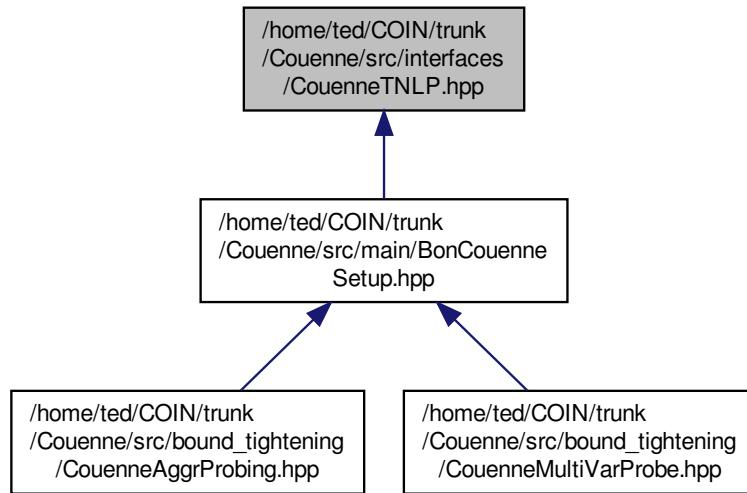
8.90 /home/ted/COIN/trunk/Couenne/src/interfaces/CouenneTNLP.hpp File Reference

```
#include "IpTNLP.hpp"
#include "CouenneExprJac.hpp"
#include "CouenneExprHess.hpp"
#include "CouenneTypes.hpp"
#include <vector>
#include <set>
```

Include dependency graph for CouenneTNLP.hpp:



This graph shows which files directly or indirectly include this file:



Classes

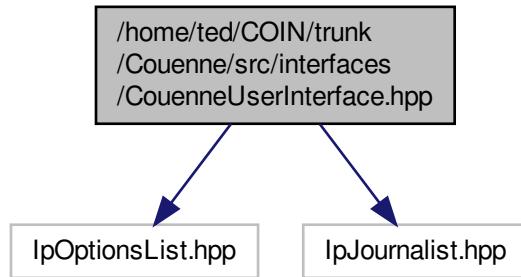
- class [Couenne::CouenneTNLP](#)
Class for handling NLPs using [CouenneProblem](#).

Namespaces

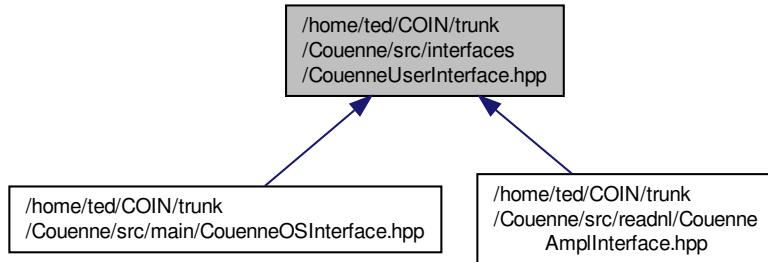
- namespace [Couenne](#)
general include file for different compilers

8.91 /home/ted/COIN/trunk/Couenne/src/interfaces/CouenneUserInterface.hpp File Reference

```
#include "IpOptionsList.hpp"
#include "IpJournalist.hpp"
Include dependency graph for CouenneUserInterface.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

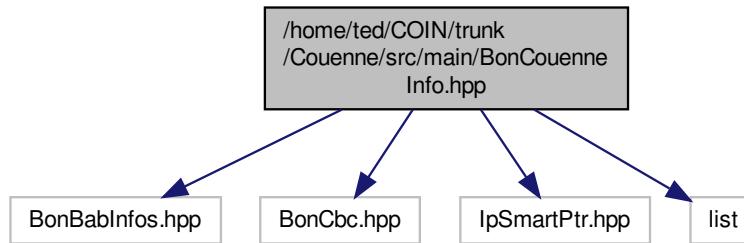
- class [Couenne::CouenneUserInterface](#)

Namespaces

- namespace [Bonmin](#)
- namespace [Couenne](#)
general include file for different compilers

8.92 /home/ted/COIN/trunk/Couenne/src/main/BonCouenneInfo.hpp File Reference

```
#include "BonBabInfos.hpp"
#include "BonCbc.hpp"
#include "IpSmartPtr.hpp"
#include <list>
Include dependency graph for BonCouenneInfo.hpp:
```



Classes

- class [Couenne::CouenneInfo](#)
Bonmin class for passing info between components of branch-and-cuts.
- class [Couenne::CouenneInfo::NlpSolution](#)
Class for storing an Nlp Solution.

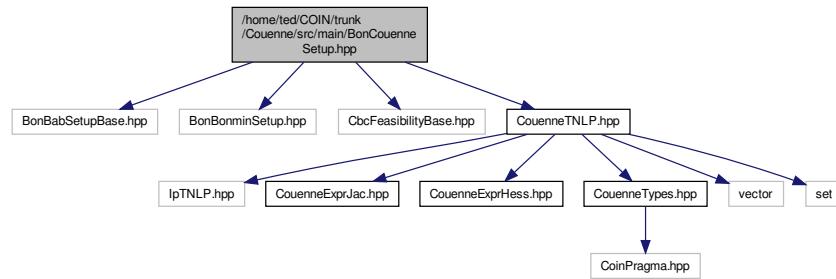
Namespaces

- namespace [Couenne](#)
general include file for different compilers

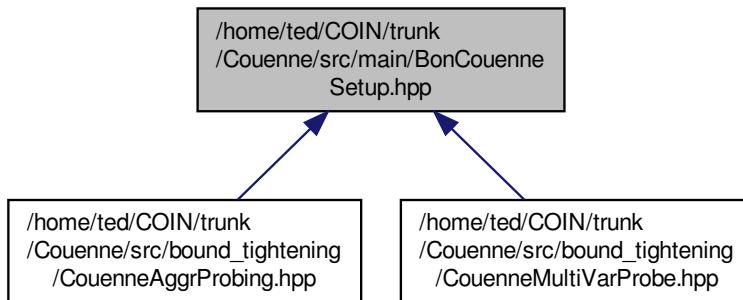
8.93 /home/ted/COIN/trunk/Couenne/src/main/BonCouenneSetup.hpp File Reference

```
#include "BonBabSetupBase.hpp"
#include "BonBonminSetup.hpp"
#include "CbcFeasibilityBase.hpp"
#include "CouenneTNLP.hpp"
```

Include dependency graph for BonCouenneSetup.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::SmartAsl](#)
- class [Couenne::CouenneSetup](#)

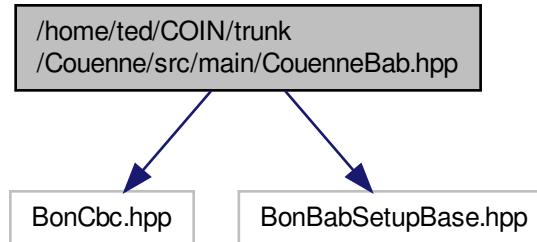
Namespaces

- namespace [Bonmin](#)
- namespace [Couenne](#)
general include file for different compilers

8.94 /home/ted/COIN/trunk/Couenne/src/main/CouenneBab.hpp File Reference

```
#include "BonCbc.hpp"
#include "BonBabSetupBase.hpp"
```

Include dependency graph for CouenneBab.hpp:



Classes

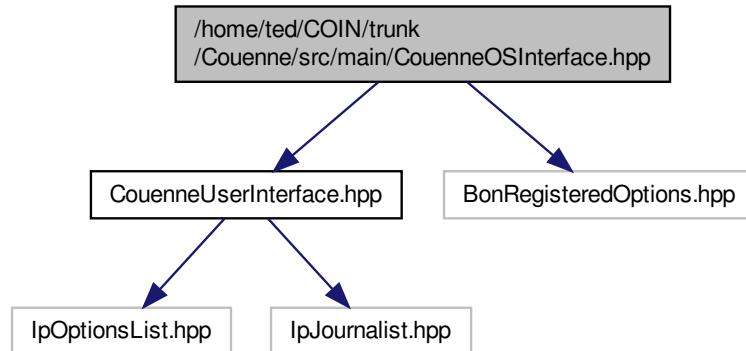
- class [Couenne::CouenneBab](#)

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.95 /home/ted/COIN/trunk/Couenne/src/main/CouenneOSInterface.hpp File Reference

```
#include "CouenneUserInterface.hpp"
#include "BonRegisteredOptions.hpp"
Include dependency graph for CouenneOSInterface.hpp:
```



Classes

- class [Couenne::CouenneOSInterface](#)

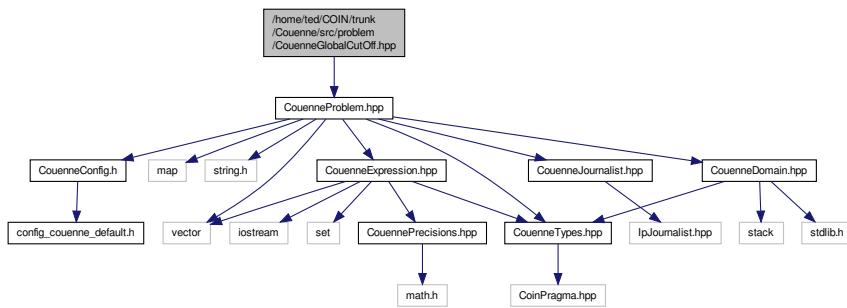
Namespaces

- namespace [Bonmin](#)
- namespace [Ipopt](#)
- namespace [Couenne](#)

general include file for different compilers

8.96 /home/ted/COIN/trunk/Couenne/src/problem/CouenneGlobalCutOff.hpp File Reference

```
#include "CouenneProblem.hpp"
Include dependency graph for CouenneGlobalCutOff.hpp:
```



Classes

- class [Couenne::GlobalCutOff](#)

Namespaces

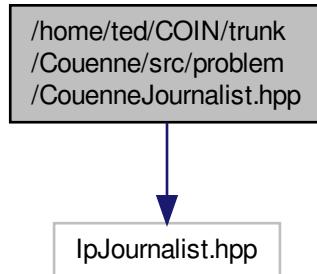
- namespace [Couenne](#)

general include file for different compilers

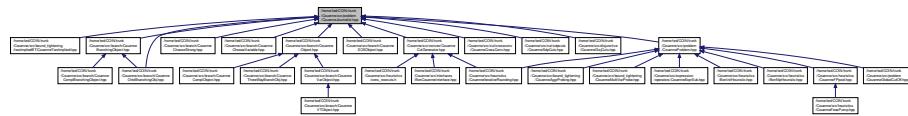
8.97 /home/ted/COIN/trunk/Couenne/src/problem/CouenneJournalist.hpp File Reference

```
#include "IpJournalist.hpp"
```

Include dependency graph for CouenneJournalist.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [Couenne](#)
general include file for different compilers

Functions

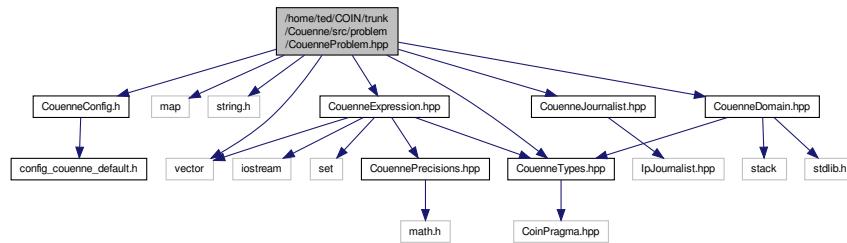
- const Ipopt::EJournalCategory [Couenne::J_BRANCHING](#) (Ipopt::J_USER1)
- const Ipopt::EJournalCategory [Couenne::J_BOUNDTIGHTENING](#) (Ipopt::J_USER2)
- const Ipopt::EJournalCategory [Couenne::J_CONVEXIFYING](#) (Ipopt::J_USER3)
- const Ipopt::EJournalCategory [Couenne::J_PROBLEM](#) (Ipopt::J_USER4)
- const Ipopt::EJournalCategory [Couenne::J_NLPHEURISTIC](#) (Ipopt::J_USER5)
- const Ipopt::EJournalCategory [Couenne::J_DISJCUTS](#) (Ipopt::J_USER6)
- const Ipopt::EJournalCategory [Couenne::J_REFORMULATE](#) (Ipopt::J_USER7)
- const Ipopt::EJournalCategory [Couenne::J_COUNNE](#) (Ipopt::J_USER8)

8.98 /home/ted/COIN/trunk/Couenne/src/problem/CouenneProblem.hpp File Reference

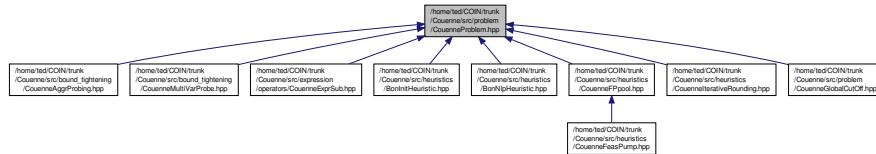
```
#include <vector>
```

```
#include <map>
#include <string.h>
#include "CouenneConfig.h"
#include "CouenneTypes.hpp"
#include "CouenneExpression.hpp"
#include "CouenneJournalist.hpp"
#include "CouenneDomain.hpp"

Include dependency graph for CouenneProblem.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Node](#)
- struct [myclass0](#)
- struct [myclass](#)
- struct [less_than_str](#)
- class [Couenne::CouenneProblem](#)

Class for MINLP problems with symbolic information.

Namespaces

- namespace [Ipopt](#)
- namespace [Bonmin](#)
- namespace [Couenne](#)

general include file for different compilers

Macros

- #define [FM_TRACE_OPTSOL](#)

- #define FM_CHECKNLP2
- #define COUENNE_EPS_SYMM 1e-8

Enumerations

- enum Couenne::TrilinDecompType { Couenne::rAI, Couenne::treeDecomp, Couenne::bi_tri, Couenne::tri_bi }

Variables

- const CouNumber Couenne::feas_tolerance_default = 1e-5

8.98.1 Macro Definition Documentation

8.98.1.1 #define FM_TRACE_OPTSOL

Definition at line 15 of file CouenneProblem.hpp.

8.98.1.2 #define FM_CHECKNLP2

Definition at line 16 of file CouenneProblem.hpp.

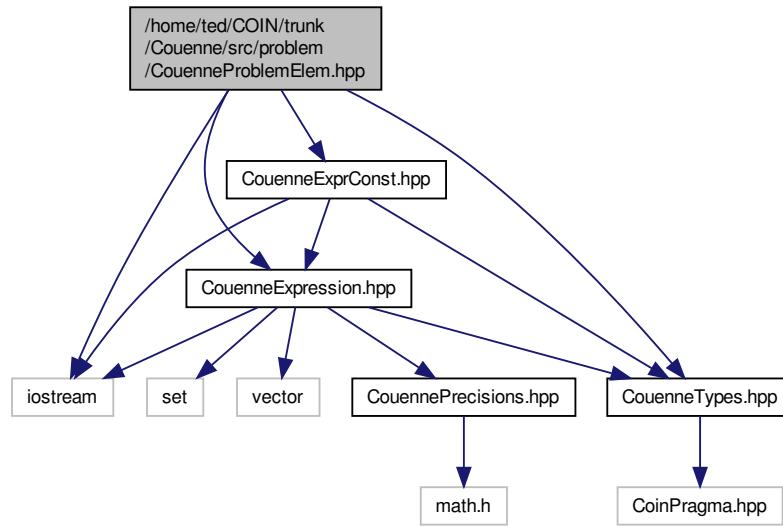
8.98.1.3 #define COUENNE_EPS_SYMM 1e-8

Definition at line 74 of file CouenneProblem.hpp.

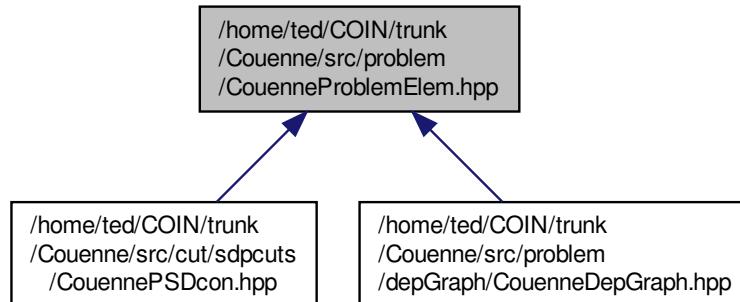
8.99 /home/ted/COIN/trunk/Couenne/src/problem/CouenneProblemElem.hpp File Reference

```
#include <iostream>
#include "CouenneTypes.hpp"
#include "CouenneExpression.hpp"
#include "CouenneExprConst.hpp"
```

Include dependency graph for CouenneProblemElem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Couenne::CouenneConstraint](#)
Class to represent nonlinear constraints.
- class [Couenne::CouenneObjective](#)
Objective function.

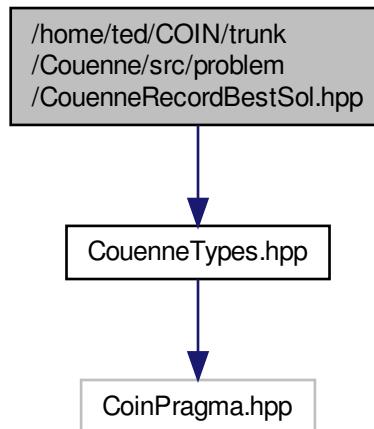
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.100 /home/ted/COIN/trunk/Couenne/src/problem/CouenneRecordBestSol.hpp File Reference

```
#include "CouenneTypes.hpp"
```

Include dependency graph for CouenneRecordBestSol.hpp:



Classes

- class [Couenne::CouenneRecordBestSol](#)

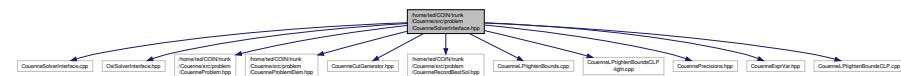
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.101 /home/ted/COIN/trunk/Couenne/src/problem/CouenneSolverInterface.hpp File Reference

```
#include "CouenneSolverInterface.cpp"
#include "CouenneLPtightenBounds.cpp"
#include "CouenneLPtightenBoundsCLP-light.cpp"
#include "CouenneLPtightenBoundsCLP.cpp"
```

Include dependency graph for CouenneSolverInterface.hpp:



Classes

- class [Couenne::CouenneSolverInterface< T >](#)

Solver interface class with a pointer to a [Couenne](#) cut generator.

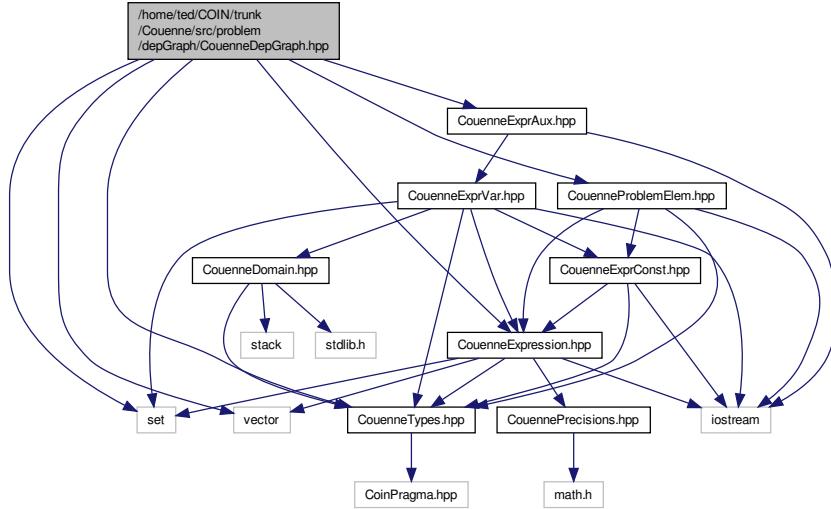
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.102 /home/ted/COIN/trunk/Couenne/src/problem/depGraph/CouenneDepGraph.hpp File Reference

```
#include <vector>
#include <set>
#include "CouenneTypes.hpp"
#include "CouenneExpression.hpp"
#include "CouenneExprAux.hpp"
#include "CouenneProblemElem.hpp"
#include "CouenneDomain.hpp"
```

Include dependency graph for CouenneDepGraph.hpp:



Classes

- struct [Couenne::compNode](#)

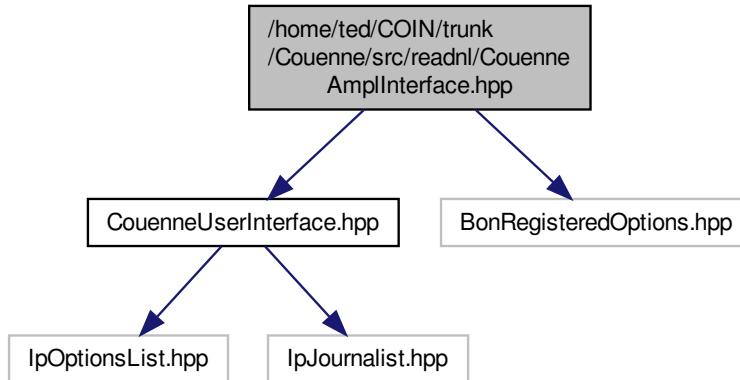
- class [Couenne::DepNode](#)
structure for comparing nodes in the dependence graph
- class [Couenne::DepGraph](#)
vertex of a dependence graph.
- class [Couenne::DepGraph](#)
Dependence graph.

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.103 /home/ted/COIN/trunk/Couenne/src/readnl/CouenneAmplInterface.hpp File Reference

```
#include "CouenneUserInterface.hpp"
#include "BonRegisteredOptions.hpp"
Include dependency graph for CouenneAmplInterface.hpp:
```



Classes

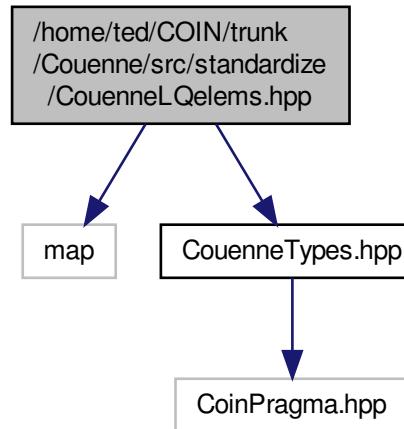
- class [Couenne::CouenneAmplInterface](#)

Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.104 /home/ted/COIN/trunk/Couenne/src/standardize/CouenneLQelems.hpp File Reference

```
#include <map>
#include "CouenneTypes.hpp"
Include dependency graph for CouenneLQelems.hpp:
```



Classes

- class [Couenne::quadElem](#)
- class [Couenne::LinMap](#)
- class [Couenne::QuadMap](#)

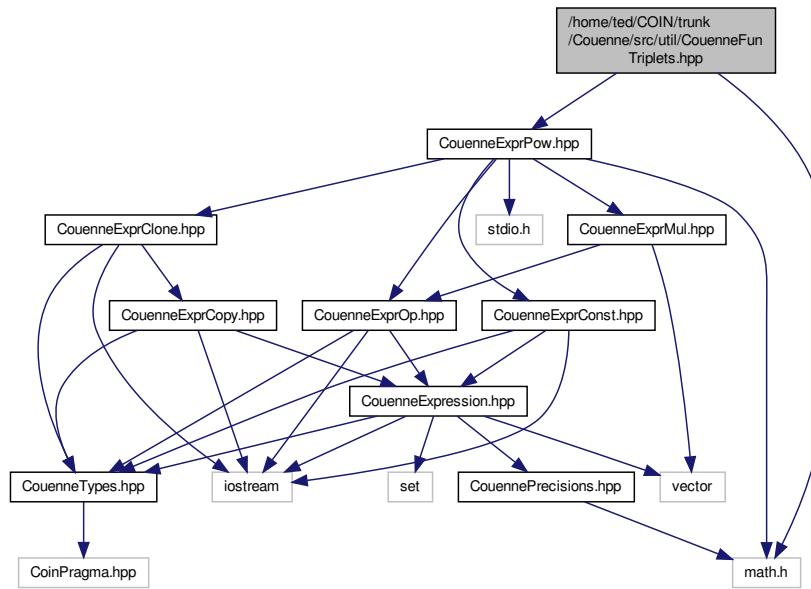
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.105 /home/ted/COIN/trunk/Couenne/src/util/CouenneFunTriplets.hpp File Reference

```
#include <math.h>
#include "CouenneExprPow.hpp"
```

Include dependency graph for CouenneFunTriplets.hpp:



Classes

- class [Couenne::funtriplet](#)
- class [Couenne::simpletriplet](#)
- class [Couenne::powertriplet](#)
- class [Couenne::kpowertriplet](#)

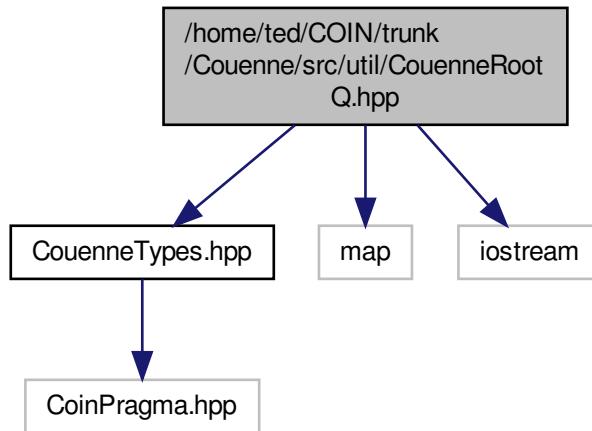
Namespaces

- namespace [Couenne](#)
general include file for different compilers

8.106 /home/ted/COIN/trunk/Couenne/src/util/CouenneRootQ.hpp File Reference

```
#include "CouenneTypes.hpp"
#include <map>
#include <iostream>
```

Include dependency graph for CouenneRootQ.hpp:



Classes

- class [Couenne::Qroot](#)
class that stores result of previous calls to rootQ into a map for faster access

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Functions

- CouNumber [Couenne::rootQ](#) (int k)
Find roots of polynomial \$Q^k(x) = \sum_{i=1}^{2k} ix^{i-1}.

8.107 /home/ted/COIN/trunk/Couenne/src/util/CouenneSparseMatrix.hpp File Reference

Classes

- class [Couenne::CouenneSparseMatrix](#)
Class for sparse Matrixs (used in modifying distances in FP)

Namespaces

- namespace [Couenne](#)
general include file for different compilers

Index

~CouenneAggrProbing
 Couenne::CouenneAggrProbing, 43

~CouenneAmplInterface
 Couenne::CouenneAmplInterface, 47

~CouenneBTPerIndicator
 Couenne::CouenneBTPerIndicator, 55

~CouenneBab
 Couenne::CouenneBab, 48

~CouenneChooseStrong
 Couenne::CouenneChooseStrong, 58

~CouenneChooseVariable
 Couenne::CouenneChooseVariable, 61

~CouenneComplObject
 Couenne::CouenneComplObject, 66

~CouenneConstraint
 Couenne::CouenneConstraint, 69

~CouenneCrossConv
 Couenne::CouenneCrossConv, 72

~CouenneCutGenerator
 Couenne::CouenneCutGenerator, 76

~CouenneDisjCuts
 Couenne::CouenneDisjCuts, 83

~CouenneExprMatrix
 Couenne::CouenneExprMatrix, 87

~CouenneFPSolution
 Couenne::CouenneFPSolution, 100

~CouenneFeasPump
 Couenne::CouenneFeasPump, 91

~CouenneFixPoint
 Couenne::CouenneFixPoint, 95

~CouenneInfo
 Couenne::CouenneInfo, 103

~CouenneInterface
 Couenne::CouenneInterface, 105

~CouennerterativeRounding
 Couenne::CouennerterativeRounding, 107

~CouenneMultiVarProbe
 Couenne::CouenneMultiVarProbe, 111

~CouenneOSInterface
 Couenne::CouenneOSInterface, 125

~CouenneObject
 Couenne::CouenneObject, 116

~CouenneObjective
 Couenne::CouenneObjective, 121

~CouennePSDcon
 Couenne::CouennePSDcon, 155

~CouenneProblem
 Couenne::CouenneProblem, 135

~CouenneRecordBestSol
 Couenne::CouenneRecordBestSol, 158

~CouenneScalar
 Couenne::CouenneScalar, 161

~CouenneSdpCuts
 Couenne::CouenneSdpCuts, 164

~CouenneSetup
 Couenne::CouenneSetup, 167

~CouenneSolverInterface
 Couenne::CouenneSolverInterface, 170

~CouenneSparseBndVec
 Couenne::CouenneSparseBndVec, 178

~CouenneSparseMatrix
 Couenne::CouenneSparseMatrix, 180

~CouenneSparseVector
 Couenne::CouenneSparseVector, 182

~CouenneTNLP
 Couenne::CouenneTNLP, 187

~CouenneTwoImplied
 Couenne::CouenneTwoImplied, 193

~CouenneUserInterface
 Couenne::CouenneUserInterface, 195

~CouenneVTOBJECT
 Couenne::CouenneVTOBJECT, 202

~CouenneVarObject
 Couenne::CouenneVarObject, 198

~DepGraph
 Couenne::DepGraph, 204

~DepNode
 Couenne::DepNode, 208

~Domain
 Couenne::Domain, 211

~DomainPoint
 Couenne::DomainPoint, 214

~ExprHess
 Couenne::ExprHess, 287

~ExprJac
 Couenne::ExprJac, 296

~GlobalCutOff
 Couenne::GlobalCutOff, 405

~InitHeuristic
 Couenne::InitHeuristic, 406

~Nauty
 Nauty, 414

~NlpSolution
 Couenne::CouennerInfo::NlpSolution, 416

~NlpSolveHeuristic
 Couenne::NlpSolveHeuristic, 417

~Qroot
 Couenne::Qroot, 425

~SmartAsl
 Couenne::SmartAsl, 430

~exprAux
 Couenne::exprAux, 225

~exprClone
 Couenne::exprClone, 237

~exprCopy
 Couenne::exprCopy, 245

~exprGroup
 Couenne::exprGroup, 283

~exprLBQuad
 Couenne::exprLBQuad, 305

~exprOp
 Couenne::exprOp, 334

~exprStore
 Couenne::exprStore, 363

~exprSum
 Couenne::exprSum, 370

~exprUBQuad
 Couenne::exprUBQuad, 383

~exprUnary
 Couenne::exprUnary, 388

~exprVar
 Couenne::exprVar, 398

~expression
 Couenne::expression, 264

~funtriplet
 Couenne::funtriplet, 404

~kpowertriplet
 Couenne::kpowertriplet, 409

~powertriplet
 Couenne::powertriplet, 421

~simpletriplet
 Couenne::simpletriplet, 428

/home/ted/COIN/trunk/Couenne/src/CouenneConfig.h, 456

/home/ted/COIN/trunk/Couenne/src/bound_tightening/- CouenneAggrProbing.hpp, 433

/home/ted/COIN/trunk/Couenne/src/bound_tightening/- CouenneBTPerIndicator.hpp, 434

/home/ted/COIN/trunk/Couenne/src/bound_tightening/- CouenneFixPoint.hpp, 435

/home/ted/COIN/trunk/Couenne/src/bound_tightening/- CouenneInfeasCut.hpp, 435

/home/ted/COIN/trunk/Couenne/src/bound_tightening/- CouenneMultiVarProbe.hpp, 436

/home/ted/COIN/trunk/Couenne/src/bound_tightening/- CouenneSparseBndVec.hpp, 437

/home/ted/COIN/trunk/Couenne/src/bound_tightening/two- ImpliedBT/CouenneTwoImplied.hpp, 437

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- BranchingObject.hpp, 438

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- ChooseStrong.hpp, 440

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- ChooseVariable.hpp, 441

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- ComplBranchingObject.hpp, 443

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- ComplObject.hpp, 444

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- Object.hpp, 444

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- OrbitBranchingObj.hpp, 446

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- OrbitObj.hpp, 447

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- Projections.hpp, 447

/home/ted/COIN/trunk/Couenne/src/branch/CouenneSO- SObject.hpp, 448

/home/ted/COIN/trunk/Couenne/src/branch/Couenne- ThreeWayBranchObj.hpp, 449

/home/ted/COIN/trunk/Couenne/src/branch/CouenneVT- Object.hpp, 452

/home/ted/COIN/trunk/Couenne/src/branch/CouenneVar- Object.hpp, 451

/home/ted/COIN/trunk/Couenne/src/branch/Nauty.h, 452

/home/ted/COIN/trunk/Couenne/src/config_couenne_- default.h, 453

/home/ted/COIN/trunk/Couenne/src/config_default.h, 454

/home/ted/COIN/trunk/Couenne/src/convex/CouenneCut- Generator.hpp, 455

/home/ted/COIN/trunk/Couenne/src/cut/crossconv/- CouenneCrossConv.hpp, 456

/home/ted/COIN/trunk/Couenne/src/cut/ellipcuts/Couenne- EllipCuts.hpp, 457

/home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/Couenne- Matrix.hpp, 457

/home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/Couenne- PSDcon.hpp, 458

/home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/Couenne- SdpCuts.hpp, 459

/home/ted/COIN/trunk/Couenne/src/cut/sdpcuts/dsyevx_- wrapper.hpp, 460

/home/ted/COIN/trunk/Couenne/src/disjunctive/Couenne- DisjCuts.hpp, 460

/home/ted/COIN/trunk/Couenne/src/expression/CouExpr.- hpp, 478

/home/ted/COIN/trunk/Couenne/src/expression/Couenne- Domain.hpp, 461

/home/ted/COIN/trunk/Couenne/src/expression/Couenne- ExprAux.hpp, 462

/home/ted/COIN/trunk/Couenne/src/expression/Couenne- ExprBound.hpp, 464

/home/ted/COIN/trunk/Couenne/src/expression/Couenne- ExprClone.hpp, 465

/home/ted/COIN/trunk/Couenne/src/expression/Couenne- ExprConst.hpp, 466

/home/ted/COIN/trunk/Couenne/src/expression/Couenne- ExprCopy.hpp, 467

/home/ted/COIN/trunk/Couenne/src/expression/Couenne- ExprVar.hpp, 469

/home/ted/COIN/trunk/Couenne/src/expression/Couenne-ExprOp.hpp, 469
/home/ted/COIN/trunk/Couenne/src/expression/Couenne-ExprStore.hpp, 471
/home/ted/COIN/trunk/Couenne/src/expression/Couenne-ExprUnary.hpp, 472
/home/ted/COIN/trunk/Couenne/src/expression/Couenne-ExprVar.hpp, 473
/home/ted/COIN/trunk/Couenne/src/expression/Couenne-Expression.hpp, 468
/home/ted/COIN/trunk/Couenne/src/expression/Couenne-Precisions.hpp, 474
/home/ted/COIN/trunk/Couenne/src/expression/Couenne-Types.hpp, 476
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprAbs.hpp, 484
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprBinProd.hpp, 484
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprCeil.hpp, 486
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprCos.hpp, 487
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprDiv.hpp, 487
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprEvenPow.hpp, 489
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprExp.hpp, 489
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprFloor.hpp, 491
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprGroup.hpp, 491
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprIf.hpp, 493
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprInv.hpp, 493
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprLog.hpp, 495
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprMax.hpp, 496
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprMin.hpp, 497
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprMul.hpp, 497
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprMultiLin.hpp, 498
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprNorm.hpp, 499
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprOddPow.hpp, 500
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprOpp.hpp, 501
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprPWLinear.hpp, 504
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprPow.hpp, 502
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprQuad.hpp, 504
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprSignPow.hpp, 506
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprSin.hpp, 506
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprSub.hpp, 508
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprSum.hpp, 509
/home/ted/COIN/trunk/Couenne/src/expression/operators/- CouenneExprTrilinear.hpp, 510
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/- CouenneExprBCos.hpp, 479
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/- CouenneExprBDiv.hpp, 480
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/- CouenneExprBMul.hpp, 481
/home/ted/COIN/trunk/Couenne/src/expression/operators/bounds/- CouenneExprBQuad.hpp, 482
/home/ted/COIN/trunk/Couenne/src/expression/partial/- CouenneExprHess.hpp, 512
/home/ted/COIN/trunk/Couenne/src/expression/partial/- CouenneExprJac.hpp, 513
/home/ted/COIN/trunk/Couenne/src/heuristics/BonInit-Heuristic.hpp, 513
/home/ted/COIN/trunk/Couenne/src/heuristics/BonNlp-Heuristic.hpp, 514
/home/ted/COIN/trunk/Couenne/src/heuristics/CouenneF- Ppool.hpp, 516
/home/ted/COIN/trunk/Couenne/src/heuristics/Couenne-FeasPump.hpp, 515
/home/ted/COIN/trunk/Couenne/src/heuristics/Couenne-IterativeRounding.hpp, 518
/home/ted/COIN/trunk/Couenne/src/heuristics/cons_-rowcuts.h, 515
/home/ted/COIN/trunk/Couenne/src/interfaces/Bon- CouenneInterface.hpp, 519
/home/ted/COIN/trunk/Couenne/src/interfaces/Couenne- MINLPInterface.hpp, 519
/home/ted/COIN/trunk/Couenne/src/interfaces/CouenneT- NLP.hpp, 520
/home/ted/COIN/trunk/Couenne/src/interfaces/Couenne- UserInterface.hpp, 522
/home/ted/COIN/trunk/Couenne/src/main/BonCouenne- Info.hpp, 523
/home/ted/COIN/trunk/Couenne/src/main/BonCouenne- Setup.hpp, 523
/home/ted/COIN/trunk/Couenne/src/main/CouenneBab.- hpp, 524

/home/ted/COIN/trunk/Couenne/src/main/CouenneOS-Interface.hpp, 525
/home/ted/COIN/trunk/Couenne/src/problem/Couenne-GlobalCutOff.hpp, 526
/home/ted/COIN/trunk/Couenne/src/problem/Couenne-Journalist.hpp, 526
/home/ted/COIN/trunk/Couenne/src/problem/Couenne-Problem.hpp, 527
/home/ted/COIN/trunk/Couenne/src/problem/Couenne-ProblemElem.hpp, 529
/home/ted/COIN/trunk/Couenne/src/problem/Couenne-RecordBestSol.hpp, 531
/home/ted/COIN/trunk/Couenne/src/problem/Couenne-SolverInterface.hpp, 531
/home/ted/COIN/trunk/Couenne/src/problem/depGraph/- CouenneDepGraph.hpp, 532
/home/ted/COIN/trunk/Couenne/src/readnl/Couenne-AmplInterface.hpp, 533
/home/ted/COIN/trunk/Couenne/src/standardize/Couenne-LQelems.hpp, 534
/home/ted/COIN/trunk/Couenne/src/util/CouenneFun-Triplets.hpp, 534
/home/ted/COIN/trunk/Couenne/src/util/CouenneRootQ.-hpp, 535
/home/ted/COIN/trunk/Couenne/src/util/CouenneSparse-Matrix.hpp, 536

AFFINE
 Couenne, 30

ALL_VARS
 Couenne, 31

AROUND_CURPOINT
 Couenne, 29

AUX
 Couenne, 28

AUX_EQ
 Couenne::expression, 264

AUX_GEQ
 Couenne::expression, 264

AUX_LEQ
 Couenne::expression, 264

AUX_UNDEF
 Couenne::expression, 264

AGGR_MUL
 CouenneObject.hpp, 446

activeCols_
 Couenne::CouenneDisjCuts, 86

activeRows_
 Couenne::CouenneDisjCuts, 86

add_element
 Couenne::CouenneExprMatrix, 88
 Couenne::CouenneSparseVector, 182

addAuxiliary
 Couenne::CouenneProblem, 139

addBabPlugins
 Couenne::CouenneUserInterface, 196

addEQConstraint
 Couenne::CouenneProblem, 139

addElement
 Nauty, 414

addEnvelope
 Couenne::CouenneCutGenerator, 77

addGEConstraint
 Couenne::CouenneProblem, 139

addLEConstraint
 Couenne::CouenneProblem, 139

addMilpCutGenerators
 Couenne::CouenneSetup, 167

addObjective
 Couenne::CouenneProblem, 139

addPowEnvelope
 Couenne, 35

addPreviousCut_
 Couenne::CouenneDisjCuts, 86

addRNGConstraint
 Couenne::CouenneProblem, 139

addSegment
 Couenne::CouenneCutGenerator, 77

addSolution
 Couenne::CouenneInfo, 103

addTangent
 Couenne::CouenneCutGenerator, 77

addToTimer
 Couenne::CouenneBTPerfIndicator, 55

addVariable
 Couenne::CouenneProblem, 139

addViolated
 Couenne::CouenneCutGenerator, 76

addviolated_
 Couenne::CouenneCutGenerator, 78

aggressiveBT
 Couenne::CouenneProblem, 141

alpha_
 Couenne::CouenneObject, 118

alphaConvexify
 Couenne::exprQuad, 353

AmplInterface
 BonCouenneInterface.hpp, 519

analyzeSparsity
 Couenne::CouenneProblem, 145

appName
 Couenne::CouenneInterface, 105

applyColCuts
 Couenne::CouenneDisjCuts, 84

ArgList
 Couenne::exprCopy, 247
 Couenne::expression, 265
 Couenne::exprOp, 335

ArgPtr
 Couenne::exprCopy, 247
 Couenne::expression, 265
 Couenne::exprUnary, 389

arglist_
 Couenne::exprOp, 337

Argument
 Couenne::exprCopy, 247
 Couenne::expression, 265
 Couenne::exprUnary, 389

argument_
 Couenne::exprUnary, 391

asl
 Couenne::SmartAsl, 430

asl_
 Couenne::CouenneProblem, 151

AuxSet
 Couenne::CouenneProblem, 138

auxSet_
 Couenne::CouenneProblem, 148

auxSign
 Couenne::expression, 264

auxiliarize
 Couenne::CouenneProblem, 142

BALANCED
 Couenne::CouenneObject, 115

BRANCH_NONE
 Couenne, 27

BR_MULT
 CouenneExprDiv.hpp, 488

BR_NEXT_ZERO
 CouenneExprDiv.hpp, 488

BabPtr_
 Couenne::CouenneCutGenerator, 79

balancedMul
 Couenne::exprBinProd, 231
 Couenne::exprMultiLin, 326

bestBound
 Couenne::CouenneBab, 49

bestObj
 Couenne::CouenneBab, 49
 Couenne::CouenneProblem, 139

bestObj_
 Couenne::CouenneProblem, 149

bestSol
 Couenne::CouenneProblem, 139

bestSolution
 Couenne::CouenneBab, 49

bi_tri
 Couenne, 32

Body
 Couenne::CouenneConstraint, 70
 Couenne::CouenneObjective, 121

body_
 Couenne::CouenneConstraint, 70
 Couenne::CouenneObjective, 121

bonBase
 Couenne::CouenneProblem, 145

bonBase_
 Couenne::CouenneProblem, 151

BonCouenneInterface.hpp
 AmplInterface, 519

Bonmin, 18

boundBranch
 Couenne::CouenneBranchingObject, 52
 Couenne::CouenneOrbitBranchingObj, 123

boundRatio_
 Couenne::CouenneBTPerfIndicator, 55

boundTightening
 Couenne::CouenneProblem, 141

bounds
 Node, 419

bounds_
 Couenne::exprQuad, 356

brSelStrat
 Couenne::CouenneObject, 115

brVar_
 Couenne::CouenneThreeWayBranchObj, 184

branch
 Couenne::CouenneBranchingObject, 51
 Couenne::CouenneComplBranchingObject, 64
 Couenne::CouenneOrbitBranchingObj, 123
 Couenne::CouenneSOSBranchingObject, 174
 Couenne::CouenneThreeWayBranchObj, 184

branch_obj
 Couenne::CouenneObject, 115

branchAndBound
 Couenne::CouenneBab, 49

branchCore
 Couenne::CouenneBranchingObject, 52

branchingMethod_
 Couenne::CouenneDisjCuts, 85

branchtime_
 Couenne::CouenneChooseStrong, 60

btCore
 Couenne::CouenneProblem, 141

c0_
 Couenne::exprGroup, 286

CHANGED
 Couenne::t_chg_bounds, 432

CONCAVE
 Couenne, 30

CONST
 Couenne, 28

CONSTANT
 Couenne, 28

CONV_CONSTANT
 Couenne, 30
CONV_LINEAR
 Couenne, 30
CONV_ZERO
 Couenne, 30
CONVEX
 Couenne, 30
COPY
 Couenne, 28
COU_COSINE
 Couenne, 31
COU_EXPRABS
 Couenne, 30
COU_EXPRCEIL
 Couenne, 30
COU_EXPRCONST
 Couenne, 29
COU_EXPRCOS
 Couenne, 29
COU_EXPRDIV
 Couenne, 29
COU_EXPRESSION
 Couenne, 29
COU_EXPREXP
 Couenne, 30
COU_EXPRFLOOR
 Couenne, 30
COU_EXPRGROUP
 Couenne, 29
COU_EXPRINV
 Couenne, 30
COU_EXPRLBOUND
 Couenne, 29
COU_EXPRLOG
 Couenne, 30
COU_EXPRMAX
 Couenne, 29
COU_EXPRMIN
 Couenne, 29
COU_EXPRMUL
 Couenne, 29
COU_EXPROP
 Couenne, 29
COU_EXPROPP
 Couenne, 30
COU_EXPRPOW
 Couenne, 29
COU_EXPRQUAD
 Couenne, 29
COU_EXPRSIGNPOW
 Couenne, 29
COU_EXPRSIN
 Couenne, 30

COU_EXPRSUB
 Couenne, 29
COU_EXPRSUM
 Couenne, 29
COU_EXPRTRILINEAR
 Couenne, 29
COU_EXPRUBOUND
 Couenne, 29
COU_EXPRUNARY
 Couenne, 29
COU_EXPRVAR
 Couenne, 29
COU_SINE
 Couenne, 31
COUENNE_EQ
 Couenne, 29
COUENNE_FEASIBLE
 Couenne, 27
COUENNE_GE
 Couenne, 29
COUENNE_INFEASIBLE
 Couenne, 27
COUENNE_LE
 Couenne, 29
COUENNE RNG
 Couenne, 29
COUENNE_TIGHTENED
 Couenne, 27
COUNT
 Couenne, 31
CURRENT_ONLY
 Couenne, 29
COU_MAX_COEFF
 CouennePrecisions.hpp, 475
COU_MIN_COEFF
 CouennePrecisions.hpp, 475
COUENNE_BOUND_PREC
 CouennePrecisions.hpp, 475
COUENNE_CROP
 CouenneBranchingObject.hpp, 440
COUENNE_EPS
 CouennePrecisions.hpp, 475
COUENNE_EPS_INT
 CouennePrecisions.hpp, 475
COUENNE_EPS_SIMPL
 CouennePrecisions.hpp, 475
COUENNE_EPS_SYMM
 CouenneProblem.hpp, 529
COUENNE_INFINITY
 CouennePrecisions.hpp, 475
COUENNE_LCROP
 CouenneBranchingObject.hpp, 440
COUENNE_NEAR_BOUND
 CouenneBranchingObject.hpp, 440

COUENNE_VERSION
 config_couenne_default.h, 453

COUENNE_round
 CouennePrecisions.hpp, 475

COUENNE_sign
 CouennePrecisions.hpp, 476

CPUtime_
 Couenne::CouenneFixPoint, 96

call_iter
 Couenne::CouenneProblem, 145

cardInitDom
 Couenne::CouenneRecordBestSol, 159

cardModSol
 Couenne::CouenneRecordBestSol, 160

cardSol
 Couenne::CouenneRecordBestSol, 160

ChangeBounds
 Couenne::CouenneProblem, 137

ChangeStatus
 Couenne::t_chg_bounds, 432

check_lp
 Couenne::CouenneCutGenerator, 78

check_lp_
 Couenne::CouenneCutGenerator, 80

checkAux
 Couenne::CouenneProblem, 147

checkAuxBounds
 Couenne::CouenneProblem, 144

checkAuxBounds_
 Couenne::CouenneProblem, 152

checkBounds
 Couenne::CouenneProblem, 146

checkCons
 Couenne::CouenneProblem, 147

checkCycles
 Couenne::DepGraph, 205

checkDisjSide
 Couenne::CouenneDisjCuts, 84

checkInfeasibility
 Couenne::CouenneComplObject, 67

 Couenne::CouenneObject, 116

 Couenne::CouenneVarObject, 199

checkInt
 Couenne::CouenneProblem, 146

checkNLP
 Couenne::CouenneProblem, 142

checkNLP0
 Couenne::CouenneProblem, 147

checkNLP2
 Couenne::CouenneProblem, 147

checkObj
 Couenne::CouenneProblem, 146

chooseVariable
 Couenne::CouenneChooseStrong, 59

clearPartitions
 Nauty, 414

clone
 Couenne::CouenneAggrProbing, 44

 Couenne::CouenneBranchingObject, 51

 Couenne::CouenneChooseStrong, 58

 Couenne::CouenneChooseVariable, 61

 Couenne::CouenneComplBranchingObject, 64

 Couenne::CouenneComplObject, 67

 Couenne::CouenneConstraint, 69

 Couenne::CouenneCrossConv, 72

 Couenne::CouenneCutGenerator, 76

 Couenne::CouenneDisjCuts, 83

 Couenne::CouenneExprMatrix, 87

 Couenne::CouenneFeasPump, 91

 Couenne::CouenneFixPoint, 95

 Couenne::CouenneInfo, 103

 Couenne::CouenneInterface, 105

 Couenne::CouenneIterativeRounding, 107

 Couenne::CouenneMultiVarProbe, 111

 Couenne::CouenneObject, 116

 Couenne::CouenneObjective, 121

 Couenne::CouenneOrbitBranchingObj, 123

 Couenne::CouenneProblem, 136

 Couenne::CouennePSDcon, 155

 Couenne::CouenneScalar, 162

 Couenne::CouenneSdpCuts, 164

 Couenne::CouenneSetup, 167

 Couenne::CouenneSolverInterface, 170

 Couenne::CouenneSOSBranchingObject, 174

 Couenne::CouenneSOSObject, 176

 Couenne::CouenneSparseMatrix, 180

 Couenne::CouenneSparseVector, 182

 Couenne::CouenneThreeWayBranchObj, 184

 Couenne::CouenneTNLP, 187

 Couenne::CouenneTwoImplied, 193

 Couenne::CouenneVarObject, 199

 Couenne::CouenneVTOBJECT, 202

 Couenne::exprAbs, 219

 Couenne::exprAux, 225

 Couenne::exprCeil, 233

 Couenne::exprClone, 237

 Couenne::exprConst, 240

 Couenne::exprCopy, 246

 Couenne::exprCos, 253

 Couenne::exprDiv, 258

 Couenne::expression, 265

 Couenne::exprExp, 274

 Couenne::exprFloor, 277

 Couenne::exprGroup, 283

 Couenne::ExprHess, 287

 Couenne::exprInv, 291

 Couenne::exprVar, 295

 Couenne::ExprJac, 296

Couenne::exprLBCos, 299
Couenne::exprLBDiv, 301
Couenne::exprLBMul, 303
Couenne::exprLBQuad, 305
Couenne::exprLBSin, 308
Couenne::exprLog, 311
Couenne::exprLowerBound, 315
Couenne::exprMax, 318
Couenne::exprMin, 322
Couenne::exprOddPow, 330
Couenne::exprOpp, 340
Couenne::exprPow, 344
Couenne::exprQuad, 352
Couenne::exprSin, 359
Couenne::exprStore, 363
Couenne::exprSub, 366
Couenne::exprSum, 370
Couenne::exprTrilinear, 373
Couenne::exprUBCos, 376
Couenne::exprUBDiv, 378
Couenne::exprUBMul, 381
Couenne::exprUBQuad, 383
Couenne::exprUBSin, 385
Couenne::exprUpperBound, 394
Couenne::exprVar, 399
Couenne::InitHeuristic, 407
Couenne::NlpSolveHeuristic, 418
Couenne::quadElem, 426
clonearglist
 Couenne::exprOp, 336
closeToBounds
 Couenne, 36
closestFeasible
 Couenne::exprAbs, 220
 Couenne::exprBinProd, 231
 Couenne::exprCeil, 235
 Couenne::exprCopy, 250
 Couenne::exprCos, 255
 Couenne::exprDiv, 259
 Couenne::expression, 271
 Couenne::exprFloor, 279
 Couenne::exprMultiLin, 326
 Couenne::exprPow, 346
 Couenne::exprQuad, 356
 Couenne::exprSin, 360
 Couenne::exprTrilinear, 374
code
 Couenne::exprAbs, 219
 Couenne::exprBinProd, 231
 Couenne::exprCeil, 234
 Couenne::exprConst, 241
 Couenne::exprCopy, 249
 Couenne::exprCos, 254
 Couenne::exprDiv, 259
couenne
 Couenne::expression, 269
 Couenne::exprExp, 274
 Couenne::exprFloor, 278
 Couenne::exprGroup, 285
 Couenne::exprInv, 292
 Couenne::exprLog, 311
 Couenne::exprLowerBound, 316
 Couenne::exprMax, 320
 Couenne::exprMin, 323
 Couenne::exprMultiLin, 325
 Couenne::exprOddPow, 331
 Couenne::exprOp, 336
 Couenne::exprOpp, 341
 Couenne::exprPow, 346
 Couenne::exprQuad, 355
 Couenne::exprSin, 360
 Couenne::exprSub, 367
 Couenne::exprSum, 371
 Couenne::exprTrilinear, 374
 Couenne::exprUnary, 390
 Couenne::exprUpperBound, 395
 Couenne::exprVar, 402
coeff
 Couenne::quadElem, 426
Coin, 18
CoinCopyDisp
 Couenne, 32
CoinInvN
 Couenne, 32
col
 Couenne::CouenneSparseMatrix, 180
col_
 Couenne::CouenneExprMatrix, 88
color
 Couenne::DepNode, 209
color_
 Couenne::DepNode, 209
color_node
 Nauty, 415
color_vertex
 Node, 419
columnNumber
 Couenne::CouenneObject, 118
commonExprs
 Couenne::CouenneProblem, 138
commonexprs_
 Couenne::CouenneProblem, 148
Commutated
 Couenne::CouenneProblem, 139
commuted_
 Couenne::CouenneProblem, 149
compDistInt
 Couenne::CouenneFeasPump, 92
compare

Couenne::CouenneFPSolution, 100
Couenne::CouenneProblem, 137
Couenne::exprCopy, 249
Couenne::expression, 269
Couenne::exprGroup, 285
Couenne::exprOp, 337
Couenne::exprQuad, 355
Couenne::exprUnary, 390
compareAndSave
 Couenne::CouenneRecordBestSol, 159
compareExpr
 Couenne, 33
comparedTerm_
 Couenne, 36
Compute_Symmetry
 Couenne::CouenneProblem, 137
computeAuto
 Nauty, 414
computeBranchingPoint
 Couenne::CouenneVarObject, 199
computeQBound
 Couenne::exprQuad, 356
computeQuadFiniteBound
 Couenne::exprQuad, 356
Con
 Couenne::CouenneProblem, 137
con_sign
 Couenne, 28
config_couenne_default.h
 COUENNE_VERSION, 453
ConstJnlstPtr
 Couenne, 27
constObjVal
 Couenne::CouenneProblem, 145
constObjVal_
 Couenne::CouenneProblem, 152
ConstraintClass
 Couenne::CouenneProblem, 147
ConstraintClass_
 Couenne::CouenneProblem, 153
constraints_
 Couenne::CouenneProblem, 148
Continuous
 Couenne::exprAux, 224
conv_type
 Couenne, 29
ConvType
 Couenne::CouenneCutGenerator, 76
convexity
 Couenne, 30
 Couenne::exprCopy, 249
 Couenne::expression, 269
 Couenne::exprVar, 403
convtype_

Couenne::CouenneCutGenerator, 78
copied_
 Couenne::CouenneFPSolution, 101
 Couenne::DomainPoint, 216
Copy
 Couenne::exprCopy, 249
 Couenne::expression, 271
copy_
 Couenne::exprCopy, 251
cos
 Couenne, 33, 34
cou_trig
 Couenne, 31
CouExpr
 Couenne::CouExpr, 202
CouNumber
 Couenne, 27
Couenne, 18
 AFFINE, 30
 ALL_VARS, 31
 AROUND_CURPOINT, 29
 AUX, 28
 addPowEnvelope, 35
 BRANCH_NONE, 27
 bi_tri, 32
 CONCAVE, 30
 CONST, 28
 CONSTANT, 28
 CONV_CONSTANT, 30
 CONV_LINEAR, 30
 CONV_ZERO, 30
 CONVEX, 30
 COPY, 28
 COU_COSINE, 31
 COU_EXPRABS, 30
 COU_EXPRCEIL, 30
 COU_EXPRCONST, 29
 COU_EXPRCOS, 29
 COU_EXPRDIV, 29
 COU_EXPRESSION, 29
 COU_EXPREXP, 30
 COU_EXPRFLOOR, 30
 COU_EXPRGROUP, 29
 COU_EXPRINV, 30
 COU_EXPRLBOUND, 29
 COU_EXPRLOG, 30
 COU_EXPRMAX, 29
 COU_EXPRMIN, 29
 COU_EXPRMUL, 29
 COU_EXPROP, 29
 COU_EXPROPP, 30
 COU_EXPRPOW, 29
 COU_EXPRQUAD, 29
 COU_EXPRSIGNPOW, 29

COU_EXPRSIN, 30
COU_EXPRSUB, 29
COU_EXPRSUM, 29
COU_EXPRTRILINEAR, 29
COU_EXPRUBOUND, 29
COU_EXPRUNARY, 29
COU_EXPRVAR, 29
COU_SINE, 31
COUENNE_EQ, 29
COUENNE_FEASIBLE, 27
COUENNE_GE, 29
COUENNE_INFEASIBLE, 27
COUENNE_LE, 29
COUENNE RNG, 29
COUENNE_TIGHTENED, 27
COUNT, 31
CURRENT_ONLY, 29
closeToBounds, 36
CoinCopyDisp, 32
CoinInvN, 32
compareExpr, 33
comparedTerm_, 36
con_sign, 28
ConstJnlstPtr, 27
conv_type, 29
convexity, 30
cos, 33, 34
cou_trig, 31
CouNumber, 27
Couenne_large_bound, 36
DECLIN, 30
default_alpha, 36
default_clamp, 36
dig_type, 30
draw_cuts, 32
EAll, 31
EFilterSQP, 31
Elpopt, 31
EMPTY, 28
exp, 33, 34
expr_type, 29
feas_tolerance_default, 36
getOriginal, 33
INCLIN, 30
INSIDE, 28
INTEGER_VARS, 31
inv, 34
inv_dblprime, 34
is_boundbox_regular, 34
isInteger, 33
J_BOUNDTIGHTENING, 35
J_BRANCHING, 35
J_CONVEXIFYING, 35
J_COUENNE, 36
J_DISJCUTS, 36
J_NLPHEURISTIC, 35
J_PROBLEM, 35
J_REFORMULATE, 36
JnlstPtr, 27
LINEAR, 28
large_bound, 36
linearity_type, 28
log, 33, 34
MAX_COU_EXPR_CODE, 30
MON_CONST, 30
MON_UNSET, 30
MON_ZERO, 30
max_pseudocost, 36
maxHeight, 32
maxNlpInf_0, 36
minMaxDelta, 32
modulo, 35
monotonicity, 30
N_ARY, 28
NDECreas, 30
NINCreas, 30
NONCONVEX, 30
NONE, 28
NONLINEAR, 28
NONMONOTONE, 30
nodeType, 28
OBJVAL, 31
ORIG_ONLY, 31
operator<, 32, 35
operator*, 33, 34
operator[^], 33, 34
operator+, 33, 34
operator-, 33, 34
operator/, 33, 34
operator%, 33, 34
opp, 35
oppInvSqr, 34
POST, 28
PRE, 28
pos, 28
powNewton, 35
project, 32
projectSeg, 32
QUADRATIC, 28
rAI, 32
rootQ, 36
STOP_AT_AUX, 31
SUM_INF, 31
SUM_NINF, 31
safe_pow, 35
safeDiv, 34
safeProd, 34
sin, 33, 34

Solver, 31
sparse2dense, 32
TAG_AND_RECURSIVE, 31
THREE_CENTER, 27
THREE_LEFT, 27
THREE_RAND, 27
THREE_RIGHT, 27
TWO_LEFT, 27
TWO_RAND, 27
TWO_RIGHT, 27
treeDecomp, 32
tri_bi, 32
trigImpliedBound, 35
trigNewton, 34
trigSelBranch, 35
TrilinDecompType, 31
UNARY, 28
UNIFORM_GRID, 29
UNSET, 30
unary_function, 27
updateBound, 32
VAR, 28
what_to_compare, 31
ZERO, 28
zero_fun, 33
Couenne::CouenneFeasPump
 FP_CUT_EXTERNAL, 91
 FP_CUT_INTEGRATED, 91
 FP_CUT_NONE, 91
 FP_CUT_POST, 91
 FP_DIST_ALL, 90
 FP_DIST_INT, 90
 FP_DIST_POST, 90
 FP_TABU_CUT, 91
 FP_TABU_NONE, 91
 FP_TABU_PERTURB, 91
 FP_TABU_POOL, 91
Couenne::CouenneObject
 BALANCED, 115
 EXPR_OBJ, 115
 INFEASIBILITY, 115
 INTERVAL_BR, 115
 INTERVAL_BR_REV, 115
 INTERVAL_LP, 115
 INTERVAL_LP_REV, 115
 LP_CENTRAL, 115
 LP_CLAMPED, 115
 MID_INTERVAL, 115
 MIN_AREA, 115
 NO_BRANCH, 115
 NO_STRATEGY, 115
 PROJECTDIST, 115
 VAR_OBJ, 115
 VT_OBJ, 115
Couenne::CouenneProblem
 MulSepNone, 135
 MulSepSimple, 135
 MulSepTight, 135
Couenne::DepNode
 DEP_BLACK, 208
 DEP_GRAY, 208
 DEP_WHITE, 208
Couenne::exprAux
 Continuous, 224
 Integer, 224
 Unset, 224
Couenne::expression
 AUX_EQ, 264
 AUX_GEQ, 264
 AUX_LEQ, 264
 AUX_UNDEF, 264
Couenne::t_chg_bounds
 CHANGED, 432
 EXACT, 432
 UNCHANGED, 432
Couenne::AuxRelation, 37
 findRelations, 37
 generateCuts, 37
Couenne::BiProdDivRel, 38
 findRelations, 39
 generateCuts, 39
Couenne::CouExpr, 202
 CouExpr, 202
 Expression, 203
 operator=, 203
Couenne::CouenneAggrProbing, 41
 ~CouenneAggrProbing, 43
 clone, 44
 couenne_, 44
 CouenneAggrProbing, 43
 generateCuts, 44
 getMaxFailedSteps, 44
 getMaxNodes, 44
 getMaxTime, 44
 getRestoreCutoff, 44
 initCutoff_, 45
 maxFailedSteps_, 45
 maxNodes_, 45
 maxTime_, 45
 numCols_, 45
 probeVariable, 44
 probeVariable2, 44
 registerOptions, 44
 restoreCutoff_, 45
 setMaxFailedSteps, 44
 setMaxNodes, 44
 setMaxTime, 44
 setRestoreCutoff, 44

Couenne::CouenneAmplInterface, 45
~CouenneAmplInterface, 47
CouenneAmplInterface, 47
getCouenneProblem, 47
getTMINLP, 47
registerOptions, 47
setRegisteredOptions, 47
writeSolution, 47
Couenne::CouenneBTPerfIndicator, 53
~CouenneBTPerfIndicator, 55
addToTimer, 55
boundRatio_, 55
CouenneBTPerfIndicator, 55
nFixed_, 55
nProvedInfeas_, 56
nRuns_, 56
name_, 55
oldLB_, 56
oldUB_, 56
operator=, 55
problem_, 56
setOldBounds, 55
shrunkDoubleInf_, 56
shrunkInf_, 55
stats_, 56
totalTime_, 56
update, 55
weightSum_, 56
Couenne::CouenneBab, 47
~CouenneBab, 48
bestBound, 49
bestObj, 49
bestSolution, 49
branchAndBound, 49
CouenneBab, 48
problem_, 49
setProblem, 49
Couenne::CouenneBranchingObject, 49
boundBranch, 52
branch, 51
branchCore, 52
clone, 51
CouenneBranchingObject, 51
cutGen_, 52
doConvCuts_, 53
doFBBT_, 53
downEstimate_, 53
jnlst_, 53
maxDepthOrbBranch, 52
nOrbBr, 52
nSGcomputations, 52
problem_, 52
setSimulate, 52
simulate_, 53
upEstimate_, 53
variable, 52
variable_, 53
Couenne::CouenneChooseStrong, 57
~CouenneChooseStrong, 58
branchtime_, 60
chooseVariable, 59
clone, 58
CouenneChooseStrong, 58
doStrongBranching, 59
estimateProduct_, 59
feasibleSolution, 59
gutsOfSetupList, 59
jnlst_, 60
operator=, 58
problem_, 59
pseudoUpdateLP_, 59
registerOptions, 59
setupList, 58
simulateBranch, 59
Couenne::CouenneChooseVariable, 60
~CouenneChooseVariable, 61
clone, 61
CouenneChooseVariable, 61
feasibleSolution, 62
jnlst_, 62
operator=, 61
problem_, 62
registerOptions, 62
setupList, 62
Couenne::CouenneComplBranchingObject, 62
branch, 64
clone, 64
CouenneComplBranchingObject, 64
sign_, 65
variable2_, 64
Couenne::CouenneComplObject, 65
~CouenneComplObject, 66
checkInfeasibility, 67
clone, 67
CouenneComplObject, 66
createBranch, 67
infeasibility, 67
Couenne::CouenneConstraint, 67
~CouenneConstraint, 69
Body, 70
body_, 70
clone, 69
CouenneConstraint, 69
Lb, 69
lb_, 70
print, 70
standardize, 70
Ub, 69

ub_, 70
Couenne::CouenneCrossConv, 70
~CouenneCrossConv, 72
clone, 72
CouenneCrossConv, 72
generateCuts, 72
jnlist_, 72
problem_, 72
registerOptions, 72
setup, 72
Couenne::CouenneCutGenerator, 73
~CouenneCutGenerator, 76
addEnvelope, 77
addSegment, 77
addTangent, 77
addViolated, 76
addviolated_, 78
BabPtr_, 79
check_lp, 78
check_lp_, 80
clone, 76
ConvType, 76
convtype_, 78
CouenneCutGenerator, 75
createCut, 76, 77
enable_lp_implied_bounds_, 80
enableLpImpliedBounds, 78
firstcall_, 78
genColCuts, 77
genRowCuts, 77
generateCuts, 76
getStats, 77
getnvars, 76
infeasNode, 77
infeasNode_, 79
isFirst, 76
Jnlst, 78
jnlist_, 79
lastPrintLine, 80
nSamples, 76
nSamples_, 78
nlp_, 79
nrootcuts_, 79
ntotalcuts_, 79
objValue_, 79
printLineInfo, 78
Problem, 76
problem_, 79
registerOptions, 78
rootTime, 78
rootTime_, 79
septime_, 79
setBabPtr, 77
setJnlst, 78
setProblem, 76
Couenne::CouenneDisjCuts, 80
~CouenneDisjCuts, 83
activeCols_, 86
activeRows_, 86
addPreviousCut_, 86
applyColCuts, 84
branchingMethod_, 85
checkDisjSide, 84
clone, 83
couenneCG, 83
couenneCG_, 84
CouenneDisjCuts, 83
cpuTime_, 86
depthLevelling_, 85
depthStopSeparate_, 85
generateCuts, 83
generateDisjCuts, 83
getBoxUnion, 84
getDisjunctions, 83
getSingleDisjunction, 84
initDisjNumber_, 85
initDisjPercentage_, 85
isBranchingStrong_, 85
Jnlst, 83
jnlist_, 85
mergeBoxes, 84
minlp_, 85
nrootcuts_, 84
ntotalcuts_, 84
numDisjunctions_, 85
objValue_, 85
OsiCuts2MatrVec, 84
OsiSI2MatrVec, 84
registerOptions, 83
separateWithDisjunction, 83
septime_, 84
Couenne::CouenneExprMatrix, 86
~CouenneExprMatrix, 87
add_element, 88
clone, 87
col_, 88
CouenneExprMatrix, 87
getCols, 88
getRows, 88
operator*, 88
operator=, 87
print, 88
row_, 88
size, 88
varIndices, 88
varIndices_, 88
Couenne::CouenneExprMatrix::compare_pair_ind, 39
operator(), 39

Couenne::CouenneFPpool, 97
 CouenneFPpool, 98
 findClosestAndReplace, 98
 operator=, 98
 Problem, 98
 problem_, 98
 Set, 98
 set_, 98
Couenne::CouenneFPsolution, 98
 ~CouenneFPsolution, 100
 compare, 100
 copied_, 101
 CouenneFPsolution, 100
 maxlInf_, 101
 maxNLInf_, 101
 n, 100
 n_, 101
 nlInf_, 101
 nNLInf_, 101
 objVal_, 101
 operator=, 100
 problem_, 101
 x, 100
 x_, 100
Couenne::CouenneFeasPump, 89
 ~CouenneFeasPump, 91
 clone, 91
 compDistInt, 92
 CouenneFeasPump, 91
 findSolution, 92
 fixIntVariables, 92
 fpCompDistIntType, 90
 fpCutPlane, 90
 fpTabuMgtPolicy, 91
 init_MILP, 92
 initlpoptApp, 92
 milpPhase, 93
 multDistMILP, 93
 multDistNLP, 93
 multHessMILP, 93
 multHessNLP, 93
 multObjFMILP, 93
 multObjFNLP, 93
 nCalls, 93
 nlp, 93
 nlpPhase, 93
 operator=, 91
 Problem, 92
 registerOptions, 92
 resetModel, 91
 setNumberSolvePerLevel, 92
 solution, 91
 solveMILP, 92
 solveNLP, 92
 updateNLPObj, 92
Couenne::CouenneFixPoint, 94
 ~CouenneFixPoint, 95
 CPUtime_, 96
 clone, 95
 CouenneFixPoint, 95
 createRow, 96
 extendedModel_, 96
 firstCall_, 96
 generateCuts, 95
 nTightened_, 96
 perfIndicator_, 96
 problem_, 96
 registerOptions, 96
Couenne::CouenneInfo, 101
 ~CouenneInfo, 103
 addSolution, 103
 clone, 103
 CouenneInfo, 103
 nlpSols_, 104
 NlpSolutions, 103
Couenne::CouenneInfo::NlpSolution, 415
 ~NlpSolution, 416
 nVars, 416
 NlpSolution, 416
 objVal, 416
 solution, 416
Couenne::CouenneInterface, 104
 ~CouenneInterface, 105
 appName, 105
 clone, 105
 CouenneInterface, 104
 extractLinearRelaxation, 105
 have_nlp_solution_, 105
 haveNlpSolution, 105
 setAppDefaultOptions, 105
Couenne::CouenneIterativeRounding, 105
 ~CouenneIterativeRounding, 107
 clone, 107
 CouenneIterativeRounding, 107
 operator=, 107
 registerOptions, 108
 resetModel, 107
 setAggressiveness, 108
 setBaseLbRhs, 108
 setCouenneProblem, 107
 setMaxFirPoints, 108
 setMaxRoundingIter, 108
 setMaxTime, 108
 setMaxTimeFirstCall, 108
 setNlp, 107
 setOmega, 108
 solution, 107
Couenne::CouenneMINLPInterface, 109

options, 109
problem, 109
setInitSol, 109
setObj, 109
solve, 109
Couenne::CouenneMultiVarProbe, 110
 ~CouenneMultiVarProbe, 111
 clone, 111
 couenne_, 111
 CouenneMultiVarProbe, 111
 generateCuts, 111
 maxTime_, 111
 numCols_, 111
Couenne::CouenneOSInterface, 124
 ~CouenneOSInterface, 125
 CouenneOSInterface, 125
 getCouenneProblem, 126
 getTMINLP, 126
 registerOptions, 126
 writeSolution, 126
Couenne::CouenneObject, 112
 ~CouenneObject, 116
 alpha_, 118
 brSelStrat, 115
 branch_obj, 115
 checkInfeasibility, 116
 clone, 116
 columnNumber, 118
 CouenneObject, 115, 116
 createBranch, 116
 cutGen_, 118
 doConvCuts_, 119
 doFBBT_, 119
 downEstimate, 117
 downEstimate_, 119
 feas_tolerance_, 118
 feasibleRegion, 116
 getBrPoint, 117
 infeasibility, 116
 intInfeasibility, 117
 isCuttable, 117
 jnlst_, 118
 lp_clamp, 118
 lp_clamp_, 118
 midInterval, 117
 problem_, 118
 pseudoMultType_, 119
 pseudocostMult, 115
 Reference, 117
 reference_, 118
 setEstimate, 117
 setEstimates, 117
 setParameters, 116
 Strategy, 117
 strategy_, 118
 upEstimate, 117
 upEstimate_, 119
Couenne::CouenneObjective, 119
 ~CouenneObjective, 121
 Body, 121
 body_, 121
 clone, 121
 CouenneObjective, 121
 print, 121
 standardize, 121
Couenne::CouenneOrbitBranchingObj, 122
 boundBranch, 123
 branch, 123
 clone, 123
 CouenneOrbitBranchingObj, 123
 setSimulate, 124
Couenne::CouennePSDcon, 153
 ~CouennePSDcon, 155
 clone, 155
 CouennePSDcon, 155
 getX, 155
 operator=, 155
 print, 155
 standardize, 155
 X_, 155
Couenne::CouenneProblem, 126
 ~CouenneProblem, 135
 addAuxiliary, 139
 addEQConstraint, 139
 addGEConstraint, 139
 addLEConstraint, 139
 addObjective, 139
 addRNGConstraint, 139
 addVariable, 139
 aggressiveBT, 141
 analyzeSparsity, 145
 asl_, 151
 AuxSet, 138
 auxSet_, 148
 auxiliarize, 142
 bestObj, 139
 bestObj_, 149
 bestSol, 139
 bonBase, 145
 bonBase_, 151
 boundTightening, 141
 btCore, 141
 call_iter, 145
 ChangeBounds, 137
 checkAux, 147
 checkAuxBounds, 144
 checkAuxBounds_, 152
 checkBounds, 146

checkCons, 147
checkInt, 146
checkNLP, 142
checkNLP0, 147
checkNLP2, 147
checkObj, 146
clone, 136
commonExprs, 138
commonexprs_, 148
Commuted, 139
commuted_, 149
compare, 137
Compute_Symmetry, 137
Con, 137
constObjVal, 145
constObjVal_, 152
ConstraintClass, 147
ConstraintClass_, 153
constraints_, 148
CouenneProblem, 135
createUnusedOriginals, 144
created_pcutoff_, 150
curnvars_, 148
decomposeTerm, 143
Dependence, 143
dependence_, 151
doABT, 140
doABT_, 150
doFBBT, 140
doFBBT_, 150
doOBBT, 140
doOBBT_, 150
doPrint_, 148
doRCBT, 140
doRCBT_, 150
domain, 138
domain_, 148
evalOrder, 137
evalVector, 137
exprMul, 147
fake_tighten, 145
fbbtReachedIterLimit, 144
fbbtReachedIterLimit_, 152
feas_tolerance_, 151
fillDependence, 146
fillIntegerRank, 146
fillObjCoeff, 142
fillQuadIndices, 142
Find_Orbit, 137
findSOS, 143
flattenMul, 146
getAuxs, 141
getCutOff, 142
getCutOffSol, 142
getDepGraph, 138
getFeasTol, 146
getIntegerCandidate, 142
getLastPrioSort, 146
getMaxCpuTime, 144
getNtyInfo, 137
getRecordBestSol, 146
getSdpCutGen, 145
getTrilinDecompType, 145
graph_, 149
impliedBounds, 141
indcoe2vector, 143
index_sort, 153
initAuxs, 141
initOptions, 136
installCutOff, 142
integerRank_, 151
Jnlst, 142
jnlst_, 150
lastPrioSort_, 152
Lb, 138
linStandardize, 142
logAbtLev, 140
logAbtLev_, 150
logObbtLev, 140
logObbtLev_, 150
max_fbbt_iter_, 152
maxCpuTime_, 151
minDepthPrint_, 147
minNodePrint_, 148
multiSep, 135
MultilinSep, 144
multilinSep_, 152
nCons, 136
nDefVars, 136
nIntVars, 136
nIntVars_, 148
nObjs, 136
nOrigCons, 136
nOrigCons_, 149
nOrigIntVars, 136
nOrigIntVars_, 149
nOrigVars, 136
nOrigVars_, 149
nUnusedOriginals, 144
nUnusedOriginals_, 152
nVars, 136
nauty_info, 153
undefined_, 149
node_info, 153
node_sort, 153
numberInRank_, 151
numbering_, 149
obbt, 141

obbt_iter, 145
obbtInner, 145
Obj, 137
objectives_, 148
Objects, 143
objects_, 151
opt_window_, 150
optimum_, 149
orbitalBranching, 144
orbitalBranching_, 152
pcutoff_, 149
perfIndicator_, 152
print, 140
Print_Orbits, 137
problemName, 143
problemName_, 148
readOptimum, 142
realign, 146
recBSol, 152
redCostBT, 141
reformulate, 139
registerOptions, 142
resetCutOff, 142
restoreUnusedOriginals, 144
sdpCutGen_, 153
setBase, 144
setCheckAuxBounds, 144
setCutOff, 142
setLastPrioSort, 146
setMaxCpuTime, 143
setNDefVars, 137
setObjective, 139
setProblemName, 143
setupSymmetry, 137
splitAux, 143
standardize, 140
sym_setup, 137
testIntFix, 146
tightenBounds, 141
trilinDecompType_, 152
Ub, 138
unusedOriginalsIndices, 144
unusedOriginalsIndices_, 151
useQuadratic_, 150
Var, 137
Variables, 137
variables_, 148
writeAMPL, 140
writeGAMS, 141
writeLP, 141
X, 138
Couenne::CouenneRecordBestSol, 156
~CouenneRecordBestSol, 158
cardInitDom, 159
cardModSol, 160
cardSol, 160
compareAndSave, 159
CouenneRecordBestSol, 158
getCardInitDom, 158
getCardModSol, 159
getCardSol, 158
getHasSol, 158
getInitDomLb, 158
getInitDomUb, 158
getInitIsInt, 158
getListInt, 158
getMaxViol, 159
getModSol, 159
getModSolMaxViol, 159
getModSolVal, 159
getSol, 158
getVal, 159
hasSol, 160
initDomLb, 159
initDomUb, 160
initIsInt, 159
listInt, 159
maxViol, 160
modSol, 160
modSolMaxViol, 160
modSolVal, 160
printSol, 159
setCardSol, 158
setHasSol, 158
setInitDomLb, 158
setInitDomUb, 158
setInitIsInt, 158
setModSol, 159
setSol, 158
setVal, 159
sol, 160
update, 159
val, 160
Couenne::CouenneSOSBranchingObject, 172
branch, 174
clone, 174
CouenneSOSBranchingObject, 174
doConvCuts_, 174
doFBBT_, 174
jnls_, 174
problem_, 174
reference_, 174
Couenne::CouenneSOSObject, 175
clone, 176
CouenneSOSObject, 176
createBranch, 176
doConvCuts_, 177
doFBBT_, 177

jnlst_, 176
problem_, 176
reference_, 176
Couenne::CouenneScalar, 160
~CouenneScalar, 161
clone, 162
CouenneScalar, 161, 162
elem_, 162
getElem, 162
getIndex, 162
index_, 162
operator<, 162
operator=, 162
print, 162
Couenne::CouenneSdpCuts, 163
~CouenneSdpCuts, 164
clone, 164
CouenneSdpCuts, 164
doNotUse, 164
doNotUse_, 165
fillMissingTerms_, 165
generateCuts, 164
minors_, 165
numEigVec_, 165
onlyNegEV_, 165
operator=, 164
problem_, 165
registerOptions, 165
updateSol, 165
useSparsity_, 165
Couenne::CouenneSetup, 166
~CouenneSetup, 167
addMilpCutGenerators, 167
clone, 167
couennePtr, 167
CouenneSetup, 166
displayStats, 167
getDoubleParameter, 167
InitializeCouenne, 167
readOptionsFile, 167
registerAllOptions, 167
registerOptions, 167
setDoubleParameter, 167
setNodeComparisonMethod, 168
Couenne::CouenneSolverInterface
~CouenneSolverInterface, 170
clone, 170
CouenneSolverInterface, 170
CutGen, 170
cutgen_, 172
getObjValue, 171
initialSolve, 171
isProvenDualInfeasible, 171
isProvenOptimal, 170
isProvenPrimalInfeasible, 170
knowDualInfeasible_, 172
knowInfeasible_, 172
knowOptimal_, 172
markHotStart, 171
resolve, 171
resolve_nobt, 171
setCutGenPtr, 170
solveFromHotStart, 171
tightenBounds, 171
tightenBoundsCLP, 171
tightenBoundsCLP_Light, 171
unmarkHotStart, 171
Couenne::CouenneSolverInterface< T >, 168
Couenne::CouenneSparseBndVec
~CouenneSparseBndVec, 178
CouenneSparseBndVec, 178
data, 178
indices, 178
nElements, 178
reset, 178
resize, 178
Couenne::CouenneSparseBndVec< T >, 177
Couenne::CouenneSparseMatrix, 179
~CouenneSparseMatrix, 180
clone, 180
col, 180
CouenneSparseMatrix, 179
num, 180
operator=, 180
row, 180
val, 180
Couenne::CouenneSparseVector, 180
~CouenneSparseVector, 182
add_element, 182
clone, 182
CouenneSparseVector, 182
elem_, 183
getElements, 182
multiply_thres, 182
operator*, 182
operator=, 182
print, 182
Couenne::CouenneSparseVector::compare_scalars, 39
operator(), 40
Couenne::CouenneTNLP, 185
~CouenneTNLP, 187
clone, 187
CouenneTNLP, 187
eval_f, 188
eval_g, 188
eval_grad_f, 188
eval_h, 189
eval_jac_g, 188

finalize_solution, 189
get_bounds_info, 188
get_constraints_linearity, 188
get_list_of_nonlinear_variables, 189
get_nlp_info, 188
get_number_of_nonlinear_variables, 189
get_starting_point, 188
get_variables_linearity, 188
getSaveOptHessian, 189
getSolValue, 187
getSolution, 187
intermediate_callback, 189
operator=, 187
optHessian, 189
setInitSol, 187
setObjective, 189
Couenne::CouenneThreeWayBranchObj, 183
brVar_, 184
branch, 184
clone, 184
CouenneThreeWayBranchObj, 184
firstBranch_, 185
jnlst_, 185
lcrop_, 184
rcrop_, 185
Couenne::CouenneTwoImplied, 190
~CouenneTwoImplied, 193
clone, 193
CouenneTwoImplied, 193
depthLevelling_, 194
depthStopSeparate_, 194
firstCall_, 194
generateCuts, 193
jnlst_, 193
nMaxTrials_, 193
problem_, 193
registerOptions, 193
totalInitTime_, 194
totalTime_, 193
Couenne::CouenneUserInterface, 194
~CouenneUserInterface, 195
addBabPlugins, 196
CouenneUserInterface, 195
getCouenneProblem, 195
getTMINLP, 195
jnlst, 196
options, 196
setupJournals, 195
writeSolution, 196
Couenne::CouenneVTOBJECT, 200
~CouenneVTOBJECT, 202
clone, 202
CouenneVTOBJECT, 201
infeasibility, 202
Couenne::CouenneVarObject, 196
~CouenneVarObject, 198
checkInfeasibility, 199
clone, 199
computeBranchingPoint, 199
CouenneVarObject, 198
createBranch, 199
feasibleRegion, 199
infeasibility, 199
isCuttable, 199
varSelection_, 199
Couenne::DepGraph, 203
~DepGraph, 204
checkCycles, 205
Counter, 205
counter_, 206
createOrder, 205
DepGraph, 204
depends, 205
erase, 205
insert, 205
lookup, 205
print, 205
replaceIndex, 205
Vertices, 205
vertices_, 206
Couenne::DepNode, 206
~DepNode, 208
color, 209
color_, 209
createOrder, 208
dep_color, 208
DepList, 208
depList, 209
depList_, 209
DepNode, 208
depends, 208
Index, 208
index_, 209
Order, 208
order_, 209
print, 208
replaceIndex, 209
Couenne::Domain, 209
~Domain, 211
current, 212
domStack_, 213
Domain, 211
lb, 212
point_, 212
pop, 212
push, 211, 212
ub, 212
x, 212

Couenne::DomainPoint, 213
 ~DomainPoint, 214
 copied_, 216
 Dimension, 215
 dimension_, 216
 Domain, 215
 DomainPoint, 214
 isNlp, 215
 isNlp_, 216
 lb, 215
 lb_, 216
 operator=, 215
 resize, 214
 size, 214
 ub, 215
 ub_, 216
 x, 215
 x_, 216
Couenne::ExprHess, 286
 ~ExprHess, 287
 clone, 287
 expr, 287
 ExprHess, 287
 iRow, 287
 jCol, 287
 laml, 287
 nnz, 287
 numL, 287
 operator=, 287
Couenne::ExprJac, 296
 ~ExprJac, 296
 clone, 296
 expr, 297
 ExprJac, 296
 iRow, 296
 jCol, 296
 nRows, 297
 nnz, 296
 operator=, 296
Couenne::GlobalCutOff, 405
 ~GlobalCutOff, 405
 getCutOff, 405
 getCutOffSol, 406
 GlobalCutOff, 405
 setCutOff, 405
Couenne::InitHeuristic, 406
 ~InitHeuristic, 406
 clone, 407
 InitHeuristic, 406
 operator=, 407
 resetModel, 407
 solution, 407
Couenne::LinMap, 410
 insert, 411
 Map, 411
 Couenne::MultiProdRel, 411
 findRelations, 412
 generateCuts, 412
 Couenne::NlpSolveHeuristic, 416
 ~NlpSolveHeuristic, 417
 clone, 418
 NlpSolveHeuristic, 417
 operator=, 418
 registerOptions, 418
 resetModel, 418
 setCouenneProblem, 418
 setMaxNlpIter, 418
 setNlp, 418
 setNumberSolvePerLevel, 418
 solution, 418
 Couenne::PowRel, 422
 findRelations, 423
 generateCuts, 423
 Couenne::Qroot, 423
 ~Qroot, 425
 operator(), 425
 Qmap, 425
 Qroot, 425
 Couenne::QuadMap, 426
 insert, 427
 Map, 426
 Couenne::SmartAsl, 429
 ~SmartAsl, 430
 asl, 430
 SmartAsl, 430
 Couenne::SumLogAuxRel, 430
 findRelations, 431
 generateCuts, 431
 Couenne::compExpr, 40
 operator(), 41
 Couenne::compNode, 41
 operator(), 41
 Couenne::compareSol, 40
 operator(), 40
 Couenne::exprAbs, 216
 clone, 219
 closestFeasible, 220
 code, 219
 differentiate, 219
 exprAbs, 218
 F, 219
 generateCuts, 219
 getBounds, 219
 gradientNorm, 219
 impliedBound, 220
 isCuttable, 220
 isInteger, 220
 printOp, 219

selectBranch, 220
Couenne::exprAux, 220
 ~exprAux, 225
 clone, 225
 crossBounds, 226
 decreaseMult, 227
 DepList, 226
 exprAux, 225
 generateCuts, 226
 Image, 225, 226
 image_, 228
 increaseMult, 227
 intType, 224
 integer_, 228
 isDefinedInteger, 227
 isInteger, 227
 Lb, 225
 lb_, 228
 Linearity, 226
 linkDomain, 227
 Multiplicity, 227
 multiplicity_, 228
 operator(), 226
 print, 225
 properObject, 228
 rank, 226
 rank_, 228
 setInteger, 227
 sign, 228
 sign_, 229
 simplify, 226
 top_level, 227
 top_level_, 228
 Type, 225
 Ub, 225
 ub_, 228
 zeroMult, 227
Couenne::exprBinProd, 229
 balancedMul, 231
 closestFeasible, 231
 code, 231
 differentiate, 230
 exprBinProd, 230
 generateCuts, 231
 getBounds, 230
 gradientNorm, 230
 impliedBound, 231
 isCuttable, 231
 Linearity, 230
 selectBranch, 231
 simplify, 230
 standardize, 231
Couenne::exprCeil, 231
 clone, 233
 closestFeasible, 235
 code, 234
 differentiate, 234
 exprCeil, 233
 F, 234
 generateCuts, 234
 getBounds, 234
 gradientNorm, 234
 impliedBound, 234
 isCuttable, 235
 printOp, 234
 selectBranch, 235
Couenne::exprClone, 235
 ~exprClone, 237
 clone, 237
 exprClone, 237
 operator(), 238
 print, 237
 Value, 237
Couenne::exprConst, 238
 clone, 240
 code, 241
 dependsOn, 241
 differentiate, 241
 exprConst, 240
 generateCuts, 241
 getBounds, 241
 isInteger, 242
 Linearity, 241
 operator(), 241
 print, 240
 rank, 242
 Type, 240
 Value, 240
Couenne::exprCopy, 242
 ~exprCopy, 245
 ArgList, 247
 ArgPtr, 247
 Argument, 247
 clone, 246
 closestFeasible, 250
 code, 249
 compare, 249
 convexity, 249
 Copy, 246
 copy_, 251
 DepList, 248
 differentiate, 248
 exprCopy, 245
 fillDepSet, 250
 generateCuts, 249
 getBounds, 248, 249
 gradientNorm, 248
 Image, 246

impliedBound, 249
Index, 246
inverse, 250
isBijective, 250
isCuttable, 251
isDefinedInteger, 248
isInteger, 248
isaCopy, 246
Linearity, 248
Multiplicity, 250
nArgs, 246
operator(), 247
Original, 246
print, 247
rank, 249
realign, 250
replace, 250
selectBranch, 250
simplify, 248
standardize, 249
Type, 246
Value, 247
value_, 251
Couenne::exprCos, 251
clone, 253
closestFeasible, 255
code, 254
differentiate, 254
exprCos, 253
F, 254
generateCuts, 254
getBounds, 254
gradientNorm, 254
impliedBound, 254
isCuttable, 255
printOp, 254
selectBranch, 255
Couenne::exprDiv, 255
clone, 258
closestFeasible, 259
code, 259
differentiate, 258
exprDiv, 258
generateCuts, 259
getBounds, 259
gradientNorm, 258
impliedBound, 259
isCuttable, 260
isInteger, 259
Linearity, 258
operator(), 258
printOp, 258
selectBranch, 259
simplify, 258
standardize, 259
Couenne::exprExp, 272
clone, 274
code, 274
differentiate, 274
exprExp, 273
F, 274
generateCuts, 274
getBounds, 274
gradientNorm, 274
impliedBound, 275
inverse, 275
isBijective, 275
isCuttable, 275
printOp, 274
selectBranch, 275
Couenne::exprFloor, 275
clone, 277
closestFeasible, 279
code, 278
differentiate, 278
exprFloor, 277
F, 278
generateCuts, 278
getBounds, 278
gradientNorm, 278
impliedBound, 278
isCuttable, 279
printOp, 278
selectBranch, 279
Couenne::exprGroup, 279
~exprGroup, 283
c0_, 286
clone, 283
code, 285
compare, 285
DepList, 284
differentiate, 284
exprGroup, 283
fillDepSet, 285
genExprGroup, 283
generateCuts, 285
getBounds, 284, 285
getc0, 283
gradientNorm, 284
isInteger, 285
lcoeff, 283
lcoeff_, 286
lincoeff, 283
Linearity, 284
operator(), 284
print, 283
rank, 285
realign, 286

replace, 285
simplify, 284
`Couenne::exprVar`, 292
clone, 295
`exprVar`, 295
`isDefinedInteger`, 295
`isInteger`, 295
print, 295
`Couenne::exprIf`, 287
`Couenne::exprInv`, 289
clone, 291
code, 292
differentiate, 291
`exprInv`, 291
`F`, 291
generateCuts, 292
getBounds, 291
gradientNorm, 291
impliedBound, 292
inverse, 292
isBijective, 292
isCuttable, 292
Linearity, 291
print, 291
selectBranch, 292
`Couenne::exprLBCos`, 297
clone, 299
`exprLBCos`, 298
operator(), 299
printOp, 299
printPos, 299
`Couenne::exprLBDiv`, 299
clone, 301
`exprLBDiv`, 301
operator(), 301
printOp, 301
printPos, 301
`Couenne::exprLBMul`, 302
clone, 303
`exprLBMul`, 303
operator(), 303
printOp, 303
printPos, 303
`Couenne::exprLBQuad`, 304
`~exprLBQuad`, 305
clone, 305
`exprLBQuad`, 305
operator(), 305
print, 306
`Couenne::exprLBSin`, 306
clone, 308
`exprLBSin`, 307
operator(), 308
printOp, 308
printPos, 308
printPos, 308
`Couenne::exprLog`, 308
clone, 311
code, 311
differentiate, 311
`exprLog`, 310
`F`, 311
generateCuts, 311
getBounds, 311
gradientNorm, 311
impliedBound, 312
inverse, 312
isBijective, 312
isCuttable, 312
printOp, 311
selectBranch, 312
`Couenne::exprLowerBound`, 312
clone, 315
code, 316
dependsOn, 316
differentiate, 316
`exprLowerBound`, 315
Linearity, 316
operator(), 316
print, 315
Type, 315
`Couenne::exprMax`, 316
clone, 318
code, 320
differentiate, 319
`exprMax`, 318
generateCuts, 320
getBounds, 319
Linearity, 319
operator(), 319
printOp, 319
printPos, 319
simplify, 319
standardize, 319
`Couenne::exprMin`, 320
clone, 322
code, 323
differentiate, 323
`exprMin`, 322
generateCuts, 323
getBounds, 323
Linearity, 323
operator(), 322
printOp, 322
printPos, 322
simplify, 323
standardize, 323
`Couenne::exprMultiLin`, 324
`balancedMul`, 326

closestFeasible, 326
code, 325
differentiate, 325
exprMultiLin, 325
generateCuts, 325
getBounds, 325
gradientNorm, 325
impliedBound, 326
impliedBoundMul, 326
isCuttable, 326
Linearity, 325
selectBranch, 326
simplify, 325
standardize, 325
Couenne::exprNorm, 326
Couenne::exprOddPow, 328
clone, 330
code, 331
exprOddPow, 330
generateCuts, 331
getBounds, 331
getFixVar, 331
impliedBound, 331
isCuttable, 331
operator(), 330
printOp, 330
selectBranch, 331
standardize, 331
Couenne::exprOp, 332
~exprOp, 334
ArgList, 335
arglist_, 337
clonearglist, 336
code, 336
compare, 337
DepList, 336
exprOp, 334, 335
fillDepSet, 337
isInteger, 337
Linearity, 336
nArgs, 335
nargs_, 337
print, 335
printOp, 335
printPos, 335
rank, 337
realign, 337
replace, 337
shrink_arglist, 336
simplify, 336
standardize, 336
Type, 335
Couenne::exprOpp, 338
clone, 340
code, 341
differentiate, 340
exprOpp, 340
F, 340
generateCuts, 341
getBounds, 341
gradientNorm, 340
impliedBound, 341
isInteger, 341
Linearity, 341
print, 340
simplify, 341
standardize, 341
Couenne::exprPWLinear, 347
Couenne::exprPow, 342
clone, 344
closestFeasible, 346
code, 346
differentiate, 345
exprPow, 344
generateCuts, 346
getBounds, 345
getFixVar, 346
gradientNorm, 345
impliedBound, 346
isCuttable, 347
isInteger, 345
isSignpower, 347
Linearity, 345
operator(), 345
printOp, 345
selectBranch, 346
simplify, 345
standardize, 346
Couenne::exprQuad, 348
alphaConvexify, 353
bounds_, 356
clone, 352
closestFeasible, 356
code, 355
compare, 355
computeQBound, 356
computeQuadFiniteBound, 356
DepList, 355
differentiate, 353
eigen_, 356
exprQuad, 352
fillDepSet, 355
generateCuts, 353
getBounds, 353
getQ, 352
getnQTerms, 352
gradientNorm, 353
impliedBound, 356

isCuttable, 356
isInteger, 355
Linearity, 353
matrix_, 356
nqterms_, 357
operator(), 352
print, 352
quadCuts, 353
rank, 355
realign, 356
replace, 355
selectBranch, 355
simplify, 353
sparseQ, 352
sparseQcol, 352
Couenne::exprSin, 357
clone, 359
closestFeasible, 360
code, 360
differentiate, 359
exprSin, 359
F, 359
generateCuts, 360
getBounds, 360
gradientNorm, 359
impliedBound, 360
isCuttable, 360
printOp, 359
selectBranch, 360
Couenne::exprStore, 361
~exprStore, 363
clone, 363
exprStore, 363
operator(), 363
print, 363
value_, 363
Couenne::exprSub, 364
clone, 366
code, 367
differentiate, 366
exprSub, 365
generateCuts, 367
getBounds, 366
impliedBound, 367
Linearity, 366
operator(), 366
printOp, 366
simplify, 366
standardize, 366
Couenne::exprSum, 367
~exprSum, 370
clone, 370
code, 371
createQuadratic, 372
differentiate, 370
exprSum, 370
generateCuts, 371
getBounds, 371
impliedBound, 371
impliedBoundSum, 372
Linearity, 371
operator(), 370
printOp, 370
simplify, 371
standardize, 371
Couenne::exprTrilinear, 372
clone, 373
closestFeasible, 374
code, 374
exprTrilinear, 373
generateCuts, 373
getBounds, 373
gradientNorm, 373
impliedBound, 374
selectBranch, 374
Couenne::exprUBCos, 374
clone, 376
exprUBCos, 376
operator(), 376
printOp, 376
printPos, 376
Couenne::exprUBDiv, 377
clone, 378
exprUBDiv, 378
operator(), 378
printOp, 378
printPos, 378
Couenne::exprUBMul, 379
clone, 381
exprUBMul, 380
operator(), 381
printOp, 381
printPos, 381
Couenne::exprUBQuad, 381
~exprUBQuad, 383
clone, 383
exprUBQuad, 383
operator(), 383
print, 383
Couenne::exprUBSin, 384
clone, 385
exprUBSin, 385
operator(), 385
printOp, 385
printPos, 385
Couenne::exprUnary, 386
~exprUnary, 388
ArgPtr, 389

Argument, 389
argument_, 391
code, 390
compare, 390
DepList, 390
exprUnary, 388
F, 389
fillDepSet, 391
isInteger, 390
Linearity, 390
nArgs, 389
operator(), 389
print, 389
printOp, 389
printPos, 389
rank, 390
realign, 391
replace, 391
simplify, 390
standardize, 390
Type, 388
Couenne::exprUpperBound, 391
clone, 394
code, 395
dependsOn, 395
differentiate, 395
exprUpperBound, 394
Linearity, 395
operator(), 394
print, 394
Type, 394
Couenne::exprVar, 395
~exprVar, 398
clone, 399
code, 402
convexity, 403
crossBounds, 400
decreaseMult, 402
DepList, 400
differentiate, 400
domain, 402
domain_, 403
exprVar, 398, 399
fillDepSet, 402
generateCuts, 401
getBounds, 401
gradientNorm, 400
impliedBound, 402
Index, 399
isDefinedInteger, 401
isFixed, 402
isInteger, 401
Lb, 399
lb, 399
Linearity, 401
linkDomain, 402
operator(), 400
print, 400
properObject, 403
rank, 402
setInteger, 403
sign, 403
simplify, 401
Type, 399
Ub, 399
ub, 399
varIndex_, 403
zeroMult, 403
Couenne::expression, 260
~expression, 264
ArgList, 265
ArgPtr, 265
Argument, 265
auxSign, 264
clone, 265
closestFeasible, 271
code, 269
compare, 269
convexity, 269
Copy, 271
DepList, 267
dependsOn, 267
differentiate, 267
expression, 264
fillDepSet, 270
generateCuts, 269
getBounds, 268
gradientNorm, 267
Image, 266
impliedBound, 270
Index, 265
inverse, 271
isBijective, 271
isCuttable, 271
isDefinedInteger, 268
isInteger, 268
isaCopy, 271
Linearity, 268
linkDomain, 270
Multiplicity, 270
nArgs, 265
operator(), 267
Original, 266
print, 266
rank, 269
realign, 271
replace, 270
selectBranch, 270

simplify, 267
standardize, 268
Type, 266
Value, 266
Couenne::funtriplet, 404
 ~funtriplet, 404
 F, 405
 Fp, 405
 FpInv, 405
 Fpp, 405
 funtriplet, 404
Couenne::kpowertriplet, 407
 ~kpowertriplet, 409
 F, 409
 Fp, 409
 FpInv, 409
 Fpp, 409
 kpowertriplet, 409
 mult_, 409
Couenne::powertriplet, 420
 ~powertriplet, 421
 exponent_, 422
 F, 421
 Fp, 421
 FpInv, 422
 Fpp, 421
 issignpower_, 422
 powertriplet, 421
Couenne::quadElem, 425
 clone, 426
 coeff, 426
 quadElem, 426
 varl, 426
 varJ, 426
Couenne::simpletriplet, 427
 ~simpletriplet, 428
 F, 428
 f_, 429
 Fp, 428
 fp_, 429
 fpl_, 429
 FpInv, 429
 Fpp, 428
 fpp_, 429
 simpletriplet, 428
Couenne::t_chg_bounds, 431
 ChangeStatus, 432
 lower, 432
 operator=, 433
 setLower, 433
 setLowerBits, 433
 setUpper, 433
 setUpperBits, 433
 t_chg_bounds, 432
 upper, 432
couenne_
 Couenne::CouenneAggrProbing, 44
 Couenne::CouenneMultiVarProbe, 111
couenne_large_bound
 Couenne, 36
CouenneAggrProbing
 Couenne::CouenneAggrProbing, 43
CouenneAmplInterface
 Couenne::CouenneAmplInterface, 47
CouenneBTPerIndicator
 Couenne::CouenneBTPerIndicator, 55
CouenneBab
 Couenne::CouenneBab, 48
CouenneBranchingObject
 Couenne::CouenneBranchingObject, 51
CouenneBranchingObject.hpp
 COUENNE_CROP, 440
 COUENNE_LCROP, 440
couenneCG
 Couenne::CouenneDisjCuts, 83
couenneCG_
 Couenne::CouenneDisjCuts, 84
CouenneChooseStrong
 Couenne::CouenneChooseStrong, 58
CouenneChooseVariable
 Couenne::CouenneChooseVariable, 61
CouenneComplBranchingObject
 Couenne::CouenneComplBranchingObject, 64
CouenneComplObject
 Couenne::CouenneComplObject, 66
CouenneConstraint
 Couenne::CouenneConstraint, 69
CouenneCrossConv
 Couenne::CouenneCrossConv, 72
CouenneCutGenerator
 Couenne::CouenneCutGenerator, 75
CouenneDisjCuts
 Couenne::CouenneDisjCuts, 83
CouenneExprBCos.hpp
 M_PI, 480
CouenneExprBMul.hpp
 MUL_INF, 482
 MUL_ZERO, 482
CouenneExprBSin.hpp
 M_PI, 484
CouenneExprDiv.hpp
 BR_MULT, 488
 BR_NEXT_ZERO, 488
 SAFE_COEFFICIENT, 489
CouenneExprMatrix
 Couenne::CouenneExprMatrix, 87
CouenneExprOp.hpp
 MAX_ARG_LINE, 470

CouenneFPpool
 Couenne::CouenneFPpool, 98

CouenneFPSolution
 Couenne::CouenneFPSolution, 100

CouenneFeasPump
 Couenne::CouenneFeasPump, 91

CouenneFeasPump.hpp
 fadingCoeff, 516

CouenneFixPoint
 Couenne::CouenneFixPoint, 95

CouenneInfeasCut.hpp
 isWiped, 436
 WipeMakeInfeas, 436

CouenneInfo
 Couenne::CouenneInfo, 103

CouenneInterface
 Couenne::CouenneInterface, 104

CouenneIterativeRounding
 Couenne::CouenneIterativeRounding, 107

CouenneMultiVarProbe
 Couenne::CouenneMultiVarProbe, 111

CouenneOSInterface
 Couenne::CouenneOSInterface, 125

CouenneObject
 Couenne::CouenneObject, 115, 116

CouenneObject.hpp
 AGGR_MUL, 446
 THRES_ZERO_SYMM, 446

CouenneObjective
 Couenne::CouenneObjective, 121

CouenneOrbitBranchingObj
 Couenne::CouenneOrbitBranchingObj, 123

CouennePSDcon
 Couenne::CouennePSDcon, 155

CouennePrecisions.hpp
 COU_MAX_COEFF, 475
 COU_MIN_COEFF, 475
 COUENNE_BOUND_PREC, 475
 COUENNE_EPS, 475
 COUENNE_EPS_INT, 475
 COUENNE_EPS_SIMPL, 475
 COUENNE_INFINITY, 475
 COUENNE_round, 475
 COUENNE_sign, 476
 MAX_BOUND, 476

CouenneProblem
 Couenne::CouenneProblem, 135

CouenneProblem.hpp
 COUENNE_EPS_SYMM, 529
 FM_CHECKNLP2, 529
 FM_TRACE_OPTSOL, 529

couennePtr
 Couenne::CouenneSetup, 167

CouenneRecordBestSol
 Couenne::CouenneRecordBestSol, 158

CouenneSOSBranchingObject
 Couenne::CouenneSOSBranchingObject, 174

CouenneSOSObject
 Couenne::CouenneSOSObject, 176

CouenneScalar
 Couenne::CouenneScalar, 161, 162

CouenneSdpCuts
 Couenne::CouenneSdpCuts, 164

CouenneSetup
 Couenne::CouenneSetup, 166

CouenneSolverInterface
 Couenne::CouenneSolverInterface, 170

CouenneSparseBndVec
 Couenne::CouenneSparseBndVec, 178

CouenneSparseMatrix
 Couenne::CouenneSparseMatrix, 179

CouenneSparseVector
 Couenne::CouenneSparseVector, 182

CouenneTNLP
 Couenne::CouenneTNLP, 187

CouenneThreeWayBranchObj
 Couenne::CouenneThreeWayBranchObj, 184

CouenneTwoImplied
 Couenne::CouenneTwoImplied, 193

CouenneUserInterface
 Couenne::CouenneUserInterface, 195

CouenneVTOBJECT
 Couenne::CouenneVTOBJECT, 201

CouenneVarObject
 Couenne::CouenneVarObject, 198

Counter
 Couenne::DepGraph, 205

counter_
 Couenne::DepGraph, 206

cpuTime_
 Couenne::CouenneDisjCuts, 86

createBranch
 Couenne::CouenneComplObject, 67
 Couenne::CouenneObject, 116
 Couenne::CouenneSOSObject, 176
 Couenne::CouenneVarObject, 199

createCut
 Couenne::CouenneCutGenerator, 76, 77

createOrder
 Couenne::DepGraph, 205
 Couenne::DepNode, 208

createQuadratic
 Couenne::exprSum, 372

createRow
 Couenne::CouenneFixPoint, 96

createUnusedOriginals
 Couenne::CouenneProblem, 144

created_pcutoff_

Couenne::CouenneProblem, 150
crossBounds
 Couenne::exprAux, 226
 Couenne::exprVar, 400
currvars_
 Couenne::CouenneProblem, 148
current
 Couenne::Domain, 212
CutGen
 Couenne::CouenneSolverInterface, 170
cutGen_
 Couenne::CouenneBranchingObject, 52
 Couenne::CouenneObject, 118
cutgen_
 Couenne::CouenneSolverInterface, 172

DECLIN
 Couenne, 30
DEP_BLACK
 Couenne::DepNode, 208
DEP_GRAY
 Couenne::DepNode, 208
DEP_WHITE
 Couenne::DepNode, 208
data
 Couenne::CouenneSparseBndVec, 178
decomposeTerm
 Couenne::CouenneProblem, 143
decreaseMult
 Couenne::exprAux, 227
 Couenne::exprVar, 402
default_alpha
 Couenne, 36
default_clamp
 Couenne, 36
deleteElement
 Nauty, 415
dep_color
 Couenne::DepNode, 208
DepGraph
 Couenne::DepGraph, 204
DepList
 Couenne::DepNode, 208
 Couenne::exprAux, 226
 Couenne::exprCopy, 248
 Couenne::expression, 267
 Couenne::exprGroup, 284
 Couenne::exprOp, 336
 Couenne::exprQuad, 355
 Couenne::exprUnary, 390
 Couenne::exprVar, 400
depList
 Couenne::DepNode, 209
depList_
 Couenne::DepNode, 209
 Couenne::DepNode, 209
DepNode
 Couenne::DepNode, 208
Dependence
 Couenne::CouenneProblem, 143
dependence_
 Couenne::CouenneProblem, 151
depends
 Couenne::DepGraph, 205
 Couenne::DepNode, 208
dependsOn
 Couenne::exprConst, 241
 Couenne::expression, 267
 Couenne::exprLowerBound, 316
 Couenne::exprUpperBound, 395
depthLevelling_
 Couenne::CouenneDisjCuts, 85
 Couenne::CouenneTwoImplied, 194
depthStopSeparate_
 Couenne::CouenneDisjCuts, 85
 Couenne::CouenneTwoImplied, 194
differentiate
 Couenne::exprAbs, 219
 Couenne::exprBinProd, 230
 Couenne::exprCeil, 234
 Couenne::exprConst, 241
 Couenne::exprCopy, 248
 Couenne::exprCos, 254
 Couenne::exprDiv, 258
 Couenne::expression, 267
 Couenne::exprExp, 274
 Couenne::exprFloor, 278
 Couenne::exprGroup, 284
 Couenne::exprInv, 291
 Couenne::exprLog, 311
 Couenne::exprLowerBound, 316
 Couenne::exprMax, 319
 Couenne::exprMin, 323
 Couenne::exprMultiLin, 325
 Couenne::exprOpp, 340
 Couenne::exprPow, 345
 Couenne::exprQuad, 353
 Couenne::exprSin, 359
 Couenne::exprSub, 366
 Couenne::exprSum, 370
 Couenne::exprUpperBound, 395
 Couenne::exprVar, 400
dig_type
 Couenne, 30
Dimension
 Couenne::DomainPoint, 215
dimension_
 Couenne::DomainPoint, 216
displayStats

Couenne::CouenneSetup, 167
doABT
 Couenne::CouenneProblem, 140
doABT_
 Couenne::CouenneProblem, 150
doConvCuts_
 Couenne::CouenneBranchingObject, 53
 Couenne::CouenneObject, 119
 Couenne::CouenneSOSBranchingObject, 174
 Couenne::CouenneSOSObject, 177
doFBBT
 Couenne::CouenneProblem, 140
doFBBT_
 Couenne::CouenneBranchingObject, 53
 Couenne::CouenneObject, 119
 Couenne::CouenneProblem, 150
 Couenne::CouenneSOSBranchingObject, 174
 Couenne::CouenneSOSObject, 177
doNotUse
 Couenne::CouenneSdpCuts, 164
doNotUse_
 Couenne::CouenneSdpCuts, 165
doOBBT
 Couenne::CouenneProblem, 140
doOBBT_
 Couenne::CouenneProblem, 150
doPrint_
 Couenne::CouenneProblem, 148
doRCBT
 Couenne::CouenneProblem, 140
doRCBT_
 Couenne::CouenneProblem, 150
doStrongBranching
 Couenne::CouenneChooseStrong, 59
domStack_
 Couenne::Domain, 213
Domain
 Couenne::Domain, 211
 Couenne::DomainPoint, 215
domain
 Couenne::CouenneProblem, 138
 Couenne::exprVar, 402
domain_
 Couenne::CouenneProblem, 148
 Couenne::exprVar, 403
DomainPoint
 Couenne::DomainPoint, 214
downEstimate
 Couenne::CouenneObject, 117
downEstimate_
 Couenne::CouenneBranchingObject, 53
 Couenne::CouenneObject, 119
draw_cuts
 Couenne, 32
dsyevx_interface
 dsyevx_wrapper.hpp, 460
dsyevx_wrapper.hpp
 dsyevx_interface, 460
EAll
 Couenne, 31
EFiltersQP
 Couenne, 31
Elpopt
 Couenne, 31
EMPTY
 Couenne, 28
EXACT
 Couenne::t_chg_bounds, 432
EXPR_OBJ
 Couenne::CouenneObject, 115
eigen_
 Couenne::exprQuad, 356
elem_
 Couenne::CouenneScalar, 162
 Couenne::CouenneSparseVector, 183
enable_lp_implied_bounds_
 Couenne::CouenneCutGenerator, 80
enableLplImpliedBounds
 Couenne::CouenneCutGenerator, 78
erase
 Couenne::DepGraph, 205
estimateProduct_
 Couenne::CouenneChooseStrong, 59
eval_f
 Couenne::CouenneTNLP, 188
eval_g
 Couenne::CouenneTNLP, 188
eval_grad_f
 Couenne::CouenneTNLP, 188
eval_h
 Couenne::CouenneTNLP, 189
eval_jac_g
 Couenne::CouenneTNLP, 188
evalOrder
 Couenne::CouenneProblem, 137
evalVector
 Couenne::CouenneProblem, 137
exp
 Couenne, 33, 34
exponent_
 Couenne::powertriplet, 422
expr
 Couenne::ExprHess, 287
 Couenne::ExprJac, 297
expr_type
 Couenne, 29
exprAbs

Couenne::exprAbs, 218
exprAux
 Couenne::exprAux, 225
exprBinProd
 Couenne::exprBinProd, 230
exprCeil
 Couenne::exprCeil, 233
exprClone
 Couenne::exprClone, 237
exprConst
 Couenne::exprConst, 240
exprCopy
 Couenne::exprCopy, 245
exprCos
 Couenne::exprCos, 253
exprDiv
 Couenne::exprDiv, 258
exprExp
 Couenne::exprExp, 273
exprFloor
 Couenne::exprFloor, 277
exprGroup
 Couenne::exprGroup, 283
ExprHess
 Couenne::ExprHess, 287
exprIVar
 Couenne::exprIVar, 295
exprInv
 Couenne::exprInv, 291
ExprJac
 Couenne::ExprJac, 296
exprLBCos
 Couenne::exprLBCos, 298
exprLBDiv
 Couenne::exprLBDiv, 301
exprLBMul
 Couenne::exprLBMul, 303
exprLBQuad
 Couenne::exprLBQuad, 305
exprLBSin
 Couenne::exprLBSin, 307
exprLog
 Couenne::exprLog, 310
exprLowerBound
 Couenne::exprLowerBound, 315
exprMax
 Couenne::exprMax, 318
exprMin
 Couenne::exprMin, 322
exprMul
 Couenne::CouenneProblem, 147
exprMultiLin
 Couenne::exprMultiLin, 325
exprOddPow
 Couenne::exprOddPow, 330
exprOp
 Couenne::exprOp, 334, 335
exprOpp
 Couenne::exprOpp, 340
exprPow
 Couenne::exprPow, 344
exprQuad
 Couenne::exprQuad, 352
exprSin
 Couenne::exprSin, 359
exprStore
 Couenne::exprStore, 363
exprSub
 Couenne::exprSub, 365
exprSum
 Couenne::exprSum, 370
exprTrilinear
 Couenne::exprTrilinear, 373
exprUBCos
 Couenne::exprUBCos, 376
exprUBDiv
 Couenne::exprUBDiv, 378
exprUBMul
 Couenne::exprUBMul, 380
exprUBQuad
 Couenne::exprUBQuad, 383
exprUBSin
 Couenne::exprUBSin, 385
exprUnary
 Couenne::exprUnary, 388
exprUpperBound
 Couenne::exprUpperBound, 394
exprVar
 Couenne::exprVar, 398, 399
Expression
 Couenne::CouExpr, 203
expression
 Couenne::expression, 264
extendedModel_
 Couenne::CouenneFixPoint, 96
extractLinearRelaxation
 Couenne::CouenneInterface, 105

F

Couenne::exprAbs, 219
Couenne::exprCeil, 234
Couenne::exprCos, 254
Couenne::exprExp, 274
Couenne::exprFloor, 278
Couenne::exprInv, 291
Couenne::exprLog, 311
Couenne::exprOpp, 340
Couenne::exprSin, 359

Couenne::exprUnary, 389
Couenne::funtriplet, 405
Couenne::kpowertriplet, 409
Couenne::powertriplet, 421
Couenne::simpletriplet, 428
FIX_AT_ONE
 Nauty, 414
FIX_AT_ZERO
 Nauty, 414
FP_CUT_EXTERNAL
 Couenne::CouenneFeasPump, 91
FP_CUT_INTEGRATED
 Couenne::CouenneFeasPump, 91
FP_CUT_NONE
 Couenne::CouenneFeasPump, 91
FP_CUT_POST
 Couenne::CouenneFeasPump, 91
FP_DIST_ALL
 Couenne::CouenneFeasPump, 90
FP_DIST_INT
 Couenne::CouenneFeasPump, 90
FP_DIST_POST
 Couenne::CouenneFeasPump, 90
FP_TABU_CUT
 Couenne::CouenneFeasPump, 91
FP_TABU_NONE
 Couenne::CouenneFeasPump, 91
FP_TABU_PERTURB
 Couenne::CouenneFeasPump, 91
FP_TABU_POOL
 Couenne::CouenneFeasPump, 91
FREE
 Nauty, 414
f_
 Couenne::simpletriplet, 429
FM_CHECKNLP2
 CouenneProblem.hpp, 529
FM_TRACE_OPTSOL
 CouenneProblem.hpp, 529
fadingCoeff
 CouenneFeasPump.hpp, 516
fake_tighten
 Couenne::CouenneProblem, 145
ffbtReachedIterLimit
 Couenne::CouenneProblem, 144
ffbtReachedIterLimit_
 Couenne::CouenneProblem, 152
feas_tolerance_
 Couenne::CouenneObject, 118
 Couenne::CouenneProblem, 151
feas_tolerance_default
 Couenne, 36
feasibleRegion
 Couenne::CouenneObject, 116
 Couenne::CouenneVarObject, 199
feasibleSolution
 Couenne::CouenneChooseStrong, 59
 Couenne::CouenneChooseVariable, 62
fillDepSet
 Couenne::exprCopy, 250
 Couenne::expression, 270
 Couenne::exprGroup, 285
 Couenne::exprOp, 337
 Couenne::exprQuad, 355
 Couenne::exprUnary, 391
 Couenne::exprVar, 402
fillDependence
 Couenne::CouenneProblem, 146
fillIntegerRank
 Couenne::CouenneProblem, 146
fillMissingTerms_
 Couenne::CouenneSdpCuts, 165
fillObjCoeff
 Couenne::CouenneProblem, 142
fillQuadIndices
 Couenne::CouenneProblem, 142
finalize_solution
 Couenne::CouenneTNLP, 189
Find_Orbit
 Couenne::CouenneProblem, 137
findClosestAndReplace
 Couenne::CouenneFPpool, 98
findRelations
 Couenne::AuxRelation, 37
 Couenne::BiProdDivRel, 39
 Couenne::MultiProdRel, 412
 Couenne::PowRel, 423
 Couenne::SumLogAuxRel, 431
findSOS
 Couenne::CouenneProblem, 143
findSolution
 Couenne::CouenneFeasPump, 92
firstBranch_
 Couenne::CouenneThreeWayBranchObj, 185
firstCall_
 Couenne::CouenneFixPoint, 96
 Couenne::CouenneTwoImplied, 194
firstcall_
 Couenne::CouenneCutGenerator, 78
fixIntVariables
 Couenne::CouenneFeasPump, 92
flattenMul
 Couenne::CouenneProblem, 146
Fp
 Couenne::funtriplet, 405
 Couenne::kpowertriplet, 409
 Couenne::powertriplet, 421
 Couenne::simpletriplet, 428

fp_
 Couenne::simpletriplet, 429
fpCompDistIntType
 Couenne::CouenneFeasPump, 90
fpCutPlane
 Couenne::CouenneFeasPump, 90
fpl_
 Couenne::simpletriplet, 429
FplInv
 Couenne::funtriplet, 405
 Couenne::kpowertriplet, 409
 Couenne::powertriplet, 422
 Couenne::simpletriplet, 429
fpTabuMgtPolicy
 Couenne::CouenneFeasPump, 91
Fpp
 Couenne::funtriplet, 405
 Couenne::kpowertriplet, 409
 Couenne::powertriplet, 421
 Couenne::simpletriplet, 428
fpp_
 Couenne::simpletriplet, 429
funtriplet
 Couenne::funtriplet, 404
genColCuts
 Couenne::CouenneCutGenerator, 77
genExprGroup
 Couenne::exprGroup, 283
genRowCuts
 Couenne::CouenneCutGenerator, 77
generateCuts
 Couenne::AuxRelation, 37
 Couenne::BiProdDivRel, 39
 Couenne::CouenneAggrProbing, 44
 Couenne::CouenneCrossConv, 72
 Couenne::CouenneCutGenerator, 76
 Couenne::CouenneDisjCuts, 83
 Couenne::CouenneFixPoint, 95
 Couenne::CouenneMultiVarProbe, 111
 Couenne::CouenneSdpCuts, 164
 Couenne::CouenneTwoImplied, 193
 Couenne::exprAbs, 219
 Couenne::exprAux, 226
 Couenne::exprBinProd, 231
 Couenne::exprCeil, 234
 Couenne::exprConst, 241
 Couenne::exprCopy, 249
 Couenne::exprCos, 254
 Couenne::exprDiv, 259
 Couenne::expression, 269
 Couenne::exprExp, 274
 Couenne::exprFloor, 278
 Couenne::exprGroup, 285
 Couenne::exprInv, 292
 Couenne::exprLog, 311
 Couenne::exprMax, 320
 Couenne::exprMin, 323
 Couenne::exprMultiLin, 325
 Couenne::exprOddPow, 331
 Couenne::exprOpp, 341
 Couenne::exprPow, 346
 Couenne::exprQuad, 353
 Couenne::exprSin, 360
 Couenne::exprSub, 367
 Couenne::exprSum, 371
 Couenne::exprTrilinear, 373
 Couenne::exprVar, 401
 Couenne::MultiProdRel, 412
 Couenne::PowRel, 423
 Couenne::SumLogAuxRel, 431
generateDisjCuts
 Couenne::CouenneDisjCuts, 83
get_bounds_info
 Couenne::CouenneTNLP, 188
get_code
 Node, 419
get_coeff
 Node, 419
get_color
 Node, 419
get_constraints_linearity
 Couenne::CouenneTNLP, 188
get_index
 Node, 419
get_lb
 Node, 419
get_list_of_nonlinear_variables
 Couenne::CouenneTNLP, 189
get_nlp_info
 Couenne::CouenneTNLP, 188
get_number_of_nonlinear_variables
 Couenne::CouenneTNLP, 189
get_sign
 Node, 419
get_starting_point
 Couenne::CouenneTNLP, 188
get_ub
 Node, 419
get_variables_linearity
 Couenne::CouenneTNLP, 188
getAuxs
 Couenne::CouenneProblem, 141
getBounds
 Couenne::exprAbs, 219
 Couenne::exprBinProd, 230
 Couenne::exprCeil, 234
 Couenne::exprConst, 241

Couenne::exprCopy, 248, 249
Couenne::exprCos, 254
Couenne::exprDiv, 259
Couenne::expression, 268
Couenne::exprExp, 274
Couenne::exprFloor, 278
Couenne::exprGroup, 284, 285
Couenne::exprInv, 291
Couenne::exprLog, 311
Couenne::exprMax, 319
Couenne::exprMin, 323
Couenne::exprMultiLin, 325
Couenne::exprOddPow, 331
Couenne::exprOpp, 341
Couenne::exprPow, 345
Couenne::exprQuad, 353
Couenne::exprSin, 360
Couenne::exprSub, 366
Couenne::exprSum, 371
Couenne::exprTrilinear, 373
Couenne::exprVar, 401
getBoxUnion
 Couenne::CouenneDisjCuts, 84
getBrPoint
 Couenne::CouenneObject, 117
getCardInitDom
 Couenne::CouenneRecordBestSol, 158
getCardModSol
 Couenne::CouenneRecordBestSol, 159
getCardSol
 Couenne::CouenneRecordBestSol, 158
getCols
 Couenne::CouenneExprMatrix, 88
getCouenneProblem
 Couenne::CouenneAmplInterface, 47
 Couenne::CouenneOSInterface, 126
 Couenne::CouenneUserInterface, 195
getCutOff
 Couenne::CouenneProblem, 142
 Couenne::GlobalCutOff, 405
getCutOffSol
 Couenne::CouenneProblem, 142
 Couenne::GlobalCutOff, 406
getDepGraph
 Couenne::CouenneProblem, 138
getDisjunctions
 Couenne::CouenneDisjCuts, 83
getDoubleParameter
 Couenne::CouenneSetup, 167
getElem
 Couenne::CouenneScalar, 162
getElements
 Couenne::CouenneSparseVector, 182
getFeasTol
 Nauty, 415
getFixVar
 Couenne::exprOddPow, 331
 Couenne::exprPow, 346
getGroupSize
 Nauty, 415
getHasSol
 Couenne::CouenneRecordBestSol, 158
getIndex
 Couenne::CouenneScalar, 162
getInitDomLb
 Couenne::CouenneRecordBestSol, 158
getInitDomUb
 Couenne::CouenneRecordBestSol, 158
getInitIsInt
 Couenne::CouenneRecordBestSol, 158
getIntegerCandidate
 Couenne::CouenneProblem, 142
getLastPrioSort
 Couenne::CouenneProblem, 146
getListInt
 Couenne::CouenneRecordBestSol, 158
getMaxCpuTime
 Couenne::CouenneProblem, 144
getMaxFailedSteps
 Couenne::CouenneAggrProbing, 44
getMaxNodes
 Couenne::CouenneAggrProbing, 44
getMaxTime
 Couenne::CouenneAggrProbing, 44
getMaxViol
 Couenne::CouenneRecordBestSol, 159
getModSol
 Couenne::CouenneRecordBestSol, 159
getModSolMaxViol
 Couenne::CouenneRecordBestSol, 159
getModSolVal
 Couenne::CouenneRecordBestSol, 159
getN
 Nauty, 415
getNautyCalls
 Nauty, 415
getNautyTime
 Nauty, 415
getNtyInfo
 Couenne::CouenneProblem, 137
getNumGenerators
 Nauty, 415
getNumOrbits
 Nauty, 415
getObjValue
 Couenne::CouenneSolverInterface, 171
getOrbits
 Nauty, 415

getOriginal
 Couenne, 33

getQ
 Couenne::exprQuad, 352

getRecordBestSol
 Couenne::CouenneProblem, 146

getRestoreCutoff
 Couenne::CouenneAggrProbing, 44

getRows
 Couenne::CouenneExprMatrix, 88

getSaveOptHessian
 Couenne::CouenneTNLP, 189

getSdpCutGen
 Couenne::CouenneProblem, 145

getSingleDisjunction
 Couenne::CouenneDisjCuts, 84

getSol
 Couenne::CouenneRecordBestSol, 158

getSolValue
 Couenne::CouenneTNLP, 187

getSolution
 Couenne::CouenneTNLP, 187

getStats
 Couenne::CouenneCutGenerator, 77

getTMINLP
 Couenne::CouenneAmplInterface, 47
 Couenne::CouenneOSInterface, 126
 Couenne::CouenneUserInterface, 195

getTrilinDecompType
 Couenne::CouenneProblem, 145

getVal
 Couenne::CouenneRecordBestSol, 159

getVstat
 Nauty, 415

getX
 Couenne::CouennePSDcon, 155

getc0
 Couenne::exprGroup, 283

getnQTerms
 Couenne::exprQuad, 352

getnvars
 Couenne::CouenneCutGenerator, 76

GlobalCutOff
 Couenne::GlobalCutOff, 405

gradientNorm
 Couenne::exprAbs, 219
 Couenne::exprBinProd, 230
 Couenne::exprCeil, 234
 Couenne::exprCopy, 248
 Couenne::exprCos, 254
 Couenne::exprDiv, 258
 Couenne::expression, 267
 Couenne::exprExp, 274
 Couenne::exprFloor, 278

Couenne::exprGroup, 284
Couenne::exprInv, 291
Couenne::exprLog, 311
Couenne::exprMultiLin, 325
Couenne::exprOpp, 340
Couenne::exprPow, 345
Couenne::exprQuad, 353
Couenne::exprSin, 359
Couenne::exprTrilinear, 373
Couenne::exprVar, 400

graph_
 Couenne::CouenneProblem, 149

gutsOfSetupList
 Couenne::CouenneChooseStrong, 59

hasSol
 Couenne::CouenneRecordBestSol, 160

have_nlp_solution_
 Couenne::CouenneInterface, 105

haveNlpSolution
 Couenne::CouenneInterface, 105

INCLIN
 Couenne, 30

INFEASIBILITY
 Couenne::CouenneObject, 115

INSIDE
 Couenne, 28

INTEGER_VARS
 Couenne, 31

INTERVAL_BR
 Couenne::CouenneObject, 115

INTERVAL_BR_REV
 Couenne::CouenneObject, 115

INTERVAL_LP
 Couenne::CouenneObject, 115

INTERVAL_LP_REV
 Couenne::CouenneObject, 115

iRow
 Couenne::ExprHess, 287
 Couenne::ExprJac, 296

Image
 Couenne::exprAux, 225, 226
 Couenne::exprCopy, 246
 Couenne::expression, 266

image_
 Couenne::exprAux, 228

impliedBound
 Couenne::exprAbs, 220
 Couenne::exprBinProd, 231
 Couenne::exprCeil, 234
 Couenne::exprCopy, 249
 Couenne::exprCos, 254
 Couenne::exprDiv, 259
 Couenne::expression, 270

Couenne::exprExp, 275
Couenne::exprFloor, 278
Couenne::exprInv, 292
Couenne::exprLog, 312
Couenne::exprMultiLin, 326
Couenne::exprOddPow, 331
Couenne::exprOpp, 341
Couenne::exprPow, 346
Couenne::exprQuad, 356
Couenne::exprSin, 360
Couenne::exprSub, 367
Couenne::exprSum, 371
Couenne::exprTrilinear, 374
Couenne::exprVar, 402
impliedBoundMul
 Couenne::exprMultiLin, 326
impliedBoundSum
 Couenne::exprSum, 372
impliedBounds
 Couenne::CouenneProblem, 141
increaseMult
 Couenne::exprAux, 227
indcoe2vector
 Couenne::CouenneProblem, 143
Index
 Couenne::DepNode, 208
 Couenne::exprCopy, 246
 Couenne::expression, 265
 Couenne::exprVar, 399
index_
 Couenne::CouenneScalar, 162
 Couenne::DepNode, 209
index_sort
 Couenne::CouenneProblem, 153
indices
 Couenne::CouenneSparseBndVec, 178
infeasNode
 Couenne::CouenneCutGenerator, 77
infeasNode_
 Couenne::CouenneCutGenerator, 79
infeasibility
 Couenne::CouenneComplObject, 67
 Couenne::CouenneObject, 116
 Couenne::CouenneVarObject, 199
 Couenne::CouenneVTOObject, 202
init_MILP
 Couenne::CouenneFeasPump, 92
initAuxs
 Couenne::CouenneProblem, 141
initCutoff_
 Couenne::CouenneAggrProbing, 45
initDisjNumber_
 Couenne::CouenneDisjCuts, 85
initDisjPercentage_

Couenne::CouenneDisjCuts, 85
initDomLb
 Couenne::CouenneRecordBestSol, 159
initDomUb
 Couenne::CouenneRecordBestSol, 160
InitHeuristic
 Couenne::InitHeuristic, 406
initIpoptApp
 Couenne::CouenneFeasPump, 92
initIlsInt
 Couenne::CouenneRecordBestSol, 159
initOptions
 Couenne::CouenneProblem, 136
initialSolve
 Couenne::CouenneSolverInterface, 171
InitializeCouenne
 Couenne::CouenneSetup, 167
insert
 Couenne::DepGraph, 205
 Couenne::LinMap, 411
 Couenne::QuadMap, 427
insertRHS
 Nauty, 415
installCutOff
 Couenne::CouenneProblem, 142
intInfeasibility
 Couenne::CouenneObject, 117
intType
 Couenne::exprAux, 224
Integer
 Couenne::exprAux, 224
integer_
 Couenne::exprAux, 228
integerRank_
 Couenne::CouenneProblem, 151
intermediate_callback
 Couenne::CouenneTNLP, 189
inv
 Couenne, 34
inv_dblprime
 Couenne, 34
inverse
 Couenne::exprCopy, 250
 Couenne::expression, 271
 Couenne::exprExp, 275
 Couenne::exprInv, 292
 Couenne::exprLog, 312
Ipopt, 37
is_boundbox_regular
 Couenne, 34
isBijective
 Couenne::exprCopy, 250
 Couenne::expression, 271
 Couenne::exprExp, 275

Couenne::exprInv, 292
Couenne::exprLog, 312
isBranchingStrong_
 Couenne::CouenneDisjCuts, 85
isCuttable
 Couenne::CouenneObject, 117
 Couenne::CouenneVarObject, 199
 Couenne::exprAbs, 220
 Couenne::exprBinProd, 231
 Couenne::exprCeil, 235
 Couenne::exprCopy, 251
 Couenne::exprCos, 255
 Couenne::exprDiv, 260
 Couenne::expression, 271
 Couenne::exprExp, 275
 Couenne::exprFloor, 279
 Couenne::exprInv, 292
 Couenne::exprLog, 312
 Couenne::exprMultiLin, 326
 Couenne::exprOddPow, 331
 Couenne::exprPow, 347
 Couenne::exprQuad, 356
 Couenne::exprSin, 360
isDefinedInteger
 Couenne::exprAux, 227
 Couenne::exprCopy, 248
 Couenne::expression, 268
 Couenne::exprVar, 295
 Couenne::exprVar, 401
isFirst
 Couenne::CouenneCutGenerator, 76
isFixed
 Couenne::exprVar, 402
isInteger
 Couenne, 33
 Couenne::exprAbs, 220
 Couenne::exprAux, 227
 Couenne::exprConst, 242
 Couenne::exprCopy, 248
 Couenne::exprDiv, 259
 Couenne::expression, 268
 Couenne::exprGroup, 285
 Couenne::exprVar, 295
 Couenne::exprOp, 337
 Couenne::exprOpp, 341
 Couenne::exprPow, 345
 Couenne::exprQuad, 355
 Couenne::exprUnary, 390
 Couenne::exprVar, 401
isNlp
 Couenne::DomainPoint, 215
isNlp_
 Couenne::DomainPoint, 216
isProvenDualInfeasible
 Couenne::CouenneSolverInterface, 171
isProvenOptimal
 Couenne::CouenneSolverInterface, 170
isProvenPrimallInfeasible
 Couenne::CouenneSolverInterface, 170
isSignpower
 Couenne::exprPow, 347
isWiped
 CouenneInfeasCut.hpp, 436
isaCopy
 Couenne::exprCopy, 246
 Couenne::expression, 271
issignpower_
 Couenne::powertriplet, 422
J_BOUNDTIGHTENING
 Couenne, 35
J_BRANCHING
 Couenne, 35
J_CONVEXIFYING
 Couenne, 35
J_COUENNE
 Couenne, 36
J_DISJCUTS
 Couenne, 36
J_NLPHEURISTIC
 Couenne, 35
J_PROBLEM
 Couenne, 35
J_REFORMULATE
 Couenne, 36
jCol
 Couenne::ExprHess, 287
 Couenne::ExprJac, 296
Jnlst
 Couenne::CouenneCutGenerator, 78
 Couenne::CouenneDisjCuts, 83
 Couenne::CouenneProblem, 142
jnlist
 Couenne::CouenneUserInterface, 196
jnlist_
 Couenne::CouenneBranchingObject, 53
 Couenne::CouenneChooseStrong, 60
 Couenne::CouenneChooseVariable, 62
 Couenne::CouenneCrossConv, 72
 Couenne::CouenneCutGenerator, 79
 Couenne::CouenneDisjCuts, 85
 Couenne::CouenneObject, 118
 Couenne::CouenneProblem, 150
 Couenne::CouenneSOSBranchingObject, 174
 Couenne::CouenneSOSObject, 176
 Couenne::CouenneThreeWayBranchObj, 185
 Couenne::CouenneTwoImplied, 193
JnlstPtr

Couenne, 27

knowDualInfeasible_
 Couenne::CouenneSolverInterface, 172

knowInfeasible_
 Couenne::CouenneSolverInterface, 172

knowOptimal_
 Couenne::CouenneSolverInterface, 172

kpowertriplet
 Couenne::kpowertriplet, 409

LINEAR
 Couenne, 28

LP_CENTRAL
 Couenne::CouenneObject, 115

LP_CLAMPED
 Couenne::CouenneObject, 115

laml
 Couenne::ExprHess, 287

large_bound
 Couenne, 36

lastPrintLine
 Couenne::CouenneCutGenerator, 80

lastPrioSort_
 Couenne::CouenneProblem, 152

Lb
 Couenne::CouenneConstraint, 69
 Couenne::CouenneProblem, 138
 Couenne::exprAux, 225
 Couenne::exprVar, 399

lb
 Couenne::Domain, 212
 Couenne::DomainPoint, 215
 Couenne::exprVar, 399

lb_
 Couenne::CouenneConstraint, 70
 Couenne::DomainPoint, 216
 Couenne::exprAux, 228

lcoeff
 Couenne::exprGroup, 283

lccoeff_
 Couenne::exprGroup, 286

lcrop_
 Couenne::CouenneThreeWayBranchObj, 184

less_than_str, 410
 operator(), 410

linStandardize
 Couenne::CouenneProblem, 142

lincoeff
 Couenne::exprGroup, 283

Linearity
 Couenne::exprAux, 226
 Couenne::exprBinProd, 230
 Couenne::exprConst, 241
 Couenne::exprCopy, 248

Couenne::exprDiv, 258
Couenne::expression, 268
Couenne::exprGroup, 284
Couenne::exprInv, 291
Couenne::exprLowerBound, 316
Couenne::exprMax, 319
Couenne::exprMin, 323
Couenne::exprMultiLin, 325
Couenne::exprOp, 336
Couenne::exprOpp, 341
Couenne::exprPow, 345
Couenne::exprQuad, 353
Couenne::exprSub, 366
Couenne::exprSum, 371
Couenne::exprUnary, 390
Couenne::exprUpperBound, 395
Couenne::exprVar, 401

linearity_type
 Couenne, 28

linkDomain
 Couenne::exprAux, 227
 Couenne::expression, 270
 Couenne::exprVar, 402

listInt
 Couenne::CouenneRecordBestSol, 159

log
 Couenne, 33, 34

logAbtLev
 Couenne::CouenneProblem, 140

logAbtLev_
 Couenne::CouenneProblem, 150

logObbtLev
 Couenne::CouenneProblem, 140

logObbtLev_
 Couenne::CouenneProblem, 150

lookup
 Couenne::DepGraph, 205

lower
 Couenne::t_chg_bounds, 432

lp_clamp
 Couenne::CouenneObject, 118

lp_clamp_
 Couenne::CouenneObject, 118

MAX_COU_EXPR_CODE
 Couenne, 30

MID_INTERVAL
 Couenne::CouenneObject, 115

MIN_AREA
 Couenne::CouenneObject, 115

MON_CONST
 Couenne, 30

MON_UNSET
 Couenne, 30

MON_ZERO
 Couenne, 30

M_PI
 CouenneExprBCos.hpp, 480
 CouenneExprBSin.hpp, 484

MAX_ARG_LINE
 CouenneExprOp.hpp, 470

MAX_BOUND
 CouennePrecisions.hpp, 476

MUL_INF
 CouenneExprBMul.hpp, 482

MUL_ZERO
 CouenneExprBMul.hpp, 482

Map
 Couenne::LinMap, 411
 Couenne::QuadMap, 426

markHotStart
 Couenne::CouenneSolverInterface, 171

matrix_
 Couenne::exprQuad, 356

max_fbbt_iter_
 Couenne::CouenneProblem, 152

max_pseudocost
 Couenne, 36

maxCpuTime_
 Couenne::CouenneProblem, 151

maxDepthOrbBranch
 Couenne::CouenneBranchingObject, 52

maxFailedSteps_
 Couenne::CouenneAggrProbing, 45

maxHeight
 Couenne, 32

maxlInf_
 Couenne::CouenneFPsolution, 101

maxNLInf_
 Couenne::CouenneFPsolution, 101

maxNlplInf_0
 Couenne, 36

maxNodes_
 Couenne::CouenneAggrProbing, 45

maxTime_
 Couenne::CouenneAggrProbing, 45
 Couenne::CouenneMultiVarProbe, 111

maxViol
 Couenne::CouenneRecordBestSol, 160

mergeBoxes
 Couenne::CouenneDisjCuts, 84

midInterval
 Couenne::CouenneObject, 117

milpPhase
 Couenne::CouenneFeasPump, 93

minDepthPrint_
 Couenne::CouenneProblem, 147

minMaxDelta

Couenne, 32

minNodePrint_
 Couenne::CouenneProblem, 148

minlp_
 Couenne::CouenneDisjCuts, 85

minors_
 Couenne::CouenneSdpCuts, 165

modSol
 Couenne::CouenneRecordBestSol, 160

modSolMaxViol
 Couenne::CouenneRecordBestSol, 160

modSolVal
 Couenne::CouenneRecordBestSol, 160

modulo
 Couenne, 35

monotonicity
 Couenne, 30

MulSepNone
 Couenne::CouenneProblem, 135

MulSepSimple
 Couenne::CouenneProblem, 135

MulSepTight
 Couenne::CouenneProblem, 135

mult_
 Couenne::kpowertriplet, 409

multDistMILP
 Couenne::CouenneFeasPump, 93

multDistNLP
 Couenne::CouenneFeasPump, 93

multHessMILP
 Couenne::CouenneFeasPump, 93

multHessNLP
 Couenne::CouenneFeasPump, 93

multObjFMILP
 Couenne::CouenneFeasPump, 93

multObjFNLP
 Couenne::CouenneFeasPump, 93

multiSep
 Couenne::CouenneProblem, 135

MultilinSep
 Couenne::CouenneProblem, 144

multilinSep_
 Couenne::CouenneProblem, 152

Multiplicity
 Couenne::exprAux, 227
 Couenne::exprCopy, 250
 Couenne::expression, 270

multiplicity_
 Couenne::exprAux, 228

multiply_thres
 Couenne::CouenneSparseVector, 182

myclass, 412
 operator(), 413

myclass0, 413

operator(), 413

n
 Couenne::CouenneFPSolution, 100

N_ARY
 Couenne, 28

NDECreas
 Couenne, 30

NINCreas
 Couenne, 30

NO_BRANCH
 Couenne::CouenneObject, 115

NO_STRATEGY
 Couenne::CouenneObject, 115

NONCONVEX
 Couenne, 30

NONE
 Couenne, 28

NONLINEAR
 Couenne, 28

NONMONOTONE
 Couenne, 30

n_
 Couenne::CouenneFPSolution, 101

nArgs
 Couenne::exprCopy, 246
 Couenne::expression, 265
 Couenne::exprOp, 335
 Couenne::exprUnary, 389

nCalls
 Couenne::CouenneFeasPump, 93

nCons
 Couenne::CouenneProblem, 136

nDefVars
 Couenne::CouenneProblem, 136

nElements
 Couenne::CouenneSparseBndVec, 178

nFixed_
 Couenne::CouenneBTPerfIndicator, 55

nlinf_
 Couenne::CouenneFPSolution, 101

nIntVars
 Couenne::CouenneProblem, 136

nIntVars_
 Couenne::CouenneProblem, 148

nMaxTrials_
 Couenne::CouenneTwoImplied, 193

nNLinf_
 Couenne::CouenneFPSolution, 101

nObjs
 Couenne::CouenneProblem, 136

nOrbBr
 Couenne::CouenneBranchingObject, 52

nOrigCons
 Couenne::CouenneProblem, 136

nOrigCons_
 Couenne::CouenneProblem, 149

nOrigIntVars
 Couenne::CouenneProblem, 136

nOrigIntVars_
 Couenne::CouenneProblem, 149

nOrigVars
 Couenne::CouenneProblem, 136

nOrigVars_
 Couenne::CouenneProblem, 149

nProvedInfeas_
 Couenne::CouenneBTPerfIndicator, 56

nRows
 Couenne::ExprJac, 297

nRuns_
 Couenne::CouenneBTPerfIndicator, 56

nSGcomputations
 Couenne::CouenneBranchingObject, 52

nSamples
 Couenne::CouenneCutGenerator, 76

nSamples_
 Couenne::CouenneCutGenerator, 78

nTightened_
 Couenne::CouenneFixPoint, 96

nUnusedOriginals
 Couenne::CouenneProblem, 144

nUnusedOriginals_
 Couenne::CouenneProblem, 152

nVars
 Couenne::CouenneInfo::NlpSolution, 416
 Couenne::CouenneProblem, 136

name_
 Couenne::CouenneBTPerfIndicator, 55

nargs_
 Couenne::exprOp, 337

Nauty, 413
 ~Nauty, 414
 addElement, 414
 clearPartitions, 414
 color_node, 415
 computeAuto, 414
 deleteElement, 415
 FIX_AT_ONE, 414
 FIX_AT_ZERO, 414
 FREE, 414
 getGroupSize, 415
 getN, 415
 getNautyCalls, 415
 getNautyTime, 415
 getNumGenerators, 415
 getNumOrbits, 415
 getOrbits, 415
 getVstat, 415

insertRHS, 415
Nauty, 414
setWriteAutoms, 415
unsetWriteAutoms, 415
VarStatus, 414
nauty_info
 Couenne::CouenneProblem, 153
undefined_
 Couenne::CouenneProblem, 149
nlp
 Couenne::CouenneFeasPump, 93
nlp_
 Couenne::CouenneCutGenerator, 79
nlpPhase
 Couenne::CouenneFeasPump, 93
nlpSols_
 Couenne::CouenneInfo, 104
NlpSolution
 Couenne::CouenneInfo::NlpSolution, 416
NlpSolutions
 Couenne::CouenneInfo, 103
NlpSolveHeuristic
 Couenne::NlpSolveHeuristic, 417
nnz
 Couenne::ExprHess, 287
 Couenne::ExprJac, 296
Node, 418
 bounds, 419
 color_vertex, 419
 get_code, 419
 get_coeff, 419
 get_color, 419
 get_index, 419
 get_lb, 419
 get_sign, 419
 get_ub, 419
 node, 419
node
 Node, 419
node_info
 Couenne::CouenneProblem, 153
node_sort
 Couenne::CouenneProblem, 153
nodeType
 Couenne, 28
nqterms_
 Couenne::exprQuad, 357
nrootcuts_
 Couenne::CouenneCutGenerator, 79
 Couenne::CouenneDisjCuts, 84
ntotalcuts_
 Couenne::CouenneCutGenerator, 79
 Couenne::CouenneDisjCuts, 84
num
 Couenne::CouenneSparseMatrix, 180
numCols_
 Couenne::CouenneAggrProbing, 45
 Couenne::CouenneMultiVarProbe, 111
numDisjunctions_
 Couenne::CouenneDisjCuts, 85
numEigVec_
 Couenne::CouenneSdpCuts, 165
numL
 Couenne::ExprHess, 287
numberInRank_
 Couenne::CouenneProblem, 151
numbering_
 Couenne::CouenneProblem, 149
OBJVAL
 Couenne, 31
ORIG_ONLY
 Couenne, 31
obbt
 Couenne::CouenneProblem, 141
obbt_iter
 Couenne::CouenneProblem, 145
obbtInner
 Couenne::CouenneProblem, 145
Obj
 Couenne::CouenneProblem, 137
objVal
 Couenne::CouenneInfo::NlpSolution, 416
objVal_
 Couenne::CouenneFPSolution, 101
objValue_
 Couenne::CouenneCutGenerator, 79
 Couenne::CouenneDisjCuts, 85
objectives_
 Couenne::CouenneProblem, 148
Objects
 Couenne::CouenneProblem, 143
objects_
 Couenne::CouenneProblem, 151
oldLB_
 Couenne::CouenneBTPerfIndicator, 56
oldUB_
 Couenne::CouenneBTPerfIndicator, 56
onlyNegEV_
 Couenne::CouenneSdpCuts, 165
operator<
 Couenne, 32, 35
 Couenne::CouenneScalar, 162
operator*
 Couenne, 33, 34
 Couenne::CouenneExprMatrix, 88
 Couenne::CouenneSparseVector, 182
operator^

Couenne, 33, 34
operator()
 Couenne::compareSol, 40
 Couenne::compExpr, 41
 Couenne::compNode, 41
 Couenne::CouenneExprMatrix::compare_pair_ind, 39
 Couenne::CouenneSparseVector::compare_scalars, 40
 Couenne::exprAux, 226
 Couenne::exprClone, 238
 Couenne::exprConst, 241
 Couenne::exprCopy, 247
 Couenne::exprDiv, 258
 Couenne::expression, 267
 Couenne::exprGroup, 284
 Couenne::exprLBCos, 299
 Couenne::exprLBDiv, 301
 Couenne::exprLBMul, 303
 Couenne::exprLBQuad, 305
 Couenne::exprLBSin, 308
 Couenne::exprLowerBound, 316
 Couenne::exprMax, 319
 Couenne::exprMin, 322
 Couenne::exprOddPow, 330
 Couenne::exprPow, 345
 Couenne::exprQuad, 352
 Couenne::exprStore, 363
 Couenne::exprSub, 366
 Couenne::exprSum, 370
 Couenne::exprUBCos, 376
 Couenne::exprUBDiv, 378
 Couenne::exprUBMul, 381
 Couenne::exprUBQuad, 383
 Couenne::exprUBSin, 385
 Couenne::exprUnary, 389
 Couenne::exprUpperBound, 394
 Couenne::exprVar, 400
 Couenne::Qroot, 425
 less_than_str, 410
 myclass, 413
 myclass0, 413
operator+
 Couenne, 33, 34
operator-
 Couenne, 33, 34
operator/
 Couenne, 33, 34
operator=
 Couenne::CouenneBTPerfIndicator, 55
 Couenne::CouenneChooseStrong, 58
 Couenne::CouenneChooseVariable, 61
 Couenne::CouenneExprMatrix, 87
 Couenne::CouenneFeasPump, 91
Couenne::CouenneFPpool, 98
Couenne::CouenneFPsolution, 100
Couenne::CouennelterativeRounding, 107
Couenne::CouennePSDcon, 155
Couenne::CouenneScalar, 162
Couenne::CouenneSdpCuts, 164
Couenne::CouenneSparseMatrix, 180
Couenne::CouenneSparseVector, 182
Couenne::CouenneTNLP, 187
Couenne::CouExpr, 203
Couenne::DomainPoint, 215
Couenne::ExprHess, 287
Couenne::ExprJac, 296
Couenne::InitHeuristic, 407
Couenne::NlpSolveHeuristic, 418
Couenne::t_chg_bounds, 433
operator%
 Couenne, 33, 34
opp
 Couenne, 35
opplnvsqr
 Couenne, 34
opt_window_
 Couenne::CouenneProblem, 150
optHessian
 Couenne::CouenneTNLP, 189
optimum_
 Couenne::CouenneProblem, 149
options
 Couenne::CouenneMINLPInterface, 109
 Couenne::CouenneUserInterface, 196
orbitalBranching
 Couenne::CouenneProblem, 144
orbitalBranching_
 Couenne::CouenneProblem, 152
Order
 Couenne::DepNode, 208
order_
 Couenne::DepNode, 209
Original
 Couenne::exprCopy, 246
 Couenne::expression, 266
Osi, 37
OsiCuts2MatrVec
 Couenne::CouenneDisjCuts, 84
OsiSl2MatrVec
 Couenne::CouenneDisjCuts, 84
POST
 Couenne, 28
PRE
 Couenne, 28
PROJECTDIST
 Couenne::CouenneObject, 115

pcutoff_ Couenne::CouenneProblem, 149
perfIndicator_ Couenne::CouenneFixPoint, 96
 Couenne::CouenneProblem, 152
point_ Couenne::Domain, 212
pop Couenne::Domain, 212
pos Couenne, 28
powNewton Couenne, 35
powertriplet Couenne::powertriplet, 421
print Couenne::CouenneConstraint, 70
 Couenne::CouenneExprMatrix, 88
 Couenne::CouenneObjective, 121
 Couenne::CouenneProblem, 140
 Couenne::CouennePSDcon, 155
 Couenne::CouenneScalar, 162
 Couenne::CouenneSparseVector, 182
 Couenne::DepGraph, 205
 Couenne::DepNode, 208
 Couenne::exprAux, 225
 Couenne::exprClone, 237
 Couenne::exprConst, 240
 Couenne::exprCopy, 247
 Couenne::expression, 266
 Couenne::exprGroup, 283
 Couenne::exprInv, 291
 Couenne::exprVar, 295
 Couenne::exprLBQuad, 306
 Couenne::exprLowerBound, 315
 Couenne::exprOp, 335
 Couenne::exprOpp, 340
 Couenne::exprQuad, 352
 Couenne::exprStore, 363
 Couenne::exprUBQuad, 383
 Couenne::exprUnary, 389
 Couenne::exprUpperBound, 394
 Couenne::exprVar, 400
Print_Orbits Couenne::CouenneProblem, 137
printLineInfo Couenne::CouenneCutGenerator, 78
printOp Couenne::exprAbs, 219
 Couenne::exprCeil, 234
 Couenne::exprCos, 254
 Couenne::exprDiv, 258
 Couenne::exprExp, 274
 Couenne::exprFloor, 278
 Couenne::exprLBCos, 299
 Couenne::exprLBDiv, 301
 Couenne::exprLBMul, 303
 Couenne::exprLBSin, 308
 Couenne::exprLog, 311
 Couenne::exprMax, 319
 Couenne::exprMin, 322
 Couenne::exprOddPow, 330
 Couenne::exprOp, 335
 Couenne::exprPow, 345
 Couenne::exprSin, 359
 Couenne::exprSub, 366
 Couenne::exprSum, 370
 Couenne::exprUBCos, 376
 Couenne::exprUBDiv, 378
 Couenne::exprUBMul, 381
 Couenne::exprUBSin, 385
 Couenne::exprUnary, 389
printPos Couenne::exprLBCos, 299
 Couenne::exprLBDiv, 301
 Couenne::exprLBMul, 303
 Couenne::exprLBSin, 308
 Couenne::exprMax, 319
 Couenne::exprMin, 322
 Couenne::exprOp, 335
 Couenne::exprUBCos, 376
 Couenne::exprUBDiv, 378
 Couenne::exprUBMul, 381
 Couenne::exprUBSin, 385
 Couenne::exprUnary, 389
printSol Couenne::CouenneRecordBestSol, 159
probeVariable Couenne::CouenneAggrProbing, 44
probeVariable2 Couenne::CouenneAggrProbing, 44
Problem Couenne::CouenneCutGenerator, 76
 Couenne::CouenneFeasPump, 92
 Couenne::CouenneFPpool, 98
problem Couenne::CouenneMINLPInterface, 109
problem_ Couenne::CouenneBab, 49
 Couenne::CouenneBranchingObject, 52
 Couenne::CouenneBTPerIndicator, 56
 Couenne::CouenneChooseStrong, 59
 Couenne::CouenneChooseVariable, 62
 Couenne::CouenneCrossConv, 72
 Couenne::CouenneCutGenerator, 79
 Couenne::CouenneFixPoint, 96
 Couenne::CouenneFPpool, 98
 Couenne::CouenneFPsolution, 101

Couenne::CouenneObject, 118
Couenne::CouenneSdpCuts, 165
Couenne::CouenneSOSBranchingObject, 174
Couenne::CouenneSOSObject, 176
Couenne::CouenneTwoImplied, 193
problemName
 Couenne::CouenneProblem, 143
problemName_
 Couenne::CouenneProblem, 148
project
 Couenne, 32
projectSeg
 Couenne, 32
properObject
 Couenne::exprAux, 228
 Couenne::exprVar, 403
pseudoMultType_
 Couenne::CouenneObject, 119
pseudoUpdateLP_
 Couenne::CouenneChooseStrong, 59
pseudocostMult
 Couenne::CouenneObject, 115
push
 Couenne::Domain, 211, 212

QUADRATIC
 Couenne, 28

Qmap
 Couenne::Qroot, 425

Qroot
 Couenne::Qroot, 425

quadCuts
 Couenne::exprQuad, 353

quadElem
 Couenne::quadElem, 426

rAI
 Couenne, 32

rank
 Couenne::exprAux, 226
 Couenne::exprConst, 242
 Couenne::exprCopy, 249
 Couenne::expression, 269
 Couenne::exprGroup, 285
 Couenne::exprOp, 337
 Couenne::exprQuad, 355
 Couenne::exprUnary, 390
 Couenne::exprVar, 402

rank_
 Couenne::exprAux, 228

rcrop_
 Couenne::CouenneThreeWayBranchObj, 185

readOptimum
 Couenne::CouenneProblem, 142

readOptionsFile

Couenne::CouenneSetup, 167
realign
 Couenne::CouenneProblem, 146
 Couenne::exprCopy, 250
 Couenne::expression, 271
 Couenne::exprGroup, 286
 Couenne::exprOp, 337
 Couenne::exprQuad, 356
 Couenne::exprUnary, 391
recBSol
 Couenne::CouenneProblem, 152

redCostBT
 Couenne::CouenneProblem, 141

Reference
 Couenne::CouenneObject, 117

reference_
 Couenne::CouenneObject, 118
 Couenne::CouenneSOSBranchingObject, 174
 Couenne::CouenneSOSObject, 176

reformulate
 Couenne::CouenneProblem, 139

registerAllOptions
 Couenne::CouenneSetup, 167

registerOptions
 Couenne::CouenneAggrProbing, 44
 Couenne::CouenneAmplInterface, 47
 Couenne::CouenneChooseStrong, 59
 Couenne::CouenneChooseVariable, 62
 Couenne::CouenneCrossConv, 72
 Couenne::CouenneCutGenerator, 78
 Couenne::CouenneDisjCuts, 83
 Couenne::CouenneFeasPump, 92
 Couenne::CouenneFixPoint, 96
 Couenne::CouenneIterativeRounding, 108
 Couenne::CouenneOSInterface, 126
 Couenne::CouenneProblem, 142
 Couenne::CouenneSdpCuts, 165
 Couenne::CouenneSetup, 167
 Couenne::CouenneTwoImplied, 193
 Couenne::NlpSolveHeuristic, 418

replace
 Couenne::exprCopy, 250
 Couenne::expression, 270
 Couenne::exprGroup, 285
 Couenne::exprOp, 337
 Couenne::exprQuad, 355
 Couenne::exprUnary, 391

replaceIndex
 Couenne::DepGraph, 205
 Couenne::DepNode, 209

reset
 Couenne::CouenneSparseBndVec, 178

resetCutOff
 Couenne::CouenneProblem, 142

resetModel
 Couenne::CouenneFeasPump, 91
 Couenne::CouennelterativeRounding, 107
 Couenne::InitHeuristic, 407
 Couenne::NlpSolveHeuristic, 418
resize
 Couenne::CouenneSparseBndVec, 178
 Couenne::DomainPoint, 214
resolve
 Couenne::CouenneSolverInterface, 171
resolve_nobt
 Couenne::CouenneSolverInterface, 171
restoreCutoff_
 Couenne::CouenneAggrProbing, 45
restoreUnusedOriginals
 Couenne::CouenneProblem, 144
rootQ
 Couenne, 36
rootTime
 Couenne::CouenneCutGenerator, 78
rootTime_
 Couenne::CouenneCutGenerator, 79
row
 Couenne::CouenneSparseMatrix, 180
row_
 Couenne::CouenneExprMatrix, 88
STOP_AT_AUX
 Couenne, 31
SUM_INF
 Couenne, 31
SUM_NINF
 Couenne, 31
SAFE_COEFFICIENT
 CouenneExprDiv.hpp, 489
safe_pow
 Couenne, 35
safeDiv
 Couenne, 34
safeProd
 Couenne, 34
sdpCutGen_
 Couenne::CouenneProblem, 153
selectBranch
 Couenne::exprAbs, 220
 Couenne::exprBinProd, 231
 Couenne::exprCeil, 235
 Couenne::exprCopy, 250
 Couenne::exprCos, 255
 Couenne::exprDiv, 259
 Couenne::expression, 270
 Couenne::exprExp, 275
 Couenne::exprFloor, 279
 Couenne::exprInv, 292
 Couenne::exprLog, 312
 Couenne::exprMultiLin, 326
 Couenne::exprOddPow, 331
 Couenne::exprPow, 346
 Couenne::exprQuad, 355
 Couenne::exprSin, 360
 Couenne::exprTrilinear, 374
separateWithDisjunction
 Couenne::CouenneDisjCuts, 83
septime_
 Couenne::CouenneCutGenerator, 79
 Couenne::CouenneDisjCuts, 84
Set
 Couenne::CouenneFPpool, 98
set_
 Couenne::CouenneFPpool, 98
setAggressiveness
 Couenne::CouennelterativeRounding, 108
setAppDefaultOptions
 Couenne::CouenneInterface, 105
setBabPtr
 Couenne::CouenneCutGenerator, 77
setBase
 Couenne::CouenneProblem, 144
setBaseLbRhs
 Couenne::CouennelterativeRounding, 108
setCardSol
 Couenne::CouenneRecordBestSol, 158
setCheckAuxBounds
 Couenne::CouenneProblem, 144
setCouenneProblem
 Couenne::CouennelterativeRounding, 107
 Couenne::NlpSolveHeuristic, 418
setCutGenPtr
 Couenne::CouenneSolverInterface, 170
setCutOff
 Couenne::CouenneProblem, 142
 Couenne::GlobalCutOff, 405
setDoubleParameter
 Couenne::CouenneSetup, 167
setEstimate
 Couenne::CouenneObject, 117
setEstimates
 Couenne::CouenneObject, 117
setHasSol
 Couenne::CouenneRecordBestSol, 158
setInitDomLb
 Couenne::CouenneRecordBestSol, 158
setInitDomUb
 Couenne::CouenneRecordBestSol, 158
setInitIsInt
 Couenne::CouenneRecordBestSol, 158
setInitSol
 Couenne::CouenneMINLPInterface, 109

Couenne::CouenneTNLP, 187
setInteger
 Couenne::exprAux, 227
 Couenne::exprVar, 403
setJnlst
 Couenne::CouenneCutGenerator, 78
setLastPrioSort
 Couenne::CouenneProblem, 146
setLower
 Couenne::t_chg_bounds, 433
setLowerBits
 Couenne::t_chg_bounds, 433
setMaxCpuTime
 Couenne::CouenneProblem, 143
setMaxFailedSteps
 Couenne::CouenneAggrProbing, 44
setMaxFirPoints
 Couenne::CouennerterativeRounding, 108
setMaxNlplnf
 Couenne::NlpSolveHeuristic, 418
setMaxNodes
 Couenne::CouenneAggrProbing, 44
setMaxRoundingIter
 Couenne::CouennerterativeRounding, 108
setMaxTime
 Couenne::CouenneAggrProbing, 44
 Couenne::CouennerterativeRounding, 108
setMaxTimeFirstCall
 Couenne::CouennerterativeRounding, 108
setModSol
 Couenne::CouenneRecordBestSol, 159
setNDefVars
 Couenne::CouenneProblem, 137
setNlp
 Couenne::CouennerterativeRounding, 107
 Couenne::NlpSolveHeuristic, 418
setNodeComparisonMethod
 Couenne::CouenneSetup, 168
setNumberSolvePerLevel
 Couenne::CouenneFeasPump, 92
 Couenne::NlpSolveHeuristic, 418
setObj
 Couenne::CouenneMINLPInterface, 109
setObjective
 Couenne::CouenneProblem, 139
 Couenne::CouenneTNLP, 189
setOldBounds
 Couenne::CouenneBTPerfIndicator, 55
setOmega
 Couenne::CouennerterativeRounding, 108
setParameters
 Couenne::CouenneObject, 116
setProblem
 Couenne::CouenneBab, 49

 Couenne::CouenneCutGenerator, 76
setProblemName
 Couenne::CouenneProblem, 143
setRegisteredOptions
 Couenne::CouenneAmplInterface, 47
setRestoreCutoff
 Couenne::CouenneAggrProbing, 44
setSimulate
 Couenne::CouenneBranchingObject, 52
 Couenne::CouenneOrbitBranchingObj, 124
setSol
 Couenne::CouenneRecordBestSol, 158
setUp
 Couenne::t_chg_bounds, 433
setUpperBits
 Couenne::t_chg_bounds, 433
setVal
 Couenne::CouenneRecordBestSol, 159
setWriteAutoms
 Nauty, 415
setup
 Couenne::CouenneCrossConv, 72
setupJournals
 Couenne::CouenneUserInterface, 195
setupList
 Couenne::CouenneChooseStrong, 58
 Couenne::CouenneChooseVariable, 62
setupSymmetry
 Couenne::CouenneProblem, 137
shrink_arglist
 Couenne::exprOp, 336
shrankDoubleInf_
 Couenne::CouenneBTPerfIndicator, 56
shrankInf_
 Couenne::CouenneBTPerfIndicator, 55
sign
 Couenne::exprAux, 228
 Couenne::exprVar, 403
sign_
 Couenne::CouenneComplBranchingObject, 65
 Couenne::exprAux, 229
simpletriplet
 Couenne::simpletriplet, 428
simplify
 Couenne::exprAux, 226
 Couenne::exprBinProd, 230
 Couenne::exprCopy, 248
 Couenne::exprDiv, 258
 Couenne::expression, 267
 Couenne::exprGroup, 284
 Couenne::exprMax, 319
 Couenne::exprMin, 323
 Couenne::exprMultiLin, 325
 Couenne::exprOp, 336

Couenne::exprOpp, 341
Couenne::exprPow, 345
Couenne::exprQuad, 353
Couenne::exprSub, 366
Couenne::exprSum, 371
Couenne::exprUnary, 390
Couenne::exprVar, 401
simulate_
 Couenne::CouenneBranchingObject, 53
simulateBranch
 Couenne::CouenneChooseStrong, 59
sin
 Couenne, 33, 34
size
 Couenne::CouenneExprMatrix, 88
 Couenne::DomainPoint, 214
SmartAsl
 Couenne::SmartAsl, 430
sol
 Couenne::CouenneRecordBestSol, 160
solution
 Couenne::CouenneFeasPump, 91
 Couenne::CouenneInfo::NlpSolution, 416
 Couenne::CouenneIterativeRounding, 107
 Couenne::InitHeuristic, 407
 Couenne::NlpSolveHeuristic, 418
solve
 Couenne::CouenneMINLPInterface, 109
solveFromHotStart
 Couenne::CouenneSolverInterface, 171
solveMILP
 Couenne::CouenneFeasPump, 92
solveNLP
 Couenne::CouenneFeasPump, 92
Solver
 Couenne, 31
sparse2dense
 Couenne, 32
sparseQ
 Couenne::exprQuad, 352
sparseQcol
 Couenne::exprQuad, 352
splitAux
 Couenne::CouenneProblem, 143
standardize
 Couenne::CouenneConstraint, 70
 Couenne::CouenneObjective, 121
 Couenne::CouenneProblem, 140
 Couenne::CouennePSDcon, 155
 Couenne::exprBinProd, 231
 Couenne::exprCopy, 249
 Couenne::exprDiv, 259
 Couenne::expression, 268
 Couenne::exprMax, 319
 Couenne::exprMin, 323
 Couenne::exprMultiLin, 325
 Couenne::exprOddPow, 331
 Couenne::exprOp, 336
 Couenne::exprOpp, 341
 Couenne::exprPow, 346
 Couenne::exprSub, 366
 Couenne::exprSum, 371
 Couenne::exprUnary, 390
stats_
 Couenne::CouenneBTPerfIndicator, 56
Strategy
 Couenne::CouenneObject, 117
strategy_
 Couenne::CouenneObject, 118
sym_setup
 Couenne::CouenneProblem, 137
TAG_AND_RECURSIVE
 Couenne, 31
THREE_CENTER
 Couenne, 27
THREE_LEFT
 Couenne, 27
THREE_RAND
 Couenne, 27
THREE_RIGHT
 Couenne, 27
TWO_LEFT
 Couenne, 27
TWO_RAND
 Couenne, 27
TWO_RIGHT
 Couenne, 27
t_chg_bounds
 Couenne::t_chg_bounds, 432
THRES_ZERO_SYMM
 CouenneObject.hpp, 446
testIntFix
 Couenne::CouenneProblem, 146
tightenBounds
 Couenne::CouenneProblem, 141
 Couenne::CouenneSolverInterface, 171
tightenBoundsCLP
 Couenne::CouenneSolverInterface, 171
tightenBoundsCLP_Light
 Couenne::CouenneSolverInterface, 171
top_level
 Couenne::exprAux, 227
top_level_
 Couenne::exprAux, 228
totalInitTime_
 Couenne::CouenneTwoImplied, 194
totalTime_

Couenne::CouenneBTPerfIndicator, 56
Couenne::CouenneTwoImplied, 193
treeDecomp
 Couenne, 32
tri.bi
 Couenne, 32
trigImpliedBound
 Couenne, 35
trigNewton
 Couenne, 34
trigSelBranch
 Couenne, 35
trilinDecompType_
 Couenne::CouenneProblem, 152
Type
 Couenne::exprAux, 225
 Couenne::exprConst, 240
 Couenne::exprCopy, 246
 Couenne::expression, 266
 Couenne::exprLowerBound, 315
 Couenne::exprOp, 335
 Couenne::exprUnary, 388
 Couenne::exprUpperBound, 394
 Couenne::exprVar, 399

UNARY
 Couenne, 28
UNCHANGED
 Couenne::t_chg_bounds, 432
UNIFORM_GRID
 Couenne, 29
UNSET
 Couenne, 30
Ub
 Couenne::CouenneConstraint, 69
 Couenne::CouenneProblem, 138
 Couenne::exprAux, 225
 Couenne::exprVar, 399
ub
 Couenne::Domain, 212
 Couenne::DomainPoint, 215
 Couenne::exprVar, 399
ub._
 Couenne::CouenneConstraint, 70
 Couenne::DomainPoint, 216
 Couenne::exprAux, 228
unary_function
 Couenne, 27
unmarkHotStart
 Couenne::CouenneSolverInterface, 171
Unset
 Couenne::exprAux, 224

unsetWriteAutoms
 Nauty, 415
unusedOriginalsIndices
 Couenne::CouenneProblem, 144
unusedOriginalsIndices._
 Couenne::CouenneProblem, 151
upEstimate
 Couenne::CouenneObject, 117
upEstimate._
 Couenne::CouenneBranchingObject, 53
 Couenne::CouenneObject, 119
update
 Couenne::CouenneBTPerfIndicator, 55
 Couenne::CouenneRecordBestSol, 159
updateBound
 Couenne, 32
updateNLPObj
 Couenne::CouenneFeasPump, 92
updateSol
 Couenne::CouenneSdpCuts, 165
upper
 Couenne::t_chg_bounds, 432
useQuadratic_
 Couenne::CouenneProblem, 150
useSparsity_
 Couenne::CouenneSdpCuts, 165

VAR
 Couenne, 28
VAR_OBJ
 Couenne::CouenneObject, 115
VT_OBJ
 Couenne::CouenneObject, 115
val
 Couenne::CouenneRecordBestSol, 160
 Couenne::CouenneSparseMatrix, 180
Value
 Couenne::exprClone, 237
 Couenne::exprConst, 240
 Couenne::exprCopy, 247
 Couenne::expression, 266
value._
 Couenne::exprCopy, 251
 Couenne::exprStore, 363
Var
 Couenne::CouenneProblem, 137
varl
 Couenne::quadElem, 426
varIndex_
 Couenne::exprVar, 403
varIndices
 Couenne::CouenneExprMatrix, 88
varIndices._
 Couenne::CouenneExprMatrix, 88

varJ
 Couenne::quadElem, [426](#)

varSelection_
 Couenne::CouenneVarObject, [199](#)

VarStatus
 Nauty, [414](#)

variable
 Couenne::CouenneBranchingObject, [52](#)

variable2_
 Couenne::CouenneComplBranchingObject, [64](#)

variable_
 Couenne::CouenneBranchingObject, [53](#)

Variables
 Couenne::CouenneProblem, [137](#)

variables_
 Couenne::CouenneProblem, [148](#)

Vertices
 Couenne::DepGraph, [205](#)

vertices_
 Couenne::DepGraph, [206](#)

weightSum_
 Couenne::CouenneBTPerfIndicator, [56](#)

what_to_compare
 Couenne, [31](#)

WipeMakeInfeas
 CouenneInfeasCut.hpp, [436](#)

writeAMPL
 Couenne::CouenneProblem, [140](#)

writeGAMS
 Couenne::CouenneProblem, [141](#)

writeLP
 Couenne::CouenneProblem, [141](#)

writeSolution
 Couenne::CouenneAmplInterface, [47](#)
 Couenne::CouenneOSInterface, [126](#)
 Couenne::CouenneUserInterface, [196](#)

X
 Couenne::CouenneProblem, [138](#)

x
 Couenne::CouenneFPsolution, [100](#)
 Couenne::Domain, [212](#)
 Couenne::DomainPoint, [215](#)

X_
 Couenne::CouennePSDcon, [155](#)

x_
 Couenne::CouenneFPsolution, [100](#)
 Couenne::DomainPoint, [216](#)

ZERO
 Couenne, [28](#)

zero_fun
 Couenne, [33](#)

zeroMult