

CoinUtils

trunk

Generated by Doxygen 1.8.5

Mon Oct 21 2013 18:55:58

Contents

1	Todo List	1
2	Module Index	2
2.1	Modules	2
3	Namespace Index	2
3.1	Namespace List	2
4	Hierarchical Index	2
4.1	Class Hierarchy	2
5	Class Index	10
5.1	Class List	10
6	File Index	17
6.1	File List	17
7	Module Documentation	19
7.1	Presolve Matrix Manipulation Functions	19
7.1.1	Detailed Description	21
7.1.2	Function Documentation	21
7.2	Presolve Utility Functions	25
7.2.1	Detailed Description	25
7.2.2	Function Documentation	25
7.3	Presolve Debug Functions	26
7.3.1	Detailed Description	26
7.3.2	Function Documentation	27
8	Namespace Documentation	29
8.1	Coin Namespace Reference	29
8.1.1	Function Documentation	29
8.2	CoinParamUtils Namespace Reference	29
8.2.1	Detailed Description	30
8.2.2	Function Documentation	30
9	Class Documentation	32
9.1	_EKKfactinfo Struct Reference	32
9.1.1	Detailed Description	34
9.1.2	Member Data Documentation	34

9.2	doubleton_action::action Struct Reference	39
9.2.1	Detailed Description	39
9.2.2	Member Data Documentation	39
9.3	remove_fixed_action::action Struct Reference	40
9.3.1	Detailed Description	41
9.3.2	Member Data Documentation	41
9.4	forcing_constraint_action::action Struct Reference	41
9.4.1	Detailed Description	41
9.4.2	Member Data Documentation	42
9.5	tripleton_action::action Struct Reference	42
9.5.1	Detailed Description	43
9.5.2	Member Data Documentation	43
9.6	BitVector128 Class Reference	44
9.6.1	Detailed Description	44
9.6.2	Constructor & Destructor Documentation	44
9.6.3	Member Function Documentation	45
9.6.4	Friends And Related Function Documentation	45
9.7	CoinAbsFitEq Class Reference	45
9.7.1	Detailed Description	45
9.7.2	Constructor & Destructor Documentation	46
9.7.3	Member Function Documentation	46
9.8	CoinArbitraryArrayWithLength Class Reference	46
9.8.1	Detailed Description	47
9.8.2	Constructor & Destructor Documentation	48
9.8.3	Member Function Documentation	48
9.8.4	Member Data Documentation	49
9.9	CoinArrayWithLength Class Reference	49
9.9.1	Detailed Description	51
9.9.2	Constructor & Destructor Documentation	51
9.9.3	Member Function Documentation	52
9.9.4	Member Data Documentation	53
9.10	CoinBaseModel Class Reference	54
9.10.1	Detailed Description	56
9.10.2	Constructor & Destructor Documentation	56
9.10.3	Member Function Documentation	56
9.10.4	Member Data Documentation	58
9.11	CoinBigIndexArrayWithLength Class Reference	58

9.11.1 Detailed Description	59
9.11.2 Constructor & Destructor Documentation	60
9.11.3 Member Function Documentation	60
9.12 CoinBuild Class Reference	61
9.12.1 Detailed Description	62
9.12.2 Constructor & Destructor Documentation	62
9.12.3 Member Function Documentation	62
9.13 CoinDenseFactorization Class Reference	64
9.13.1 Detailed Description	66
9.13.2 Constructor & Destructor Documentation	66
9.13.3 Member Function Documentation	66
9.13.4 Friends And Related Function Documentation	69
9.14 CoinDenseVector< T > Class Template Reference	69
9.14.1 Detailed Description	70
9.14.2 Constructor & Destructor Documentation	71
9.14.3 Member Function Documentation	71
9.15 CoinDoubleArrayWithLength Class Reference	73
9.15.1 Detailed Description	74
9.15.2 Constructor & Destructor Documentation	74
9.15.3 Member Function Documentation	75
9.16 CoinError Class Reference	75
9.16.1 Detailed Description	76
9.16.2 Constructor & Destructor Documentation	77
9.16.3 Member Function Documentation	77
9.16.4 Friends And Related Function Documentation	78
9.16.5 Member Data Documentation	78
9.17 CoinExternalVectorFirstGreater_2< S, T, V > Class Template Reference	78
9.17.1 Detailed Description	78
9.17.2 Constructor & Destructor Documentation	78
9.17.3 Member Function Documentation	79
9.18 CoinExternalVectorFirstGreater_3< S, T, U, V > Class Template Reference	79
9.18.1 Detailed Description	79
9.18.2 Constructor & Destructor Documentation	79
9.18.3 Member Function Documentation	79
9.19 CoinExternalVectorFirstLess_2< S, T, V > Class Template Reference	80
9.19.1 Detailed Description	80
9.19.2 Constructor & Destructor Documentation	80

9.19.3	Member Function Documentation	80
9.20	CoinExternalVectorFirstLess_3< S, T, U, V > Class Template Reference	80
9.20.1	Detailed Description	81
9.20.2	Constructor & Destructor Documentation	81
9.20.3	Member Function Documentation	81
9.21	CoinFactorization Class Reference	81
9.21.1	Detailed Description	91
9.21.2	Constructor & Destructor Documentation	92
9.21.3	Member Function Documentation	92
9.21.4	Friends And Related Function Documentation	105
9.21.5	Member Data Documentation	105
9.22	CoinFactorizationDoubleArrayWithLength Class Reference	114
9.22.1	Detailed Description	115
9.22.2	Constructor & Destructor Documentation	115
9.22.3	Member Function Documentation	116
9.23	CoinFactorizationLongDoubleArrayWithLength Class Reference	117
9.23.1	Detailed Description	117
9.23.2	Constructor & Destructor Documentation	118
9.23.3	Member Function Documentation	118
9.24	CoinFileInput Class Reference	119
9.24.1	Detailed Description	120
9.24.2	Constructor & Destructor Documentation	120
9.24.3	Member Function Documentation	120
9.24.4	Friends And Related Function Documentation	121
9.25	CoinFileIOBase Class Reference	122
9.25.1	Detailed Description	122
9.25.2	Constructor & Destructor Documentation	122
9.25.3	Member Function Documentation	123
9.25.4	Member Data Documentation	123
9.26	CoinFileOutput Class Reference	123
9.26.1	Detailed Description	124
9.26.2	Member Enumeration Documentation	124
9.26.3	Constructor & Destructor Documentation	124
9.26.4	Member Function Documentation	124
9.27	CoinFirstAbsGreater_2< S, T > Class Template Reference	125
9.27.1	Detailed Description	126
9.27.2	Member Function Documentation	126

9.28	CoinFirstAbsGreater_3< S, T, U > Class Template Reference	126
9.28.1	Detailed Description	126
9.28.2	Member Function Documentation	126
9.29	CoinFirstAbsLess_2< S, T > Class Template Reference	127
9.29.1	Detailed Description	127
9.29.2	Member Function Documentation	127
9.30	CoinFirstAbsLess_3< S, T, U > Class Template Reference	127
9.30.1	Detailed Description	128
9.30.2	Member Function Documentation	128
9.31	CoinFirstGreater_2< S, T > Class Template Reference	128
9.31.1	Detailed Description	128
9.31.2	Member Function Documentation	128
9.32	CoinFirstGreater_3< S, T, U > Class Template Reference	129
9.32.1	Detailed Description	129
9.32.2	Member Function Documentation	129
9.33	CoinFirstLess_2< S, T > Class Template Reference	129
9.33.1	Detailed Description	130
9.33.2	Member Function Documentation	130
9.34	CoinFirstLess_3< S, T, U > Class Template Reference	130
9.34.1	Detailed Description	130
9.34.2	Member Function Documentation	130
9.35	CoinLpIO::CoinHashLink Struct Reference	131
9.35.1	Detailed Description	131
9.35.2	Member Data Documentation	131
9.36	CoinMpsIO::CoinHashLink Struct Reference	131
9.36.1	Detailed Description	131
9.36.2	Member Data Documentation	132
9.37	CoinIndexedVector Class Reference	132
9.37.1	Detailed Description	136
9.37.2	Constructor & Destructor Documentation	137
9.37.3	Member Function Documentation	138
9.37.4	Friends And Related Function Documentation	144
9.37.5	Member Data Documentation	145
9.38	CoinIntArrayWithLength Class Reference	145
9.38.1	Detailed Description	146
9.38.2	Constructor & Destructor Documentation	146
9.38.3	Member Function Documentation	147

9.39 CoinLpIO Class Reference	147
9.39.1 Detailed Description	153
9.39.2 Constructor & Destructor Documentation	155
9.39.3 Member Function Documentation	155
9.39.4 Friends And Related Function Documentation	164
9.39.5 Member Data Documentation	164
9.40 CoinMessage Class Reference	167
9.40.1 Detailed Description	168
9.40.2 Constructor & Destructor Documentation	168
9.41 CoinMessageHandler Class Reference	168
9.41.1 Detailed Description	171
9.41.2 Constructor & Destructor Documentation	173
9.41.3 Member Function Documentation	173
9.41.4 Friends And Related Function Documentation	177
9.41.5 Member Data Documentation	177
9.42 CoinMessages Class Reference	179
9.42.1 Detailed Description	180
9.42.2 Member Enumeration Documentation	181
9.42.3 Constructor & Destructor Documentation	181
9.42.4 Member Function Documentation	181
9.42.5 Member Data Documentation	182
9.43 CoinModel Class Reference	183
9.43.1 Detailed Description	190
9.43.2 Constructor & Destructor Documentation	191
9.43.3 Member Function Documentation	191
9.44 CoinModelHash Class Reference	207
9.44.1 Detailed Description	208
9.44.2 Constructor & Destructor Documentation	208
9.44.3 Member Function Documentation	208
9.45 CoinModelHash2 Class Reference	210
9.45.1 Detailed Description	210
9.45.2 Constructor & Destructor Documentation	210
9.45.3 Member Function Documentation	211
9.46 CoinModelHashLink Struct Reference	211
9.46.1 Detailed Description	212
9.46.2 Member Data Documentation	212
9.47 CoinModelInfo2 Struct Reference	212

9.47.1 Detailed Description	212
9.47.2 Constructor & Destructor Documentation	213
9.47.3 Member Data Documentation	213
9.48 CoinModelLink Class Reference	213
9.48.1 Detailed Description	214
9.48.2 Constructor & Destructor Documentation	214
9.48.3 Member Function Documentation	215
9.49 CoinModelLinkedList Class Reference	216
9.49.1 Detailed Description	217
9.49.2 Constructor & Destructor Documentation	217
9.49.3 Member Function Documentation	218
9.50 CoinModelTriple Struct Reference	220
9.50.1 Detailed Description	220
9.50.2 Member Data Documentation	220
9.51 CoinMpsCardReader Class Reference	221
9.51.1 Detailed Description	223
9.51.2 Constructor & Destructor Documentation	223
9.51.3 Member Function Documentation	223
9.51.4 Member Data Documentation	225
9.52 CoinMpsIO Class Reference	227
9.52.1 Detailed Description	234
9.52.2 Constructor & Destructor Documentation	234
9.52.3 Member Function Documentation	234
9.52.4 Friends And Related Function Documentation	243
9.52.5 Member Data Documentation	243
9.53 CoinOneMessage Class Reference	247
9.53.1 Detailed Description	248
9.53.2 Constructor & Destructor Documentation	248
9.53.3 Member Function Documentation	248
9.53.4 Member Data Documentation	249
9.54 CoinOslFactorization Class Reference	250
9.54.1 Detailed Description	252
9.54.2 Constructor & Destructor Documentation	252
9.54.3 Member Function Documentation	253
9.54.4 Friends And Related Function Documentation	256
9.54.5 Member Data Documentation	256
9.55 CoinOtherFactorization Class Reference	257

9.55.1 Detailed Description	260
9.55.2 Constructor & Destructor Documentation	260
9.55.3 Member Function Documentation	260
9.55.4 Member Data Documentation	265
9.56 CoinPackedMatrix Class Reference	267
9.56.1 Detailed Description	273
9.56.2 Constructor & Destructor Documentation	274
9.56.3 Member Function Documentation	275
9.56.4 Friends And Related Function Documentation	287
9.56.5 Member Data Documentation	287
9.57 CoinPackedVector Class Reference	288
9.57.1 Detailed Description	291
9.57.2 Constructor & Destructor Documentation	292
9.57.3 Member Function Documentation	293
9.57.4 Friends And Related Function Documentation	296
9.58 CoinPackedVectorBase Class Reference	297
9.58.1 Detailed Description	299
9.58.2 Constructor & Destructor Documentation	299
9.58.3 Member Function Documentation	299
9.59 CoinPair< S, T > Struct Template Reference	302
9.59.1 Detailed Description	302
9.59.2 Constructor & Destructor Documentation	302
9.59.3 Member Data Documentation	303
9.60 CoinParam Class Reference	303
9.60.1 Detailed Description	306
9.60.2 Member Typedef Documentation	307
9.60.3 Member Enumeration Documentation	307
9.60.4 Constructor & Destructor Documentation	307
9.60.5 Member Function Documentation	308
9.60.6 Friends And Related Function Documentation	311
9.61 CoinPartitionedVector Class Reference	314
9.61.1 Detailed Description	316
9.61.2 Constructor & Destructor Documentation	316
9.61.3 Member Function Documentation	316
9.61.4 Member Data Documentation	318
9.62 CoinPostsolveMatrix Class Reference	318
9.62.1 Detailed Description	320

9.62.2	Constructor & Destructor Documentation	321
9.62.3	Member Function Documentation	321
9.62.4	Member Data Documentation	321
9.63	CoinPrePostsolveMatrix Class Reference	322
9.63.1	Detailed Description	327
9.63.2	Member Enumeration Documentation	328
9.63.3	Constructor & Destructor Documentation	328
9.63.4	Member Function Documentation	329
9.63.5	Friends And Related Function Documentation	333
9.63.6	Member Data Documentation	333
9.64	CoinPresolveAction Class Reference	337
9.64.1	Detailed Description	338
9.64.2	Constructor & Destructor Documentation	339
9.64.3	Member Function Documentation	339
9.64.4	Member Data Documentation	340
9.65	CoinPresolveMatrix Class Reference	340
9.65.1	Detailed Description	345
9.65.2	Constructor & Destructor Documentation	346
9.65.3	Member Function Documentation	346
9.65.4	Friends And Related Function Documentation	352
9.65.5	Member Data Documentation	352
9.66	CoinPresolveMonitor Class Reference	357
9.66.1	Detailed Description	357
9.66.2	Constructor & Destructor Documentation	357
9.66.3	Member Function Documentation	358
9.67	CoinRelFltEq Class Reference	358
9.67.1	Detailed Description	358
9.67.2	Constructor & Destructor Documentation	359
9.67.3	Member Function Documentation	359
9.68	CoinSearchTree< Comp > Class Template Reference	359
9.68.1	Detailed Description	360
9.68.2	Constructor & Destructor Documentation	360
9.68.3	Member Function Documentation	360
9.69	CoinSearchTreeBase Class Reference	361
9.69.1	Detailed Description	362
9.69.2	Constructor & Destructor Documentation	362
9.69.3	Member Function Documentation	362

9.69.4	Member Data Documentation	363
9.70	CoinSearchTreeCompareBest Struct Reference	363
9.70.1	Detailed Description	364
9.70.2	Member Function Documentation	364
9.71	CoinSearchTreeCompareBreadth Struct Reference	364
9.71.1	Detailed Description	364
9.71.2	Member Function Documentation	364
9.72	CoinSearchTreeCompareDepth Struct Reference	365
9.72.1	Detailed Description	365
9.72.2	Member Function Documentation	365
9.73	CoinSearchTreeComparePreferred Struct Reference	365
9.73.1	Detailed Description	366
9.73.2	Member Function Documentation	366
9.74	CoinSearchTreeManager Class Reference	366
9.74.1	Detailed Description	366
9.74.2	Constructor & Destructor Documentation	367
9.74.3	Member Function Documentation	367
9.75	CoinSet Class Reference	368
9.75.1	Detailed Description	369
9.75.2	Constructor & Destructor Documentation	369
9.75.3	Member Function Documentation	369
9.75.4	Member Data Documentation	370
9.76	CoinShallowPackedVector Class Reference	370
9.76.1	Detailed Description	371
9.76.2	Constructor & Destructor Documentation	372
9.76.3	Member Function Documentation	373
9.76.4	Friends And Related Function Documentation	373
9.77	CoinSimpFactorization Class Reference	374
9.77.1	Detailed Description	380
9.77.2	Constructor & Destructor Documentation	380
9.77.3	Member Function Documentation	380
9.77.4	Friends And Related Function Documentation	386
9.77.5	Member Data Documentation	386
9.78	CoinSnapshot Class Reference	393
9.78.1	Detailed Description	396
9.78.2	Constructor & Destructor Documentation	396
9.78.3	Member Function Documentation	397

9.79	CoinSosSet Class Reference	403
9.79.1	Detailed Description	403
9.79.2	Constructor & Destructor Documentation	404
9.80	CoinStructuredModel Class Reference	404
9.80.1	Detailed Description	406
9.80.2	Constructor & Destructor Documentation	406
9.80.3	Member Function Documentation	406
9.81	CoinThreadRandom Class Reference	410
9.81.1	Detailed Description	410
9.81.2	Constructor & Destructor Documentation	411
9.81.3	Member Function Documentation	411
9.81.4	Member Data Documentation	411
9.82	CoinTimer Class Reference	412
9.82.1	Detailed Description	412
9.82.2	Constructor & Destructor Documentation	413
9.82.3	Member Function Documentation	413
9.83	CoinTreeNode Class Reference	414
9.83.1	Detailed Description	414
9.83.2	Constructor & Destructor Documentation	415
9.83.3	Member Function Documentation	415
9.84	CoinTreeSiblings Class Reference	416
9.84.1	Detailed Description	416
9.84.2	Constructor & Destructor Documentation	416
9.84.3	Member Function Documentation	416
9.85	CoinTriple< S, T, U > Class Template Reference	417
9.85.1	Detailed Description	417
9.85.2	Constructor & Destructor Documentation	417
9.85.3	Member Data Documentation	418
9.86	CoinUnsignedIntArrayWithLength Class Reference	418
9.86.1	Detailed Description	419
9.86.2	Constructor & Destructor Documentation	419
9.86.3	Member Function Documentation	420
9.87	CoinVoidStarArrayWithLength Class Reference	420
9.87.1	Detailed Description	421
9.87.2	Constructor & Destructor Documentation	421
9.87.3	Member Function Documentation	422
9.88	CoinWarmStart Class Reference	422

9.88.1 Detailed Description	423
9.88.2 Constructor & Destructor Documentation	423
9.88.3 Member Function Documentation	423
9.89 CoinWarmStartBasis Class Reference	424
9.89.1 Detailed Description	426
9.89.2 Member Typedef Documentation	426
9.89.3 Member Enumeration Documentation	427
9.89.4 Constructor & Destructor Documentation	427
9.89.5 Member Function Documentation	428
9.89.6 Friends And Related Function Documentation	431
9.89.7 Member Data Documentation	431
9.90 CoinWarmStartBasisDiff Class Reference	432
9.90.1 Detailed Description	432
9.90.2 Constructor & Destructor Documentation	433
9.90.3 Member Function Documentation	433
9.90.4 Friends And Related Function Documentation	434
9.91 CoinWarmStartDiff Class Reference	434
9.91.1 Detailed Description	434
9.91.2 Constructor & Destructor Documentation	434
9.91.3 Member Function Documentation	435
9.92 CoinWarmStartDual Class Reference	435
9.92.1 Detailed Description	436
9.92.2 Constructor & Destructor Documentation	436
9.92.3 Member Function Documentation	436
9.93 CoinWarmStartDualDiff Class Reference	437
9.93.1 Detailed Description	438
9.93.2 Constructor & Destructor Documentation	438
9.93.3 Member Function Documentation	438
9.93.4 Friends And Related Function Documentation	439
9.94 CoinWarmStartPrimalDual Class Reference	439
9.94.1 Detailed Description	440
9.94.2 Constructor & Destructor Documentation	440
9.94.3 Member Function Documentation	440
9.95 CoinWarmStartPrimalDualDiff Class Reference	442
9.95.1 Detailed Description	442
9.95.2 Constructor & Destructor Documentation	443
9.95.3 Member Function Documentation	443

9.95.4 Friends And Related Function Documentation	443
9.96 CoinWarmStartVector< T > Class Template Reference	444
9.96.1 Detailed Description	445
9.96.2 Constructor & Destructor Documentation	445
9.96.3 Member Function Documentation	445
9.97 CoinWarmStartVectorDiff< T > Class Template Reference	446
9.97.1 Detailed Description	447
9.97.2 Constructor & Destructor Documentation	448
9.97.3 Member Function Documentation	448
9.97.4 Friends And Related Function Documentation	449
9.98 CoinWarmStartVectorPair< T, U > Class Template Reference	449
9.98.1 Detailed Description	449
9.98.2 Constructor & Destructor Documentation	450
9.98.3 Member Function Documentation	450
9.99 CoinWarmStartVectorPairDiff< T, U > Class Template Reference	451
9.99.1 Detailed Description	452
9.99.2 Constructor & Destructor Documentation	452
9.99.3 Member Function Documentation	452
9.99.4 Friends And Related Function Documentation	453
9.100CoinYacc Class Reference	453
9.100.1 Detailed Description	453
9.100.2 Constructor & Destructor Documentation	453
9.100.3 Member Data Documentation	453
9.101do_tighten_action Class Reference	454
9.101.1 Detailed Description	454
9.101.2 Constructor & Destructor Documentation	454
9.101.3 Member Function Documentation	454
9.102doubleton_action Class Reference	455
9.102.1 Detailed Description	456
9.102.2 Constructor & Destructor Documentation	456
9.102.3 Member Function Documentation	456
9.102.4 Member Data Documentation	456
9.103drop_empty_cols_action Class Reference	456
9.103.1 Detailed Description	457
9.103.2 Constructor & Destructor Documentation	457
9.103.3 Member Function Documentation	458
9.104drop_empty_rows_action Class Reference	458

9.104.1 Detailed Description	459
9.104.2 Constructor & Destructor Documentation	459
9.104.3 Member Function Documentation	459
9.105drop_zero_coefficients_action Class Reference	459
9.105.1 Detailed Description	460
9.105.2 Constructor & Destructor Documentation	460
9.105.3 Member Function Documentation	460
9.106dropped_zero Struct Reference	461
9.106.1 Detailed Description	461
9.106.2 Member Data Documentation	461
9.107dupcol_action Class Reference	461
9.107.1 Detailed Description	462
9.107.2 Constructor & Destructor Documentation	462
9.107.3 Member Function Documentation	462
9.108duprow_action Class Reference	463
9.108.1 Detailed Description	463
9.108.2 Member Function Documentation	463
9.109EKKHlink Struct Reference	464
9.109.1 Detailed Description	464
9.109.2 Member Data Documentation	464
9.110FactorPointers Class Reference	464
9.110.1 Detailed Description	465
9.110.2 Constructor & Destructor Documentation	465
9.110.3 Member Data Documentation	465
9.111forcing_constraint_action Class Reference	466
9.111.1 Detailed Description	466
9.111.2 Constructor & Destructor Documentation	467
9.111.3 Member Function Documentation	467
9.112gubrow_action Class Reference	467
9.112.1 Detailed Description	468
9.112.2 Member Function Documentation	468
9.113implied_free_action Class Reference	468
9.113.1 Detailed Description	469
9.113.2 Constructor & Destructor Documentation	469
9.113.3 Member Function Documentation	469
9.114isolated_constraint_action Class Reference	470
9.114.1 Detailed Description	470

9.114.2 Constructor & Destructor Documentation	470
9.114.3 Member Function Documentation	470
9.115make_fixed_action Class Reference	471
9.115.1 Detailed Description	471
9.115.2 Constructor & Destructor Documentation	472
9.115.3 Member Function Documentation	472
9.115.4 Friends And Related Function Documentation	472
9.116presolvehlink Class Reference	473
9.116.1 Detailed Description	473
9.116.2 Friends And Related Function Documentation	473
9.116.3 Member Data Documentation	474
9.117Coin::ReferencedObject Class Reference	474
9.117.1 Detailed Description	474
9.117.2 Constructor & Destructor Documentation	476
9.117.3 Member Function Documentation	476
9.118remove_dual_action Class Reference	477
9.118.1 Detailed Description	477
9.118.2 Constructor & Destructor Documentation	478
9.118.3 Member Function Documentation	478
9.119remove_fixed_action Class Reference	478
9.119.1 Detailed Description	479
9.119.2 Constructor & Destructor Documentation	480
9.119.3 Member Function Documentation	480
9.119.4 Friends And Related Function Documentation	480
9.119.5 Member Data Documentation	480
9.120slack_doubleton_action Class Reference	481
9.120.1 Detailed Description	481
9.120.2 Constructor & Destructor Documentation	482
9.120.3 Member Function Documentation	482
9.121slack_singleton_action Class Reference	482
9.121.1 Detailed Description	483
9.121.2 Constructor & Destructor Documentation	483
9.121.3 Member Function Documentation	483
9.122Coin::SmartPtr< T > Class Template Reference	483
9.122.1 Detailed Description	485
9.122.2 Constructor & Destructor Documentation	486
9.122.3 Member Function Documentation	487

9.122.4 Friends And Related Function Documentation	488
9.123subst_constraint_action Class Reference	488
9.123.1 Detailed Description	489
9.123.2 Constructor & Destructor Documentation	489
9.123.3 Member Function Documentation	489
9.124symrec Struct Reference	490
9.124.1 Detailed Description	490
9.124.2 Member Data Documentation	490
9.125tripleton_action Class Reference	491
9.125.1 Detailed Description	492
9.125.2 Constructor & Destructor Documentation	492
9.125.3 Member Function Documentation	492
9.125.4 Member Data Documentation	492
9.126twoxtwo_action Class Reference	492
9.126.1 Detailed Description	493
9.126.2 Constructor & Destructor Documentation	493
9.126.3 Member Function Documentation	493
9.127useless_constraint_action Class Reference	494
9.127.1 Detailed Description	494
9.127.2 Constructor & Destructor Documentation	494
9.127.3 Member Function Documentation	495
9.127.4 Friends And Related Function Documentation	495
10 File Documentation	495
10.1 /home/ted/COIN/trunk/CoinUtils/src/Coin_C_defines.h File Reference	495
10.1.1 Macro Definition Documentation	496
10.1.2 Typedef Documentation	496
10.1.3 Function Documentation	497
10.2 /home/ted/COIN/trunk/CoinUtils/src/CoinAlloc.hpp File Reference	497
10.2.1 Macro Definition Documentation	497
10.3 /home/ted/COIN/trunk/CoinUtils/src/CoinBuild.hpp File Reference	498
10.4 /home/ted/COIN/trunk/CoinUtils/src/CoinDenseFactorization.hpp File Reference	498
10.5 /home/ted/COIN/trunk/CoinUtils/src/CoinDenseVector.hpp File Reference	498
10.5.1 Function Documentation	499
10.6 /home/ted/COIN/trunk/CoinUtils/src/CoinDistance.hpp File Reference	501
10.6.1 Function Documentation	501
10.7 /home/ted/COIN/trunk/CoinUtils/src/CoinError.hpp File Reference	502

10.7.1 Macro Definition Documentation	502
10.7.2 Function Documentation	503
10.8 /home/ted/COIN/trunk/CoinUtils/src/CoinFactorization.hpp File Reference	503
10.8.1 Macro Definition Documentation	503
10.9 /home/ted/COIN/trunk/CoinUtils/src/CoinFileIO.hpp File Reference	504
10.10/home/ted/COIN/trunk/CoinUtils/src/CoinFinite.hpp File Reference	504
10.10.1 Function Documentation	504
10.10.2 Variable Documentation	505
10.11/home/ted/COIN/trunk/CoinUtils/src/CoinFloatEqual.hpp File Reference	505
10.11.1 Detailed Description	505
10.12/home/ted/COIN/trunk/CoinUtils/src/CoinHelperFunctions.hpp File Reference	506
10.12.1 Macro Definition Documentation	508
10.12.2 Function Documentation	508
10.13/home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp File Reference	513
10.13.1 Macro Definition Documentation	514
10.13.2 Function Documentation	514
10.14/home/ted/COIN/trunk/CoinUtils/src/CoinLpIO.hpp File Reference	514
10.14.1 Typedef Documentation	514
10.14.2 Function Documentation	515
10.15/home/ted/COIN/trunk/CoinUtils/src/CoinMessage.hpp File Reference	515
10.15.1 Detailed Description	515
10.15.2 Enumeration Type Documentation	515
10.16/home/ted/COIN/trunk/CoinUtils/src/CoinMessageHandler.hpp File Reference	516
10.16.1 Detailed Description	517
10.16.2 Macro Definition Documentation	517
10.16.3 Enumeration Type Documentation	518
10.16.4 Function Documentation	518
10.17/home/ted/COIN/trunk/CoinUtils/src/CoinModel.hpp File Reference	518
10.17.1 Function Documentation	519
10.18/home/ted/COIN/trunk/CoinUtils/src/CoinModelUseful.hpp File Reference	519
10.18.1 Typedef Documentation	520
10.18.2 Function Documentation	520
10.19/home/ted/COIN/trunk/CoinUtils/src/CoinMpsIO.hpp File Reference	520
10.19.1 Macro Definition Documentation	521
10.19.2 Typedef Documentation	521
10.19.3 Enumeration Type Documentation	522
10.19.4 Function Documentation	523

10.20/home/ted/COIN/trunk/CoinUtils/src/CoinOslC.h File Reference	523
10.20.1 Macro Definition Documentation	525
10.20.2 Function Documentation	526
10.21/home/ted/COIN/trunk/CoinUtils/src/CoinOslFactorization.hpp File Reference	527
10.21.1 Typedef Documentation	528
10.22/home/ted/COIN/trunk/CoinUtils/src/CoinPackedMatrix.hpp File Reference	528
10.22.1 Function Documentation	528
10.23/home/ted/COIN/trunk/CoinUtils/src/CoinPackedVector.hpp File Reference	528
10.23.1 Macro Definition Documentation	530
10.23.2 Function Documentation	530
10.24/home/ted/COIN/trunk/CoinUtils/src/CoinPackedVectorBase.hpp File Reference	532
10.25/home/ted/COIN/trunk/CoinUtils/src/CoinParam.hpp File Reference	532
10.25.1 Detailed Description	534
10.25.2 Function Documentation	534
10.26/home/ted/COIN/trunk/CoinUtils/src/CoinPragma.hpp File Reference	534
10.27/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDoubleton.hpp File Reference	534
10.27.1 Macro Definition Documentation	534
10.28/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDual.hpp File Reference	534
10.29/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDupcol.hpp File Reference	534
10.29.1 Macro Definition Documentation	535
10.30/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveEmpty.hpp File Reference	535
10.30.1 Detailed Description	535
10.30.2 Variable Documentation	535
10.31/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveFixed.hpp File Reference	536
10.31.1 Macro Definition Documentation	536
10.32/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveForcing.hpp File Reference	536
10.32.1 Macro Definition Documentation	536
10.33/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveImpliedFree.hpp File Reference	537
10.33.1 Macro Definition Documentation	537
10.34/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveIsolated.hpp File Reference	537
10.35/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMatrix.hpp File Reference	537
10.35.1 Detailed Description	538
10.35.2 Macro Definition Documentation	538
10.35.3 Function Documentation	539
10.35.4 Variable Documentation	539
10.36/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMonitor.hpp File Reference	540
10.37/home/ted/COIN/trunk/CoinUtils/src/CoinPresolvePsdebug.hpp File Reference	540

10.38/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSingleton.hpp File Reference	540
10.38.1 Macro Definition Documentation	540
10.39/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSubst.hpp File Reference	540
10.39.1 Macro Definition Documentation	541
10.39.2 Function Documentation	541
10.40/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTighten.hpp File Reference	541
10.40.1 Macro Definition Documentation	541
10.40.2 Function Documentation	541
10.41/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTripleton.hpp File Reference	541
10.41.1 Macro Definition Documentation	542
10.42/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveUseless.hpp File Reference	542
10.42.1 Macro Definition Documentation	542
10.43/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveZeros.hpp File Reference	542
10.43.1 Detailed Description	543
10.43.2 Macro Definition Documentation	543
10.43.3 Function Documentation	543
10.44/home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp File Reference	543
10.44.1 Enumeration Type Documentation	544
10.44.2 Function Documentation	544
10.45/home/ted/COIN/trunk/CoinUtils/src/CoinShallowPackedVector.hpp File Reference	544
10.45.1 Function Documentation	544
10.46/home/ted/COIN/trunk/CoinUtils/src/CoinSignal.hpp File Reference	545
10.46.1 Typedef Documentation	545
10.47/home/ted/COIN/trunk/CoinUtils/src/CoinSimpFactorization.hpp File Reference	545
10.48/home/ted/COIN/trunk/CoinUtils/src/CoinSmartPtr.hpp File Reference	545
10.48.1 Macro Definition Documentation	546
10.48.2 Function Documentation	546
10.49/home/ted/COIN/trunk/CoinUtils/src/CoinSnapshot.hpp File Reference	547
10.50/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp File Reference	547
10.50.1 Typedef Documentation	549
10.50.2 Function Documentation	549
10.51/home/ted/COIN/trunk/CoinUtils/src/CoinStructuredModel.hpp File Reference	550
10.51.1 Typedef Documentation	550
10.52/home/ted/COIN/trunk/CoinUtils/src/CoinTime.hpp File Reference	550
10.52.1 Function Documentation	551
10.53/home/ted/COIN/trunk/CoinUtils/src/CoinTypes.hpp File Reference	551
10.53.1 Macro Definition Documentation	551

10.53.2 Typedef Documentation	552
10.54/home/ted/COIN/trunk/CoinUtils/src/CoinUtility.hpp File Reference	552
10.54.1 Function Documentation	552
10.55/home/ted/COIN/trunk/CoinUtils/src/CoinUtilsConfig.h File Reference	553
10.56/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart.hpp File Reference	553
10.56.1 Detailed Description	553
10.57/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartBasis.hpp File Reference	553
10.58/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartDual.hpp File Reference	554
10.59/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartPrimalDual.hpp File Reference	554
10.60/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartVector.hpp File Reference	554
10.61/home/ted/COIN/trunk/CoinUtils/src/config_coinutils_default.h File Reference	555
10.61.1 Macro Definition Documentation	555
10.62/home/ted/COIN/trunk/CoinUtils/src/config_default.h File Reference	555
10.62.1 Macro Definition Documentation	556

Index

557

1 Todo List

Member [CoinDrand48](#) (bool isSeed=false, unsigned int seed=1)

Anyone want to volunteer an upgrade for 64-bit architectures?

File [CoinMessageHandler.hpp](#)

This needs to be worked over for correct operation with ISO character codes.

Member [CoinMpsIO::dealWithFileName](#) (const char *filename, const char *extension, [CoinFileInput](#) *&input)

Add automatic append of .bz2 suffix when compiled with libbz.

Member [CoinMpsIO::readMps](#) ()

Provide an interface that will allow a client to associate a [CoinMpsCardReader](#) object with a [CoinMpsIO](#) object by setting the cardReader_ field.

Member [CoinPackedMatrix::appendMinorFast](#) (const int number, const [CoinBigIndex](#) *starts, const int *index, const double *element)

This method really belongs in the group of protected methods with #appendMinor; there are no safeties here even with COIN_DEBUG. Apparently this method was needed in ClpPackedMatrix and giving it proper visibility was too much trouble. Should be moved.

Class [CoinWarmStartBasis](#)

Modify this class so that the number of status entries per byte and bytes per status vector allocation unit are not hardcoded. At the least, collect this into a couple of macros.

Consider separate fields for allocated capacity and actual basis size. We could avoid some reallocation, at the price of retaining more space than we need. Perhaps more important, we could do much better sanity checks.

Class [CoinWarmStartBasisDiff](#)

This is a pretty generic structure, and vector diff is a pretty generic activity. We should be able to convert this to a template.

Using unsigned int as the data type for the diff vectors might help to contain the damage when this code is inevitably compiled for 64 bit architectures. But the notion of int as 4 bytes is hardwired into [CoinWarmStartBasis](#), so changes are definitely required.

Class [drop_empty_cols_action](#)

Confirm correct behaviour with solution in presolve.

Class [drop_empty_rows_action](#)

Confirm behaviour when a solution is present in presolve.

Class [dropped_zero](#)

Why isn't this a nested class in [drop_zero_coefficients_action](#)? That would match the structure of other presolve classes.

Group [Functions to work with variable status](#)

Why are we futzing around with three bit status? A holdover from the packed arrays of [CoinWarmStartBasis](#)? Big swaths of the presolve code manipulates colstat_ and rowstat_ as unsigned char arrays using simple assignment to set values.

Group [Methods for problem input and output](#)

Allow for file pointers and positioning

2 Module Index

2.1 Modules

Here is a list of all modules:

Presolve Matrix Manipulation Functions	19
Presolve Utility Functions	25
Presolve Debug Functions	26

3 Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Coin	29
CoinParamUtils	
Utility functions for processing CoinParam parameters	29

4 Hierarchical Index

4.1 Class Hierarchy

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

_EKKfactinfo	32
doubleton_action::action	39
remove_fixed_action::action	40
forcing_constraint_action::action	41
tripleton_action::action	42
std::allocator< T >	
std::array< T >	
std::auto_ptr< T >	
std::basic_string< Char >	
std::string	
std::wstring	
std::basic_string< char >	
std::basic_string< wchar_t >	
std::bitset< Bits >	
BitVector128	44
CoinAbsFltEq	45
CoinArrayWithLength	49
CoinArbitraryArrayWithLength	46
CoinBigIndexArrayWithLength	58
CoinDoubleArrayWithLength	73
CoinFactorizationDoubleArrayWithLength	114
CoinFactorizationLongDoubleArrayWithLength	117
CoinIntArrayWithLength	145
CoinUnsignedIntArrayWithLength	418
CoinVoidStarArrayWithLength	420
CoinBaseModel	54
CoinModel	183
CoinStructuredModel	404
CoinBuild	61
CoinDenseVector< T >	69
CoinError	75
CoinExternalVectorFirstGreater_2< S, T, V >	78
CoinExternalVectorFirstGreater_3< S, T, U, V >	79
CoinExternalVectorFirstLess_2< S, T, V >	80

CoinExternalVectorFirstLess_3< S, T, U, V >	80
CoinFactorization	81
CoinFileIOBase	122
CoinFileInput	119
CoinFileOutput	123
CoinFirstAbsGreater_2< S, T >	125
CoinFirstAbsGreater_3< S, T, U >	126
CoinFirstAbsLess_2< S, T >	127
CoinFirstAbsLess_3< S, T, U >	127
CoinFirstGreater_2< S, T >	128
CoinFirstGreater_3< S, T, U >	129
CoinFirstLess_2< S, T >	129
CoinFirstLess_3< S, T, U >	130
CoinLpIO::CoinHashLink	131
CoinMpsIO::CoinHashLink	131
CoinIndexedVector	132
CoinPartitionedVector	314
CoinLpIO	147
CoinMessageHandler	168
CoinMessages	179
CoinMessage	167
CoinModelHash	207
CoinModelHash2	210
CoinModelHashLink	211
CoinModelInfo2	212
CoinModelLink	213
CoinModelLinkedList	216
CoinModelTriple	220
CoinMpsCardReader	221
CoinMpsIO	227

CoinOneMessage	247
CoinOtherFactorization	257
CoinDenseFactorization	64
CoinOslFactorization	250
CoinSimpFactorization	374
CoinPackedMatrix	267
CoinPackedVectorBase	297
CoinPackedVector	288
CoinShallowPackedVector	370
CoinPair< S, T >	302
CoinParam	303
CoinPrePostsolveMatrix	322
CoinPostsolveMatrix	318
CoinPresolveMatrix	340
CoinPresolveAction	337
do_tighten_action	454
doubleton_action	455
drop_empty_cols_action	456
drop_empty_rows_action	458
drop_zero_coefficients_action	459
dupcol_action	461
duprow_action	463
forcing_constraint_action	466
gubrow_action	467
implied_free_action	468
isolated_constraint_action	470
make_fixed_action	471
remove_dual_action	477
remove_fixed_action	478
slack_doubleton_action	481

slack_singleton_action	482
subst_constraint_action	488
tripleton_action	491
twoxtwo_action	492
useless_constraint_action	494
CoinPresolveMonitor	357
CoinRelFltEq	358
CoinSearchTreeBase	361
CoinSearchTree< Comp >	359
CoinSearchTreeCompareBest	363
CoinSearchTreeCompareBreadth	364
CoinSearchTreeCompareDepth	365
CoinSearchTreeComparePreferred	365
CoinSearchTreeManager	366
CoinSet	368
CoinSosSet	403
CoinSnapshot	393
CoinThreadRandom	410
CoinTimer	412
CoinTreeNode	414
CoinTreeSiblings	416
CoinTriple< S, T, U >	417
CoinWarmStart	422
CoinWarmStartBasis	424
CoinWarmStartDual	435
CoinWarmStartPrimalDual	439
CoinWarmStartVector< T >	444
CoinWarmStartVector< double >	444
CoinWarmStartVector< U >	444
CoinWarmStartVectorPair< T, U >	449

CoinWarmStartDiff	434
CoinWarmStartBasisDiff	432
CoinWarmStartDualDiff	437
CoinWarmStartPrimalDualDiff	442
CoinWarmStartVectorDiff< T >	446
CoinWarmStartVectorDiff< double >	446
CoinWarmStartVectorDiff< U >	446
CoinWarmStartVectorPairDiff< T, U >	451
CoinYacc	453
std::complex	
std::list< T >::const_iterator	
std::forward_list< T >::const_iterator	
std::map< K, T >::const_iterator	
std::unordered_map< K, T >::const_iterator	
std::basic_string< Char >::const_iterator	
std::multimap< K, T >::const_iterator	
std::unordered_multimap< K, T >::const_iterator	
std::set< K >::const_iterator	
std::string::const_iterator	
std::unordered_set< K >::const_iterator	
std::wstring::const_iterator	
std::multiset< K >::const_iterator	
std::unordered_multiset< K >::const_iterator	
std::vector< T >::const_iterator	
std::deque< T >::const_iterator	
std::list< T >::const_reverse_iterator	
std::deque< T >::const_reverse_iterator	
std::unordered_map< K, T >::const_reverse_iterator	
std::multimap< K, T >::const_reverse_iterator	
std::basic_string< Char >::const_reverse_iterator	
std::unordered_multimap< K, T >::const_reverse_iterator	
std::forward_list< T >::const_reverse_iterator	
std::set< K >::const_reverse_iterator	
std::string::const_reverse_iterator	
std::map< K, T >::const_reverse_iterator	
std::unordered_set< K >::const_reverse_iterator	
std::multiset< K >::const_reverse_iterator	
std::wstring::const_reverse_iterator	
std::unordered_multiset< K >::const_reverse_iterator	
std::vector< T >::const_reverse_iterator	
std::deque< T >	
dropped_zero	461
EKKHlink	464
std::error_category	
std::error_code	
std::error_condition	

```

std::exception
std::bad_alloc
std::bad_cast
std::bad_exception
std::bad_typeid
std::ios_base::failure
std::logic_error
std::domain_error
std::invalid_argument
std::length_error
std::out_of_range
std::runtime_error
std::overflow_error
std::range_error
std::underflow_error

```

FactorPointers

464

```

std::forward_list< T >
std::ios_base
std::basic_ios< char >
std::basic_ios< wchar_t >
std::basic_ios
std::basic_istream< char >
std::basic_istream< wchar_t >
std::basic_ostream< char >
std::basic_ostream< wchar_t >
std::basic_istream
std::basic_ifstream< char >
std::basic_ifstream< wchar_t >
std::basic_iostream< char >
std::basic_iostream< wchar_t >
std::basic_istringstream< char >
std::basic_istringstream< wchar_t >
std::basic_ifstream
std::ifstream
std::wifstream
std::basic_iostream
std::basic_fstream< char >
std::basic_fstream< wchar_t >
std::basic_stringstream< char >
std::basic_stringstream< wchar_t >
std::basic_fstream
std::fstream
std::wfstream
std::basic_stringstream
std::stringstream
std::wstringstream
std::basic_istringstream
std::istringstream
std::wistringstream
std::istream
std::wistream
std::basic_ostream
std::basic_iostream< char >

```

```

    basic_iostream< wchar_t >
    basic_ofstream< char >
    basic_ofstream< wchar_t >
    basic_ostringstream< char >
    basic_ostringstream< wchar_t >
    std::basic_iostream
    std::basic_ofstream
        std::ofstream
        std::wofstream
    std::basic_ostringstream
        std::ostringstream
        std::wostringstream
    std::ostream
    std::wostream
    std::ios
    std::wios
    std::list< T >::iterator
    std::forward_list< T >::iterator
    std::deque< T >::iterator
    std::unordered_map< K, T >::iterator
    std::wstring::iterator
    std::basic_string< Char >::iterator
    std::unordered_multimap< K, T >::iterator
    std::string::iterator
    std::unordered_set< K >::iterator
    std::multimap< K, T >::iterator
    std::unordered_multiset< K >::iterator
    std::map< K, T >::iterator
    std::vector< T >::iterator
    std::multiset< K >::iterator
    std::set< K >::iterator
    std::list< T >
    std::map< K, T >
    std::multimap< K, T >
    std::multiset< K >

```

presolvehlink

473

```

std::priority_queue< T >
std::queue< T >

```

Coin::ReferencedObject

474

```

std::wstring::reverse_iterator
std::unordered_map< K, T >::reverse_iterator
std::forward_list< T >::reverse_iterator
std::set< K >::reverse_iterator
std::map< K, T >::reverse_iterator
std::multiset< K >::reverse_iterator
std::unordered_multiset< K >::reverse_iterator
std::deque< T >::reverse_iterator
std::list< T >::reverse_iterator
std::string::reverse_iterator
std::vector< T >::reverse_iterator
std::basic_string< Char >::reverse_iterator
std::unordered_multimap< K, T >::reverse_iterator

```

[std::multimap< K, T >::reverse_iterator](#)
[std::unordered_set< K >::reverse_iterator](#)
[std::set< K >](#)
[std::set< int >](#)
[std::smart_ptr< T >](#)

Coin::SmartPtr< T > [483](#)
[std::stack< T >](#)

symrec [490](#)

[std::system_error](#)
[std::thread](#)
[std::unique_ptr< T >](#)
[std::unordered_map< K, T >](#)
[std::unordered_multimap< K, T >](#)
[std::unordered_multiset< K >](#)
[std::unordered_set< K >](#)
[std::valarray< T >](#)
[std::vector< T >](#)
[std::vector< char >](#)
[std::vector< CoinTreeSiblings * >](#)
[std::vector< double >](#)
[std::vector< int >](#)
[std::vector< std::string >](#)
[std::weak_ptr< T >](#)
[bool](#)
[char](#)
[COINMpsType](#)
[CoinParamType](#)
[COINSectionType](#)
[CoinWarmStartVector< T >](#)
[CoinWarmStartVectorDiff< T >](#)
[Comp](#)
[const V *](#)
[double](#)
[FILE *](#)
[int](#)
[K](#)
[Language](#)
[S](#)
[size_t](#)
[T](#)
[T *](#)
[U](#)
[void *](#)

5 Class Index

5.1 Class List

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

[_EKKfactinfo](#) [32](#)

doubleton_action::action	39
remove_fixed_action::action	
Structure to hold information necessary to reintroduce a column into the problem representation	40
forcing_constraint_action::action	41
tripleton_action::action	42
BitVector128	44
CoinAbsFltEq	
Equality to an absolute tolerance	45
CoinArbitraryArrayWithLength	
Arbitrary version	46
CoinArrayWithLength	
Pointer with length in bytes	49
CoinBaseModel	54
CoinBigIndexArrayWithLength	
CoinBigIndex * version	58
CoinBuild	
In many cases it is natural to build a model by adding one row at a time	61
CoinDenseFactorization	
This deals with Factorization and Updates This is a simple dense version so other people can write a better one	64
CoinDenseVector< T >	
Dense Vector	69
CoinDoubleArrayWithLength	
Double * version	73
CoinError	
Error Class thrown by an exception	75
CoinExternalVectorFirstGreater_2< S, T, V >	
Function operator	78
CoinExternalVectorFirstGreater_3< S, T, U, V >	
Function operator	79
CoinExternalVectorFirstLess_2< S, T, V >	
Function operator	80
CoinExternalVectorFirstLess_3< S, T, U, V >	
Function operator	80
CoinFactorization	
This deals with Factorization and Updates	81

CoinFactorizationDoubleArrayWithLength	
CoinFactorizationDouble * version	114
CoinFactorizationLongDoubleArrayWithLength	
CoinFactorizationLongDouble * version	117
CoinFileInput	
Abstract base class for file input classes	119
CoinFileIOBase	
Base class for FileIO classes	122
CoinFileOutput	
Abstract base class for file output classes	123
CoinFirstAbsGreater_2< S, T >	
Function operator	125
CoinFirstAbsGreater_3< S, T, U >	
Function operator	126
CoinFirstAbsLess_2< S, T >	
Function operator	127
CoinFirstAbsLess_3< S, T, U >	
Function operator	127
CoinFirstGreater_2< S, T >	
Function operator	128
CoinFirstGreater_3< S, T, U >	
Function operator	129
CoinFirstLess_2< S, T >	
Function operator	129
CoinFirstLess_3< S, T, U >	
Function operator	130
CoinLpIO::CoinHashLink	131
CoinMpsIO::CoinHashLink	131
CoinIndexedVector	
Indexed Vector	132
CoinIntArrayWithLength	
Int * version	145
CoinLpIO	
Class to read and write Lp files	147
CoinMessage	
The standard set of Coin messages	167
CoinMessageHandler	
Base class for message handling	168

CoinMessages	
Class to hold and manipulate an array of massaged messages	179
CoinModel	
This is a simple minded model which is stored in a format which makes it easier to construct and modify but not efficient for algorithms	183
CoinModelHash	207
CoinModelHash2	
For int,int hashing	210
CoinModelHashLink	
For names and hashing	211
CoinModelInfo2	
This is a model which is made up of Coin(Structured)Model blocks	212
CoinModelLink	
This is for various structures/classes needed by CoinModel	213
CoinModelLinkedList	216
CoinModelTriple	
For linked lists	220
CoinMpsCardReader	
Very simple code for reading MPS data	221
CoinMpsIO	
MPS IO Interface	227
CoinOneMessage	
Class for one massaged message	247
CoinOslFactorization	250
CoinOtherFactorization	
Abstract base class which also has some scalars so can be used from Dense or Simp	257
CoinPackedMatrix	
Sparse Matrix Base Class	267
CoinPackedVector	
Sparse Vector	288
CoinPackedVectorBase	
Abstract base class for various sparse vectors	297
CoinPair< S, T >	
An ordered pair	302
CoinParam	
A base class for 'keyword value' command line parameters	303
CoinPartitionedVector	314

CoinPostsolveMatrix	
Augments CoinPrePostsolveMatrix with information about the problem that is only needed during postsolve	318
CoinPrePostsolveMatrix	
Collects all the information about the problem that is needed in both presolve and postsolve	322
CoinPresolveAction	
Abstract base class of all presolve routines	337
CoinPresolveMatrix	
Augments CoinPrePostsolveMatrix with information about the problem that is only needed during presolve	340
CoinPresolveMonitor	
Monitor a row or column for modification	357
CoinRelFitEq	
Equality to a scaled tolerance	358
CoinSearchTree< Comp >	359
CoinSearchTreeBase	361
CoinSearchTreeCompareBest	
Best first search	363
CoinSearchTreeCompareBreadth	364
CoinSearchTreeCompareDepth	
Depth First Search	365
CoinSearchTreeComparePreferred	
Function objects to compare search tree nodes	365
CoinSearchTreeManager	366
CoinSet	
Very simple class for containing data on set	368
CoinShallowPackedVector	
Shallow Sparse Vector	370
CoinSimpFactorization	374
CoinSnapshot	
NON Abstract Base Class for interfacing with cut generators or branching code or	393
CoinSosSet	
Very simple class for containing SOS set	403
CoinStructuredModel	404
CoinThreadRandom	
Class for thread specific random numbers	410

CoinTimer		
This class implements a timer that also implements a tracing functionality		412
CoinTreeNode		
A class from which the real tree nodes should be derived from		414
CoinTreeSiblings		416
CoinTriple< S, T, U >		417
CoinUnsignedIntArrayWithLength		
Unsigned int * version		418
CoinVoidStarArrayWithLength		
Void * version		420
CoinWarmStart		
Abstract base class for warm start information		422
CoinWarmStartBasis		
The default COIN simplex (basis-oriented) warm start class		424
CoinWarmStartBasisDiff		
A 'diff' between two CoinWarmStartBasis objects		432
CoinWarmStartDiff		
Abstract base class for warm start 'diff' objects		434
CoinWarmStartDual		
WarmStart information that is only a dual vector		435
CoinWarmStartDualDiff		
A 'diff' between two CoinWarmStartDual objects		437
CoinWarmStartPrimalDual		
WarmStart information that is only a dual vector		439
CoinWarmStartPrimalDualDiff		
A 'diff' between two CoinWarmStartPrimalDual objects		442
CoinWarmStartVector< T >		
WarmStart information that is only a vector		444
CoinWarmStartVectorDiff< T >		
A 'diff' between two CoinWarmStartVector objects		446
CoinWarmStartVectorPair< T, U >		449
CoinWarmStartVectorPairDiff< T, U >		451
CoinYacc		453
do_tighten_action		454
doubleton_action		
Solve $ax+by=c$ for y and substitute y out of the problem		455

drop_empty_cols_action	Physically removes empty columns in presolve, and reinserts empty columns in postsolve	456
drop_empty_rows_action	Physically removes empty rows in presolve, and reinserts empty rows in postsolve	458
drop_zero_coefficients_action	Removal of explicit zeros	459
dropped_zero	Tracking information for an explicit zero coefficient	461
dupcol_action	Detect and remove duplicate columns	461
duprow_action	Detect and remove duplicate rows	463
EKKHlink	This deals with Factorization and Updates This is ripped off from OSL!!!!!!!!!	464
FactorPointers	Pointers used during factorization	464
forcing_constraint_action	Detect and process forcing constraints and useless constraints	466
gubrow_action	Detect and remove entries whose sum is known	467
implied_free_action	Detect and process implied free variables	468
isolated_constraint_action		470
make_fixed_action	Fix a variable at a specified bound	471
presolvehlink	Links to aid in packed matrix modification	473
Coin::ReferencedObject	ReferencedObject class	474
remove_dual_action	Attempt to fix variables by bounding reduced costs	477
remove_fixed_action	Excise fixed variables from the model	478
slack_doubleton_action	Convert an explicit bound constraint to a column bound	481
slack_singleton_action	For variables with one entry	482

Coin::SmartPtr< T >	
Template class for Smart Pointers	483
subst_constraint_action	
Detect and process implied free variables	488
symrec	
For string evaluation	490
tripleton_action	
We are only going to do this if it does not increase number of elements?	491
twoxtwo_action	
Detect interesting 2 by 2 blocks	492
useless_constraint_action	494

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

/home/ted/COIN/trunk/CoinUtils/src/Coin_C_defines.h	495
/home/ted/COIN/trunk/CoinUtils/src/CoinAlloc.hpp	497
/home/ted/COIN/trunk/CoinUtils/src/CoinBuild.hpp	498
/home/ted/COIN/trunk/CoinUtils/src/CoinDenseFactorization.hpp	498
/home/ted/COIN/trunk/CoinUtils/src/CoinDenseVector.hpp	498
/home/ted/COIN/trunk/CoinUtils/src/CoinDistance.hpp	501
/home/ted/COIN/trunk/CoinUtils/src/CoinError.hpp	502
/home/ted/COIN/trunk/CoinUtils/src/CoinFactorization.hpp	503
/home/ted/COIN/trunk/CoinUtils/src/CoinFileIO.hpp	504
/home/ted/COIN/trunk/CoinUtils/src/CoinFinite.hpp	504
/home/ted/COIN/trunk/CoinUtils/src/CoinFloatEqual.hpp	
Function objects for testing equality of real numbers	505
/home/ted/COIN/trunk/CoinUtils/src/CoinHelperFunctions.hpp	506
/home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp	513
/home/ted/COIN/trunk/CoinUtils/src/CoinLpIO.hpp	514
/home/ted/COIN/trunk/CoinUtils/src/CoinMessage.hpp	
This file contains the enum for the standard set of Coin messages and a class definition whose sole purpose is to supply a constructor	515

<code>/home/ted/COIN/trunk/CoinUtils/src/CoinMessageHandler.hpp</code>	
This is a first attempt at a message handler	516
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinModel.hpp</code>	518
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinModelUseful.hpp</code>	519
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinMpsIO.hpp</code>	520
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinOslC.h</code>	523
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinOslFactorization.hpp</code>	527
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPackedMatrix.hpp</code>	528
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPackedVector.hpp</code>	528
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPackedVectorBase.hpp</code>	532
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinParam.hpp</code>	
Declaration of a class for command line parameters	532
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPragma.hpp</code>	534
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDoubleton.hpp</code>	534
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDual.hpp</code>	534
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDupcol.hpp</code>	534
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveEmpty.hpp</code>	
Drop/reinsert empty rows/columns	535
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveFixed.hpp</code>	536
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveForcing.hpp</code>	536
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveImpliedFree.hpp</code>	537
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveIsolated.hpp</code>	537
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMatrix.hpp</code>	
Declarations for <code>CoinPresolveMatrix</code> and <code>CoinPostsolveMatrix</code> and their common base class <code>Coin-PrePostsolveMatrix</code>	537
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMonitor.hpp</code>	540
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolvePsdebug.hpp</code>	540
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSingleton.hpp</code>	540
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSubst.hpp</code>	540
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTighten.hpp</code>	541
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTripleton.hpp</code>	541
<code>/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveUseless.hpp</code>	542

/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveZeros.hpp	
Drop/reintroduce explicit zeros	542
/home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp	543
/home/ted/COIN/trunk/CoinUtils/src/CoinShallowPackedVector.hpp	544
/home/ted/COIN/trunk/CoinUtils/src/CoinSignal.hpp	545
/home/ted/COIN/trunk/CoinUtils/src/CoinSimpFactorization.hpp	545
/home/ted/COIN/trunk/CoinUtils/src/CoinSmartPtr.hpp	545
/home/ted/COIN/trunk/CoinUtils/src/CoinSnapshot.hpp	547
/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp	547
/home/ted/COIN/trunk/CoinUtils/src/CoinStructuredModel.hpp	550
/home/ted/COIN/trunk/CoinUtils/src/CoinTime.hpp	550
/home/ted/COIN/trunk/CoinUtils/src/CoinTypes.hpp	551
/home/ted/COIN/trunk/CoinUtils/src/CoinUtility.hpp	552
/home/ted/COIN/trunk/CoinUtils/src/CoinUtilsConfig.h	553
/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart.hpp	
Copyright (C) 2000 – 2003, International Business Machines Corporation and others	553
/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartBasis.hpp	553
/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartDual.hpp	554
/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartPrimalDual.hpp	554
/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartVector.hpp	554
/home/ted/COIN/trunk/CoinUtils/src/config_coinutils_default.h	555
/home/ted/COIN/trunk/CoinUtils/src/config_default.h	555

7 Module Documentation

7.1 Presolve Matrix Manipulation Functions

Functions to work with the loosely packed and threaded packed matrix structures used during presolve and postsolve.

Functions

- `void presolve_make_memlists` (int *lengths, `presolvehlink` *link, int n)
Initialise linked list for major vector order in bulk storage.
- `bool presolve_expand_major` (`CoinBigIndex` *majstrts, double *majels, int *minndxs, int *majlens, `presolvehlink` *majlinks, int nmaj, int k)

Make sure a major-dimension vector `k` has room for one more coefficient.

- bool `presolve_expand_col` (CoinBigIndex *mcstrt, double *colels, int *hrow, int *hincol, [presolvehlink](#) *clink, int ncols, int colx)

Make sure a column (`colx`) in a column-major matrix has room for one more coefficient.

- bool `presolve_expand_row` (CoinBigIndex *mrstrt, double *rowels, int *hcol, int *hinrow, [presolvehlink](#) *rlink, int nrow, int rowx)

Make sure a row (`rowx`) in a row-major matrix has room for one more coefficient.

- CoinBigIndex `presolve_find_minor` (int tgt, CoinBigIndex ks, CoinBigIndex ke, const int *minndx)

Find position of a minor index in a major vector.

- CoinBigIndex `presolve_find_row` (int row, CoinBigIndex kcs, CoinBigIndex kce, const int *hrow)

Find position of a row in a column in a column-major matrix.

- CoinBigIndex `presolve_find_col` (int col, CoinBigIndex krs, CoinBigIndex kre, const int *hcol)

Find position of a column in a row in a row-major matrix.

- CoinBigIndex `presolve_find_minor1` (int tgt, CoinBigIndex ks, CoinBigIndex ke, const int *minndx)

Find position of a minor index in a major vector.

- CoinBigIndex `presolve_find_row1` (int row, CoinBigIndex kcs, CoinBigIndex kce, const int *hrow)

Find position of a row in a column in a column-major matrix.

- CoinBigIndex `presolve_find_col1` (int col, CoinBigIndex krs, CoinBigIndex kre, const int *hcol)

Find position of a column in a row in a row-major matrix.

- CoinBigIndex `presolve_find_minor2` (int tgt, CoinBigIndex ks, int majlen, const int *minndx, const CoinBigIndex *majlinks)

Find position of a minor index in a major vector in a threaded matrix.

- CoinBigIndex `presolve_find_row2` (int row, CoinBigIndex kcs, int collen, const int *hrow, const CoinBigIndex *clinks)

Find position of a row in a column in a column-major threaded matrix.

- CoinBigIndex `presolve_find_minor3` (int tgt, CoinBigIndex ks, int majlen, const int *minndx, const CoinBigIndex *majlinks)

Find position of a minor index in a major vector in a threaded matrix.

- CoinBigIndex `presolve_find_row3` (int row, CoinBigIndex kcs, int collen, const int *hrow, const CoinBigIndex *clinks)

Find position of a row in a column in a column-major threaded matrix.

- void `presolve_delete_from_major` (int majndx, int minndx, const CoinBigIndex *majstrts, int *majlens, int *minndx, double *els)

Delete the entry for a minor index from a major vector.

- void `presolve_delete_many_from_major` (int majndx, char *marked, const CoinBigIndex *majstrts, int *majlens, int *minndx, double *els)

Delete marked entries.

- void `presolve_delete_from_col` (int row, int col, const CoinBigIndex *mcstrt, int *hincol, int *hrow, double *colels)

Delete the entry for row `row` from column `col` in a column-major matrix.

- void `presolve_delete_from_row` (int row, int col, const CoinBigIndex *mrstrt, int *hinrow, int *hcol, double *rowels)

Delete the entry for column `col` from row `row` in a row-major matrix.

- void `presolve_delete_from_major2` (int majndx, int minndx, CoinBigIndex *majstrts, int *majlens, int *minndx, int *majlinks, CoinBigIndex *free_listp)

Delete the entry for a minor index from a major vector in a threaded matrix.

- void `presolve_delete_from_col2` (int row, int col, CoinBigIndex *mcstrt, int *hincol, int *hrow, int *clinks, CoinBigIndex *free_listp)

Delete the entry for row `row` from column `col` in a column-major threaded matrix.

7.1.1 Detailed Description

Functions to work with the loosely packed and threaded packed matrix structures used during presolve and postsolve.

7.1.2 Function Documentation

7.1.2.1 void presolve_make_memlists (int * *lengths*, presolvehlink * *link*, int *n*) [related]

Initialise linked list for major vector order in bulk storage.

7.1.2.2 bool presolve_expand_major (CoinBigIndex * *majstrts*, double * *majels*, int * *minndx*s, int * *majlens*, presolvehlink * *majlinks*, int *nmaj*, int *k*) [related]

Make sure a major-dimension vector *k* has room for one more coefficient.

You can use this directly, or use the inline wrappers `presolve_expand_col` and `presolve_expand_row`

7.1.2.3 bool presolve_expand_col (CoinBigIndex * *mcstrt*, double * *colels*, int * *hrow*, int * *hincol*, presolvehlink * *clink*, int *ncols*, int *colx*) [related]

Make sure a column (*colx*) in a column-major matrix has room for one more coefficient.

Definition at line 1559 of file `CoinPresolveMatrix.hpp`.

7.1.2.4 bool presolve_expand_row (CoinBigIndex * *mrstrt*, double * *rowels*, int * *hcol*, int * *hinrow*, presolvehlink * *rlink*, int *nrows*, int *rowx*) [related]

Make sure a row (*rowx*) in a row-major matrix has room for one more coefficient.

Definition at line 1570 of file `CoinPresolveMatrix.hpp`.

7.1.2.5 CoinBigIndex presolve_find_minor (int *tgt*, CoinBigIndex *ks*, CoinBigIndex *ke*, const int * *minndx*s) [related]

Find position of a minor index in a major vector.

The routine returns the position *k* in *minndx*s for the specified minor index *tgt*. It will abort if the entry does not exist. Can be used directly or via the inline wrappers `presolve_find_row` and `presolve_find_col`.

Definition at line 1585 of file `CoinPresolveMatrix.hpp`.

7.1.2.6 CoinBigIndex presolve_find_row (int *row*, CoinBigIndex *kcs*, CoinBigIndex *kce*, const int * *hrow*) [related]

Find position of a row in a column in a column-major matrix.

The routine returns the position *k* in *hrow* for the specified *row*. It will abort if the entry does not exist.

Definition at line 1609 of file `CoinPresolveMatrix.hpp`.

7.1.2.7 CoinBigIndex presolve_find_col (int *col*, CoinBigIndex *krs*, CoinBigIndex *kre*, const int * *hcol*) [related]

Find position of a column in a row in a row-major matrix.

The routine returns the position *k* in *hcol* for the specified *col*. It will abort if the entry does not exist.

Definition at line 1619 of file `CoinPresolveMatrix.hpp`.

7.1.2.8 CoinBigIndex presolve_find_minor1 (int *tgt*, CoinBigIndex *ks*, CoinBigIndex *ke*, const int * *minndx*)
[related]

Find position of a minor index in a major vector.

The routine returns the position *k* in *minndx* for the specified minor index *tgt*. A return value of *ke* means the entry does not exist. Can be used directly or via the inline wrappers *presolve_find_row1* and *presolve_find_col1*.

7.1.2.9 CoinBigIndex presolve_find_row1 (int *row*, CoinBigIndex *kcs*, CoinBigIndex *kce*, const int * *hrow*)
[related]

Find position of a row in a column in a column-major matrix.

The routine returns the position *k* in *hrow* for the specified *row*. A return value of *kce* means the entry does not exist.

Definition at line 1641 of file *CoinPresolveMatrix.hpp*.

7.1.2.10 CoinBigIndex presolve_find_col1 (int *col*, CoinBigIndex *krs*, CoinBigIndex *kre*, const int * *hcol*)
[related]

Find position of a column in a row in a row-major matrix.

The routine returns the position *k* in *hcol* for the specified *col*. A return value of *kre* means the entry does not exist.

Definition at line 1651 of file *CoinPresolveMatrix.hpp*.

7.1.2.11 CoinBigIndex presolve_find_minor2 (int *tgt*, CoinBigIndex *ks*, int *majlen*, const int * *minndx*, const CoinBigIndex * *majlinks*) [related]

Find position of a minor index in a major vector in a threaded matrix.

The routine returns the position *k* in *minndx* for the specified minor index *tgt*. It will abort if the entry does not exist. Can be used directly or via the inline wrapper *presolve_find_row2*.

7.1.2.12 CoinBigIndex presolve_find_row2 (int *row*, CoinBigIndex *kcs*, int *collen*, const int * *hrow*, const CoinBigIndex * *clinks*) [related]

Find position of a row in a column in a column-major threaded matrix.

The routine returns the position *k* in *hrow* for the specified *row*. It will abort if the entry does not exist.

Definition at line 1674 of file *CoinPresolveMatrix.hpp*.

7.1.2.13 CoinBigIndex presolve_find_minor3 (int *tgt*, CoinBigIndex *ks*, int *majlen*, const int * *minndx*, const CoinBigIndex * *majlinks*) [related]

Find position of a minor index in a major vector in a threaded matrix.

The routine returns the position *k* in *minndx* for the specified minor index *tgt*. It will return -1 if the entry does not exist. Can be used directly or via the inline wrappers *presolve_find_row3*.

7.1.2.14 CoinBigIndex presolve_find_row3 (int *row*, CoinBigIndex *kcs*, int *collen*, const int * *hrow*, const CoinBigIndex * *clinks*) [related]

Find position of a row in a column in a column-major threaded matrix.

The routine returns the position *k* in *hrow* for the specified *row*. It will return -1 if the entry does not exist.

Definition at line 1698 of file *CoinPresolveMatrix.hpp*.

7.1.2.15 `void presolve_delete_from_major (int majndx, int minndx, const CoinBigIndex * majstrts, int * majlens, int * minndxs, double * els)` [related]

Delete the entry for a minor index from a major vector.

Deletes the entry for `minndx` from the major vector `majndx`. Specifically, the relevant entries are removed from the minor index (`minndxs`) and coefficient (`els`) arrays and the vector length (`majlens`) is decremented. Loose packing is maintained by swapping the last entry in the row into the position occupied by the deleted entry.

Definition at line 1712 of file `CoinPresolveMatrix.hpp`.

7.1.2.16 `void presolve_delete_many_from_major (int majndx, char * marked, const CoinBigIndex * majstrts, int * majlens, int * minndxs, double * els)` [related]

Delete marked entries.

Removes the entries specified in `marked`, compressing the major vector to maintain loose packing. `marked` is cleared in the process.

Definition at line 1734 of file `CoinPresolveMatrix.hpp`.

7.1.2.17 `void presolve_delete_from_col (int row, int col, const CoinBigIndex * mcstrt, int * hincol, int * hrow, double * colels)` [related]

Delete the entry for row `row` from column `col` in a column-major matrix.

Deletes the entry for `row` from the major vector for `col`. Specifically, the relevant entries are removed from the row index (`hrow`) and coefficient (`colels`) arrays and the vector length (`hincol`) is decremented. Loose packing is maintained by swapping the last entry in the row into the position occupied by the deleted entry.

Definition at line 1764 of file `CoinPresolveMatrix.hpp`.

7.1.2.18 `void presolve_delete_from_row (int row, int col, const CoinBigIndex * mrstrt, int * hinrow, int * hcol, double * rowels)` [related]

Delete the entry for column `col` from row `row` in a row-major matrix.

Deletes the entry for `col` from the major vector for `row`. Specifically, the relevant entries are removed from the column index (`hcol`) and coefficient (`rowels`) arrays and the vector length (`hinrow`) is decremented. Loose packing is maintained by swapping the last entry in the column into the position occupied by the deleted entry.

Definition at line 1779 of file `CoinPresolveMatrix.hpp`.

7.1.2.19 `void presolve_delete_from_major2 (int majndx, int minndx, CoinBigIndex * majstrts, int * majlens, int * minndxs, int * majlinks, CoinBigIndex * free_listp)` [related]

Delete the entry for a minor index from a major vector in a threaded matrix.

Deletes the entry for `minndx` from the major vector `majndx`. Specifically, the relevant entries are removed from the minor index (`minndxs`) and coefficient (`els`) arrays and the vector length (`majlens`) is decremented. The thread for the major vector is relinked around the deleted entry and the space is returned to the free list.

7.1.2.20 `void presolve_delete_from_col2 (int row, int col, CoinBigIndex * mcstrt, int * hincol, int * hrow, int * clinks, CoinBigIndex * free_listp)` [related]

Delete the entry for row `row` from column `col` in a column-major threaded matrix.

Deletes the entry for `row` from the major vector for `col`. Specifically, the relevant entries are removed from the row index (`hrow`) and coefficient (`colels`) arrays and the vector length (`hincol`) is decremented. The thread for the major vector is relinked around the deleted entry and the space is returned to the free list.

Definition at line 1809 of file CoinPresolveMatrix.hpp.

7.2 Presolve Utility Functions

Utilities used by multiple presolve transform objects.

Functions

- `double * presolve_dupmajor` (const double *elems, const int *indices, int length, [CoinBigIndex](#) offset, int tgt=-1)
Duplicate a major-dimension vector; optionally omit the entry with minor index `tgt`.
- `void coin_init_random_vec` (double *work, int n)
Initialize a vector with random numbers.

7.2.1 Detailed Description

Utilities used by multiple presolve transform objects.

7.2.2 Function Documentation

7.2.2.1 `double* presolve_dupmajor (const double * elems, const int * indices, int length, CoinBigIndex offset, int tgt = -1)`

Duplicate a major-dimension vector; optionally omit the entry with minor index `tgt`.

Designed to copy a major-dimension vector from the paired coefficient (`elems`) and minor index (`indices`) arrays used in the standard packed matrix representation. Copies `length` entries starting at `offset`.

If `tgt` is specified, the entry with minor index == `tgt` is omitted from the copy.

7.2.2.2 `void coin_init_random_vec (double * work, int n)`

Initialize a vector with random numbers.

7.3 Presolve Debug Functions

These functions implement consistency checks on data structures involved in presolve and postsolve and on the components of the lp solution.

Functions

- `void presolve_no_dups` (const `CoinPresolveMatrix` *preObj, bool doCol=true, bool doRow=true)
Check column-major and/or row-major matrices for duplicate entries in the major vectors.
- `void presolve_links_ok` (const `CoinPresolveMatrix` *preObj, bool doCol=true, bool doRow=true)
Check the links which track storage order for major vectors in the bulk storage area.
- `void presolve_no_zeros` (const `CoinPresolveMatrix` *preObj, bool doCol=true, bool doRow=true)
Check for explicit zeros in the column- and/or row-major matrices.
- `void presolve_consistent` (const `CoinPresolveMatrix` *preObj, bool chkvals=true)
Checks for equivalence of the column- and row-major matrices.
- `void presolve_check_threads` (const `CoinPostsolveMatrix` *obj)
Checks that column threads agree with column lengths.
- `void presolve_check_free_list` (const `CoinPostsolveMatrix` *obj, bool chkElemCnt=false)
Checks the free list.
- `void presolve_check_reduced_costs` (const `CoinPostsolveMatrix` *obj)
Check stored reduced costs for accuracy and consistency with variable status.
- `void presolve_check_duals` (const `CoinPostsolveMatrix` *postObj)
Check the dual variables for consistency with row activity.
- `void presolve_check_sol` (const `CoinPresolveMatrix` *preObj, int chkColSol=2, int chkRowAct=1, int chkStatus=1)
Check primal solution and architectural variable status.
- `void presolve_check_sol` (const `CoinPostsolveMatrix` *postObj, int chkColSol=2, int chkRowAct=2, int chkStatus=1)
Check primal solution and architectural variable status.
- `void presolve_check_nbasic` (const `CoinPresolveMatrix` *preObj)
Check for the proper number of basic variables.
- `void presolve_check_nbasic` (const `CoinPostsolveMatrix` *postObj)
Check for the proper number of basic variables.

7.3.1 Detailed Description

These functions implement consistency checks on data structures involved in presolve and postsolve and on the components of the lp solution. To use these functions, include `CoinPresolvePsdebug.hpp` in your file and define the compile-time constants `PRESOLVE_SUMMARY`, `PRESOLVE_DEBUG`, and `PRESOLVE_CONSISTENCY`. A value is needed (*i.e.*, `PRESOLVE_DEBUG=1`). In a few places, higher values will get you a bit more output.

Define the symbols `PRESOLVE_DEBUG` and `PRESOLVE_CONSISTENCY` on the configure command line (use `ADD_CXXFLAGS`), in a Makefile, or similar and do a full rebuild (including any presolve driver code). If the symbols are not consistently nonzero across *all* presolve code, you'll get something between garbage and a core dump! Debugging adds messages to `CoinMessage` and allocates and maintains arrays that hold debug information.

That said, given that you've configured and built with `PRESOLVE_DEBUG` and `PRESOLVE_CONSISTENCY` nonzero everywhere, it's safe to adjust `PRESOLVE_DEBUG` to values in the range 1..n in individual files to increase or decrease the amount of output.

The suggested approach for `PRESOLVE_DEBUG` is to define it to 1 in the build and then increase it in individual presolve code files to get more detail.

7.3.2 Function Documentation

7.3.2.1 void presolve_no_dups (const CoinPresolveMatrix * *preObj*, bool *doCol* = true, bool *doRow* = true)
[related]

Check column-major and/or row-major matrices for duplicate entries in the major vectors.

By default, scans both the column- and row-major matrices. Set *doCol* (*doRow*) to false to suppress the column (row) scan.

7.3.2.2 void presolve_links_ok (const CoinPresolveMatrix * *preObj*, bool *doCol* = true, bool *doRow* = true)
[related]

Check the links which track storage order for major vectors in the bulk storage area.

By default, scans both the column- and row-major matrix. Set *doCol* = false to suppress the column-major scan. Set *doRow* = false to suppress the row-major scan.

7.3.2.3 void presolve_no_zeros (const CoinPresolveMatrix * *preObj*, bool *doCol* = true, bool *doRow* = true)
[related]

Check for explicit zeros in the column- and/or row-major matrices.

By default, scans both the column- and row-major matrices. Set *doCol* (*doRow*) to false to suppress the column (row) scan.

7.3.2.4 void presolve_consistent (const CoinPresolveMatrix * *preObj*, bool *chkvals* = true) [related]

Checks for equivalence of the column- and row-major matrices.

Normally the routine will test for coefficient presence and value. Set *chkvals* to false to suppress the check for equal value.

7.3.2.5 void presolve_check_threads (const CoinPostsolveMatrix * *obj*) [related]

Checks that column threads agree with column lengths.

7.3.2.6 void presolve_check_free_list (const CoinPostsolveMatrix * *obj*, bool *chkElemCnt* = false) [related]

Checks the free list.

Scans the thread of free locations in the bulk store and checks that all entries are reasonable ($0 \leq \text{index} < \text{bulk0_}$). If *chkElemCnt* is true, it also checks that the total number of entries in the matrix plus the locations on the free list total to the size of the bulk store. Postsolve routines do not maintain an accurate element count, but this is useful for checking a newly constructed postsolve matrix.

7.3.2.7 void presolve_check_reduced_costs (const CoinPostsolveMatrix * *obj*) [related]

Check stored reduced costs for accuracy and consistency with variable status.

The routine will check the value of the reduced costs for architectural variables ([CoinPrePostsolveMatrix::rcosts_](#)). It performs an accuracy check by recalculating the reduced cost from scratch. It will also check the value for consistency with the status information in [CoinPrePostsolveMatrix::colstat_](#).

7.3.2.8 void presolve_check_duals (const CoinPostsolveMatrix * *postObj*) [related]

Check the dual variables for consistency with row activity.

The routine checks that the value of the dual variable is consistent with the state of the constraint (loose, tight at lower bound, or tight at upper bound).

7.3.2.9 `void presolve_check_sol (const CoinPresolveMatrix * preObj, int chkColSol = 2, int chkRowAct = 1, int chkStatus = 1)` [related]

Check primal solution and architectural variable status.

The architectural variables can be checked for bogus values, feasibility, and valid status. The row activity is checked for bogus values, accuracy, and feasibility. By default, row activity is not checked (presolve is sloppy about maintaining it). See the definitions in `CoinPresolvePsdebug.cpp` for more information.

7.3.2.10 `void presolve_check_sol (const CoinPostsolveMatrix * postObj, int chkColSol = 2, int chkRowAct = 2, int chkStatus = 1)` [related]

Check primal solution and architectural variable status.

The architectural variables can be checked for bogus values, feasibility, and valid status. The row activity is checked for bogus values, accuracy, and feasibility. See the definitions in `CoinPresolvePsdebug.cpp` for more information.

7.3.2.11 `void presolve_check_nbasic (const CoinPresolveMatrix * preObj)` [related]

Check for the proper number of basic variables.

7.3.2.12 `void presolve_check_nbasic (const CoinPostsolveMatrix * postObj)` [related]

Check for the proper number of basic variables.

8 Namespace Documentation

8.1 Coin Namespace Reference

Classes

- class [ReferencedObject](#)
ReferencedObject class.
- class [SmartPtr](#)
Template class for Smart Pointers.

Functions

- `template<class U1 , class U2 > bool ComparePointers (const U1 *lhs, const U2 *rhs)`

8.1.1 Function Documentation

8.1.1.1 `template<class U1 , class U2 > bool Coin::ComparePointers (const U1 * lhs, const U2 * rhs)`

Definition at line 476 of file CoinSmartPtr.hpp.

8.2 CoinParamUtils Namespace Reference

Utility functions for processing [CoinParam](#) parameters.

Functions

- `void setInputSrc (FILE *src)`
Take command input from the file specified by src.
- `bool isCommandLine ()`
Returns true if command line parameters are being processed.
- `bool isInteractive ()`
Returns true if parameters are being obtained from stdin.
- `std::string getStringField (int argc, const char *argv[], int *valid)`
Attempt to read a string from the input.
- `int getIntField (int argc, const char *argv[], int *valid)`
Attempt to read an integer from the input.
- `double getDoubleField (int argc, const char *argv[], int *valid)`
Attempt to read a real (double) from the input.
- `int matchParam (const CoinParamVec ¶mVec, std::string name, int &matchNdx, int &shortCnt)`
*Scan a parameter vector for parameters whose keyword (name) string matches *name* using minimal match rules.*
- `std::string getCommand (int argc, const char *argv[], const std::string prompt, std::string *pfx=0)`
Get the next command keyword (name)
- `int lookupParam (std::string name, CoinParamVec ¶mVec, int *matchCnt=0, int *shortCnt=0, int *queryCnt=0)`
Look up the command keyword (name) in the parameter vector. Print help if requested.

- `void printIt` (const char *msg)
Utility to print a long message as filled lines of text.
- `void shortOrHelpOne` (CoinParamVec ¶mVec, int matchNdx, std::string name, int numQuery)
Utility routine to print help given a short match or explicit request for help.
- `void shortOrHelpMany` (CoinParamVec ¶mVec, std::string name, int numQuery)
Utility routine to print help given multiple matches.
- `void printGenericHelp` ()
Print a generic 'how to use the command interface' help message.
- `void printHelp` (CoinParamVec ¶mVec, int firstParam, int lastParam, std::string prefix, bool shortHelp, bool longHelp, bool hidden)
Utility routine to print help messages for one or more parameters.

8.2.1 Detailed Description

Utility functions for processing `CoinParam` parameters. The functions in `CoinParamUtils` support command line or interactive parameter processing and a help facility. Consult the 'Related Functions' section of the `CoinParam` class documentation for individual function documentation.

8.2.2 Function Documentation

8.2.2.1 `void CoinParamUtils::setInputSrc (FILE * src)`

Take command input from the file specified by `src`.

Use `stdin` for `src` to specify interactive prompting for commands.

8.2.2.2 `bool CoinParamUtils::isCommandLine ()`

Returns true if command line parameters are being processed.

8.2.2.3 `bool CoinParamUtils::isInteractive ()`

Returns true if parameters are being obtained from `stdin`.

8.2.2.4 `std::string CoinParamUtils::getStringField (int argc, const char * argv[], int * valid)`

Attempt to read a string from the input.

`argc` and `argv` are used only if `isCommandLine()` would return true. If `valid` is supplied, it will be set to 0 if a string is parsed without error, 2 if no field is present.

8.2.2.5 `int CoinParamUtils::getIntField (int argc, const char * argv[], int * valid)`

Attempt to read an integer from the input.

`argc` and `argv` are used only if `isCommandLine()` would return true. If `valid` is supplied, it will be set to 0 if an integer is parsed without error, 1 if there's a parse error, and 2 if no field is present.

8.2.2.6 `double CoinParamUtils::getDoubleField (int argc, const char * argv[], int * valid)`

Attempt to read a real (double) from the input.

`argc` and `argv` are used only if `isCommandLine()` would return true. If `valid` is supplied, it will be set to 0 if a real number is parsed without error, 1 if there's a parse error, and 2 if no field is present.

8.2.2.7 `int CoinParamUtils::matchParam (const CoinParamVec & paramVec, std::string name, int & matchNdx, int & shortCnt)`

Scan a parameter vector for parameters whose keyword (name) string matches `name` using minimal match rules.

`matchNdx` is set to the index of the last parameter that meets the minimal match criteria (but note there should be at most one matching parameter if the parameter vector is properly configured). `shortCnt` is set to the number of short matches (should be zero for a properly configured parameter vector if a minimal match is found). The return value is the number of matches satisfying the minimal match requirement (should be 0 or 1 in a properly configured vector).

8.2.2.8 `std::string CoinParamUtils::getCommand (int argc, const char * argv[], const std::string prompt, std::string * pfx = 0)`

Get the next command keyword (name)

To be precise, return the next field from the current command input source, after a bit of processing. In command line mode (`isCommandLine()` returns true) the next field will normally be of the form '-keyword' or '-keyword' (i.e., a parameter keyword), and the string returned would be 'keyword'. In interactive mode (`isInteractive()` returns true), the user will be prompted if necessary. It is assumed that the user knows not to use the '-' or '-' prefixes unless specifying parameters on the command line.

There are a number of special cases if we're in command line mode. The order of processing of the raw string goes like this:

- A stand-alone '-' is forced to 'stdin'.
- A stand-alone '-' is returned as a word; interpretation is up to the client.
- A prefix of '-' or '-' is stripped from the string.

If the result is the string 'stdin', command processing shifts to interactive mode and the user is immediately prompted for a new command.

Whatever results from the above sequence is returned to the user as the return value of the function. An empty string indicates end of input.

`prompt` will be used only if it's necessary to prompt the user in interactive mode.

8.2.2.9 `int CoinParamUtils::lookupParam (std::string name, CoinParamVec & paramVec, int * matchCnt = 0, int * shortCnt = 0, int * queryCnt = 0)`

Look up the command keyword (name) in the parameter vector. Print help if requested.

In the most straightforward use, `name` is a string without '?', and the value returned is the index in `paramVec` of the single parameter that matched `name`. One or more '?' characters at the end of `name` is a query for information. The routine prints short (one '?') or long (more than one '?') help messages for a query. Help is also printed in the case where the name is ambiguous (some of the matches did not meet the minimal match length requirement).

Note that multiple matches meeting the minimal match requirement is a configuration error. The minimal match length for the parameters involved is too short.

If provided as parameters, on return

- `matchCnt` will be set to the number of matches meeting the minimal match requirement
- `shortCnt` will be set to the number of matches that did not meet the minimal match requirement
- `queryCnt` will be set to the number of '?' characters at the end of the name

The return values are:

- `>0`: index in `paramVec` of the single unique match for `name`

- -1: a query was detected (one or more '?' characters at the end of `name`)
- -2: one or more short matches, not a query
- -3: no matches, not a query
- -4: multiple matches meeting the minimal match requirement (configuration error)

8.2.2.10 `void CoinParamUtils::printIt (const char * msg)`

Utility to print a long message as filled lines of text.

The routine makes a best effort to break lines without exceeding the standard 80 character line length. Explicit newlines in `msg` will be obeyed.

8.2.2.11 `void CoinParamUtils::shortOrHelpOne (CoinParamVec & paramVec, int matchNdx, std::string name, int numQuery)`

Utility routine to print help given a short match or explicit request for help.

The two really are related, in that a query (a string that ends with one or more '?' characters) will often result in a short match. The routine expects that `name` matches a single parameter, and does not look for multiple matches.

If called with `matchNdx < 0`, the routine will look up `name` in `paramVec` and print the full name from the parameter. If called with `matchNdx > 0`, it just prints the name from the specified parameter. If the name is a query, short (one '?') or long (more than one '?') help is printed.

8.2.2.12 `void CoinParamUtils::shortOrHelpMany (CoinParamVec & paramVec, std::string name, int numQuery)`

Utility routine to print help given multiple matches.

If the name is not a query, or asks for short help (*i.e.*, contains zero or one '?' characters), the list of matching names is printed. If the name asks for long help (contains two or more '?' characters), short help is printed for each matching name.

8.2.2.13 `void CoinParamUtils::printGenericHelp ()`

Print a generic 'how to use the command interface' help message.

The message is hard coded to match the behaviour of the parsing utilities.

8.2.2.14 `void CoinParamUtils::printHelp (CoinParamVec & paramVec, int firstParam, int lastParam, std::string prefix, bool shortHelp, bool longHelp, bool hidden)`

Utility routine to print help messages for one or more parameters.

Intended as a utility to implement explicit 'help' commands. Help will be printed for all parameters in `paramVec` from `firstParam` to `lastParam`, inclusive. If `shortHelp` is true, short help messages will be printed. If `longHelp` is true, long help messages are printed. `shortHelp` overrules `longHelp`. If neither is true, only command keywords are printed. `prefix` is printed before each line; it's an imperfect attempt at indentation.

9 Class Documentation

9.1 `_EKKfactinfo` Struct Reference

```
#include <CoinOslFactorization.hpp>
```

Public Attributes

- double [drtpiv](#)
- double [demark](#)
- double [zpivlu](#)
- double [zeroTolerance](#)
- double [areaFactor](#)
- int * [xrsadr](#)
- int * [xcsadr](#)
- int * [xrnadr](#)
- int * [xcnadr](#)
- int * [krpadr](#)
- int * [kcpadr](#)
- int * [mpermu](#)
- int * [bitArray](#)
- int * [back](#)
- char * [nonzero](#)
- double * [trueStart](#)
- double * [kadrpm](#)
- int * [R_etas_index](#)
- int * [R_etas_start](#)
- double * [R_etas_element](#)
- int * [xecadr](#)
- int * [xeradr](#)
- double * [xeeadr](#)
- double * [xe2adr](#)
- EKKHlink * [kp1adr](#)
- EKKHlink * [kp2adr](#)
- double * [kw1adr](#)
- double * [kw2adr](#)
- double * [kw3adr](#)
- int * [hpivcoR](#)
- int [nrow](#)
- int [nrowmx](#)
- int [firstDoRow](#)
- int [firstLRow](#)
- int [maxinv](#)
- int [nnetas](#)
- int [iterin](#)
- int [iter0](#)
- int [invok](#)
- int [nbfinv](#)
- int [num_resets](#)
- int [nnentl](#)
- int [nnentu](#)
- int [ndenuc](#)
- int [npivots](#)
- int [kmxeta](#)
- int [xnetal](#)
- int [first_dense](#)
- int [last_dense](#)

- int [iterno](#)
- int [numberSlacks](#)
- int [lastSlack](#)
- int [firstNonSlack](#)
- int [xnetalval](#)
- int [lstart](#)
- int [if_sparse_update](#)
- int [packedMode](#)
- int [switch_off_sparse_update](#)
- int [nuspikes](#)
- bool [rows_ok](#)
- int [nR_etas](#)
- int [sortedEta](#)
- int [lastEtaCount](#)
- int [ifvsol](#)
- int [eta_size](#)
- int [last_eta_size](#)
- int [maxNNetas](#)

9.1.1 Detailed Description

Definition at line 29 of file `CoinOslFactorization.hpp`.

9.1.2 Member Data Documentation

9.1.2.1 `double _EKKfactinfo::drtpiv`

Definition at line 30 of file `CoinOslFactorization.hpp`.

9.1.2.2 `double _EKKfactinfo::demark`

Definition at line 31 of file `CoinOslFactorization.hpp`.

9.1.2.3 `double _EKKfactinfo::zpivlu`

Definition at line 32 of file `CoinOslFactorization.hpp`.

9.1.2.4 `double _EKKfactinfo::zeroTolerance`

Definition at line 33 of file `CoinOslFactorization.hpp`.

9.1.2.5 `double _EKKfactinfo::areaFactor`

Definition at line 34 of file `CoinOslFactorization.hpp`.

9.1.2.6 `int* _EKKfactinfo::xrsadr`

Definition at line 35 of file `CoinOslFactorization.hpp`.

9.1.2.7 `int* _EKKfactinfo::xcsadr`

Definition at line 36 of file `CoinOslFactorization.hpp`.

9.1.2.8 `int* _EKKfactinfo::xrnadr`

Definition at line 37 of file `CoinOslFactorization.hpp`.

9.1.2.9 `int* _EKKfactinfo::xcnadr`

Definition at line 38 of file `CoinOslFactorization.hpp`.

9.1.2.10 `int* _EKKfactinfo::krpadr`

Definition at line 39 of file `CoinOslFactorization.hpp`.

9.1.2.11 `int* _EKKfactinfo::kcpadr`

Definition at line 40 of file `CoinOslFactorization.hpp`.

9.1.2.12 `int* _EKKfactinfo::mpermu`

Definition at line 41 of file `CoinOslFactorization.hpp`.

9.1.2.13 `int* _EKKfactinfo::bitArray`

Definition at line 42 of file `CoinOslFactorization.hpp`.

9.1.2.14 `int* _EKKfactinfo::back`

Definition at line 43 of file `CoinOslFactorization.hpp`.

9.1.2.15 `char* _EKKfactinfo::nonzero`

Definition at line 44 of file `CoinOslFactorization.hpp`.

9.1.2.16 `double* _EKKfactinfo::trueStart`

Definition at line 45 of file `CoinOslFactorization.hpp`.

9.1.2.17 `double* _EKKfactinfo::kadrpm` `[mutable]`

Definition at line 46 of file `CoinOslFactorization.hpp`.

9.1.2.18 `int* _EKKfactinfo::R_etas_index`

Definition at line 47 of file `CoinOslFactorization.hpp`.

9.1.2.19 `int* _EKKfactinfo::R_etas_start`

Definition at line 48 of file `CoinOslFactorization.hpp`.

9.1.2.20 `double* _EKKfactinfo::R_etas_element`

Definition at line 49 of file `CoinOslFactorization.hpp`.

9.1.2.21 `int* _EKKfactinfo::xecadr`

Definition at line 51 of file `CoinOslFactorization.hpp`.

9.1.2.22 int* _EKKfactinfo::xeradr

Definition at line 52 of file CoinOslFactorization.hpp.

9.1.2.23 double* _EKKfactinfo::xeeadr

Definition at line 53 of file CoinOslFactorization.hpp.

9.1.2.24 double* _EKKfactinfo::xe2adr

Definition at line 54 of file CoinOslFactorization.hpp.

9.1.2.25 EKKHlink* _EKKfactinfo::kp1adr

Definition at line 55 of file CoinOslFactorization.hpp.

9.1.2.26 EKKHlink* _EKKfactinfo::kp2adr

Definition at line 56 of file CoinOslFactorization.hpp.

9.1.2.27 double* _EKKfactinfo::kw1adr

Definition at line 57 of file CoinOslFactorization.hpp.

9.1.2.28 double* _EKKfactinfo::kw2adr

Definition at line 58 of file CoinOslFactorization.hpp.

9.1.2.29 double* _EKKfactinfo::kw3adr

Definition at line 59 of file CoinOslFactorization.hpp.

9.1.2.30 int* _EKKfactinfo::hpivcoR

Definition at line 60 of file CoinOslFactorization.hpp.

9.1.2.31 int _EKKfactinfo::nrow

Definition at line 61 of file CoinOslFactorization.hpp.

9.1.2.32 int _EKKfactinfo::nrowmx

Definition at line 62 of file CoinOslFactorization.hpp.

9.1.2.33 int _EKKfactinfo::firstDoRow

Definition at line 63 of file CoinOslFactorization.hpp.

9.1.2.34 int _EKKfactinfo::firstLRow

Definition at line 64 of file CoinOslFactorization.hpp.

9.1.2.35 int _EKKfactinfo::maxinv

Definition at line 65 of file CoinOslFactorization.hpp.

9.1.2.36 `int _EKKfactinfo::nnetas`

Definition at line 66 of file `CoinOslFactorization.hpp`.

9.1.2.37 `int _EKKfactinfo::iterin`

Definition at line 67 of file `CoinOslFactorization.hpp`.

9.1.2.38 `int _EKKfactinfo::iter0`

Definition at line 68 of file `CoinOslFactorization.hpp`.

9.1.2.39 `int _EKKfactinfo::invok`

Definition at line 69 of file `CoinOslFactorization.hpp`.

9.1.2.40 `int _EKKfactinfo::nbfinv`

Definition at line 70 of file `CoinOslFactorization.hpp`.

9.1.2.41 `int _EKKfactinfo::num_resets`

Definition at line 71 of file `CoinOslFactorization.hpp`.

9.1.2.42 `int _EKKfactinfo::nnentl`

Definition at line 72 of file `CoinOslFactorization.hpp`.

9.1.2.43 `int _EKKfactinfo::nnentu`

Definition at line 73 of file `CoinOslFactorization.hpp`.

9.1.2.44 `int _EKKfactinfo::ndenuc`

Definition at line 77 of file `CoinOslFactorization.hpp`.

9.1.2.45 `int _EKKfactinfo::npivots`

Definition at line 78 of file `CoinOslFactorization.hpp`.

9.1.2.46 `int _EKKfactinfo::kxmxta`

Definition at line 79 of file `CoinOslFactorization.hpp`.

9.1.2.47 `int _EKKfactinfo::xnetal`

Definition at line 80 of file `CoinOslFactorization.hpp`.

9.1.2.48 `int _EKKfactinfo::first_dense`

Definition at line 81 of file `CoinOslFactorization.hpp`.

9.1.2.49 `int _EKKfactinfo::last_dense`

Definition at line 82 of file `CoinOslFactorization.hpp`.

9.1.2.50 `int _EKKfactinfo::iterno`

Definition at line 83 of file `CoinOslFactorization.hpp`.

9.1.2.51 `int _EKKfactinfo::numberSlacks`

Definition at line 84 of file `CoinOslFactorization.hpp`.

9.1.2.52 `int _EKKfactinfo::lastSlack`

Definition at line 85 of file `CoinOslFactorization.hpp`.

9.1.2.53 `int _EKKfactinfo::firstNonSlack`

Definition at line 86 of file `CoinOslFactorization.hpp`.

9.1.2.54 `int _EKKfactinfo::xnetalval`

Definition at line 87 of file `CoinOslFactorization.hpp`.

9.1.2.55 `int _EKKfactinfo::lstart`

Definition at line 88 of file `CoinOslFactorization.hpp`.

9.1.2.56 `int _EKKfactinfo::if_sparse_update`

Definition at line 89 of file `CoinOslFactorization.hpp`.

9.1.2.57 `int _EKKfactinfo::packedMode` `[mutable]`

Definition at line 90 of file `CoinOslFactorization.hpp`.

9.1.2.58 `int _EKKfactinfo::switch_off_sparse_update`

Definition at line 91 of file `CoinOslFactorization.hpp`.

9.1.2.59 `int _EKKfactinfo::nuspikes`

Definition at line 92 of file `CoinOslFactorization.hpp`.

9.1.2.60 `bool _EKKfactinfo::rows_ok`

Definition at line 93 of file `CoinOslFactorization.hpp`.

9.1.2.61 `int _EKKfactinfo::nR_etas`

Definition at line 97 of file `CoinOslFactorization.hpp`.

9.1.2.62 `int _EKKfactinfo::sortedEta`

Definition at line 98 of file `CoinOslFactorization.hpp`.

9.1.2.63 `int _EKKfactinfo::lastEtaCount`

Definition at line 99 of file `CoinOslFactorization.hpp`.

9.1.2.64 int _EKKfactinfo::ifvsol

Definition at line 100 of file CoinOslFactorization.hpp.

9.1.2.65 int _EKKfactinfo::eta_size

Definition at line 101 of file CoinOslFactorization.hpp.

9.1.2.66 int _EKKfactinfo::last_eta_size

Definition at line 102 of file CoinOslFactorization.hpp.

9.1.2.67 int _EKKfactinfo::maxNNetas

Definition at line 103 of file CoinOslFactorization.hpp.

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinOslFactorization.hpp](#)

9.2 doubleton_action::action Struct Reference

```
#include <CoinPresolveDoubleton.hpp>
```

Public Attributes

- double [clox](#)
- double [cupx](#)
- double [costx](#)
- double [costy](#)
- double [rlo](#)
- double [coeffx](#)
- double [coeffy](#)
- double * [colel](#)
- int [icolx](#)
- int [icoly](#)
- int [row](#)
- int [ncolx](#)
- int [ncoly](#)

9.2.1 Detailed Description

Definition at line 28 of file CoinPresolveDoubleton.hpp.

9.2.2 Member Data Documentation

9.2.2.1 double doubleton_action::action::clox

Definition at line 30 of file CoinPresolveDoubleton.hpp.

9.2.2.2 double doubleton_action::action::cupx

Definition at line 31 of file CoinPresolveDoubleton.hpp.

9.2.2.3 double doubleton_action::action::costx

Definition at line 32 of file CoinPresolveDoubleton.hpp.

9.2.2.4 double doubleton_action::action::costy

Definition at line 34 of file CoinPresolveDoubleton.hpp.

9.2.2.5 double doubleton_action::action::rlo

Definition at line 36 of file CoinPresolveDoubleton.hpp.

9.2.2.6 double doubleton_action::action::coeffx

Definition at line 38 of file CoinPresolveDoubleton.hpp.

9.2.2.7 double doubleton_action::action::coeffy

Definition at line 39 of file CoinPresolveDoubleton.hpp.

9.2.2.8 double* doubleton_action::action::colel

Definition at line 41 of file CoinPresolveDoubleton.hpp.

9.2.2.9 int doubleton_action::action::icolx

Definition at line 43 of file CoinPresolveDoubleton.hpp.

9.2.2.10 int doubleton_action::action::icoly

Definition at line 44 of file CoinPresolveDoubleton.hpp.

9.2.2.11 int doubleton_action::action::row

Definition at line 45 of file CoinPresolveDoubleton.hpp.

9.2.2.12 int doubleton_action::action::ncolx

Definition at line 46 of file CoinPresolveDoubleton.hpp.

9.2.2.13 int doubleton_action::action::ncoly

Definition at line 47 of file CoinPresolveDoubleton.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDoubleton.hpp](#)

9.3 remove_fixed_action::action Struct Reference

Structure to hold information necessary to reintroduce a column into the problem representation.

```
#include <CoinPresolveFixed.hpp>
```

Public Attributes

- int [col](#)

- column index of variable*
 - int [start](#)
start of coefficients in [coels_](#) and [colrows_](#)
 - double [sol](#)
value of variable

9.3.1 Detailed Description

Structure to hold information necessary to reintroduce a column into the problem representation.

Definition at line 30 of file CoinPresolveFixed.hpp.

9.3.2 Member Data Documentation

9.3.2.1 int remove_fixed_action::action::col

column index of variable

Definition at line 31 of file CoinPresolveFixed.hpp.

9.3.2.2 int remove_fixed_action::action::start

start of coefficients in [coels_](#) and [colrows_](#)

Definition at line 32 of file CoinPresolveFixed.hpp.

9.3.2.3 double remove_fixed_action::action::sol

value of variable

Definition at line 33 of file CoinPresolveFixed.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveFixed.hpp](#)

9.4 forcing_constraint_action::action Struct Reference

```
#include <CoinPresolveForcing.hpp>
```

Public Attributes

- const int * [rowcols](#)
- const double * [bounds](#)
- int [row](#)
- int [nlo](#)
- int [nup](#)

9.4.1 Detailed Description

Definition at line 32 of file CoinPresolveForcing.hpp.

9.4.2 Member Data Documentation

9.4.2.1 `const int* forcing_constraint_action::action::rowcols`

Definition at line 33 of file `CoinPresolveForcing.hpp`.

9.4.2.2 `const double* forcing_constraint_action::action::bounds`

Definition at line 34 of file `CoinPresolveForcing.hpp`.

9.4.2.3 `int forcing_constraint_action::action::row`

Definition at line 35 of file `CoinPresolveForcing.hpp`.

9.4.2.4 `int forcing_constraint_action::action::nlo`

Definition at line 36 of file `CoinPresolveForcing.hpp`.

9.4.2.5 `int forcing_constraint_action::action::nup`

Definition at line 37 of file `CoinPresolveForcing.hpp`.

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

- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveForcing.hpp`

9.5 `tripleton_action::action` Struct Reference

```
#include <CoinPresolveTripletion.hpp>
```

Public Attributes

- `int icolx`
- `int icolz`
- `int row`
- `int icoly`
- `double cloy`
- `double cupy`
- `double costy`
- `double clox`
- `double cupx`
- `double costx`
- `double rlo`
- `double rup`
- `double coeffx`
- `double coeffy`
- `double coeffz`
- `double * colel`
- `int ncolx`
- `int ncoly`

9.5.1 Detailed Description

Definition at line 17 of file CoinPresolveTripleton.hpp.

9.5.2 Member Data Documentation

9.5.2.1 int tripleton_action::action::icolx

Definition at line 18 of file CoinPresolveTripleton.hpp.

9.5.2.2 int tripleton_action::action::icolz

Definition at line 19 of file CoinPresolveTripleton.hpp.

9.5.2.3 int tripleton_action::action::row

Definition at line 20 of file CoinPresolveTripleton.hpp.

9.5.2.4 int tripleton_action::action::icoly

Definition at line 22 of file CoinPresolveTripleton.hpp.

9.5.2.5 double tripleton_action::action::cloy

Definition at line 23 of file CoinPresolveTripleton.hpp.

9.5.2.6 double tripleton_action::action::cupy

Definition at line 24 of file CoinPresolveTripleton.hpp.

9.5.2.7 double tripleton_action::action::costy

Definition at line 25 of file CoinPresolveTripleton.hpp.

9.5.2.8 double tripleton_action::action::clox

Definition at line 26 of file CoinPresolveTripleton.hpp.

9.5.2.9 double tripleton_action::action::cupx

Definition at line 27 of file CoinPresolveTripleton.hpp.

9.5.2.10 double tripleton_action::action::costx

Definition at line 28 of file CoinPresolveTripleton.hpp.

9.5.2.11 double tripleton_action::action::rlo

Definition at line 30 of file CoinPresolveTripleton.hpp.

9.5.2.12 double tripleton_action::action::rup

Definition at line 31 of file CoinPresolveTripleton.hpp.

9.5.2.13 double tripleton_action::action::coeffx

Definition at line 33 of file CoinPresolveTripleton.hpp.

9.5.2.14 `double tripleton_action::action::coeffy`

Definition at line 34 of file `CoinPresolveTripleton.hpp`.

9.5.2.15 `double tripleton_action::action::coeffz`

Definition at line 35 of file `CoinPresolveTripleton.hpp`.

9.5.2.16 `double* tripleton_action::action::colel`

Definition at line 37 of file `CoinPresolveTripleton.hpp`.

9.5.2.17 `int tripleton_action::action::ncolx`

Definition at line 39 of file `CoinPresolveTripleton.hpp`.

9.5.2.18 `int tripleton_action::action::ncoly`

Definition at line 40 of file `CoinPresolveTripleton.hpp`.

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

- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTripleton.hpp`

9.6 BitVector128 Class Reference

```
#include <CoinSearchTree.hpp>
```

Public Member Functions

- `BitVector128 ()`
- `BitVector128 (unsigned int bits[4])`
- `~BitVector128 ()`
- `void set (unsigned int bits[4])`
- `void setBit (int i)`
- `void clearBit (int i)`
- `std::string str () const`

Friends

- `bool operator< (const BitVector128 &b0, const BitVector128 &b1)`

9.6.1 Detailed Description

Definition at line 21 of file `CoinSearchTree.hpp`.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 `BitVector128::BitVector128 ()`

9.6.2.2 `BitVector128::BitVector128 (unsigned int bits[4])`

9.6.2.3 BitVector128::~~BitVector128 () [inline]

Definition at line 28 of file CoinSearchTree.hpp.

9.6.3 Member Function Documentation

9.6.3.1 void BitVector128::set (unsigned int *bits*[4])9.6.3.2 void BitVector128::setBit (int *i*)9.6.3.3 void BitVector128::clearBit (int *i*)

9.6.3.4 std::string BitVector128::str () const

9.6.4 Friends And Related Function Documentation

9.6.4.1 bool operator< (const BitVector128 & *b0*, const BitVector128 & *b1*) [friend]

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinSearchTree.hpp](#)

9.7 CoinAbsFltEq Class Reference

Equality to an absolute tolerance.

```
#include <CoinFloatEqual.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (const double *f1*, const double *f2*) const
Compare function.

Constructors and destructors

- [CoinAbsFltEq](#) ()
Default constructor.
- [CoinAbsFltEq](#) (const double *epsilon*)
Alternate constructor with epsilon as a parameter.
- virtual [~CoinAbsFltEq](#) ()
Destructor.
- [CoinAbsFltEq](#) (const [CoinAbsFltEq](#) &*src*)
Copy constructor.
- [CoinAbsFltEq](#) & [operator=](#) (const [CoinAbsFltEq](#) &*rhs*)
Assignment.

9.7.1 Detailed Description

Equality to an absolute tolerance.

Operands are considered equal if their difference is within an epsilon ; the test does not consider the relative magnitude of the operands.

Definition at line 46 of file CoinFloatEqual.hpp.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 `CoinAbsFltEq::CoinAbsFltEq ()` `[inline]`

Default constructor.

Default tolerance is 1.0e-10.

Definition at line 66 of file `CoinFloatEqual.hpp`.

9.7.2.2 `CoinAbsFltEq::CoinAbsFltEq (const double epsilon)` `[inline]`

Alternate constructor with epsilon as a parameter.

Definition at line 70 of file `CoinFloatEqual.hpp`.

9.7.2.3 `virtual CoinAbsFltEq::~CoinAbsFltEq ()` `[inline],[virtual]`

Destructor.

Definition at line 74 of file `CoinFloatEqual.hpp`.

9.7.2.4 `CoinAbsFltEq::CoinAbsFltEq (const CoinAbsFltEq & src)` `[inline]`

Copy constructor.

Definition at line 78 of file `CoinFloatEqual.hpp`.

9.7.3 Member Function Documentation

9.7.3.1 `bool CoinAbsFltEq::operator() (const double f1, const double f2) const` `[inline]`

Compare function.

Definition at line 52 of file `CoinFloatEqual.hpp`.

9.7.3.2 `CoinAbsFltEq& CoinAbsFltEq::operator= (const CoinAbsFltEq & rhs)` `[inline]`

Assignment.

Definition at line 82 of file `CoinFloatEqual.hpp`.

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

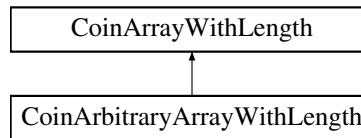
- [/home/ted/COIN/trunk/CoinUtils/src/CoinFloatEqual.hpp](#)

9.8 CoinArbitraryArrayWithLength Class Reference

arbitrary version

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for `CoinArbitraryArrayWithLength`:



Public Member Functions

Get methods.

- int [getSize](#) () const
Get the size.
- void ** [array](#) () const
Get Array.

Set methods

- void [setSize](#) (int value)
Set the size.

Condition methods

- char * [conditionalNew](#) (int length, int sizeWanted)
Conditionally gets new array.

Constructors and destructors

- [CoinArbitraryArrayWithLength](#) (int length=1)
Default constructor - NULL.
- [CoinArbitraryArrayWithLength](#) (int length, int size)
Alternate Constructor - length in bytes - size_ - 1.
- [CoinArbitraryArrayWithLength](#) (int length, int size, int mode)
Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.
- [CoinArbitraryArrayWithLength](#) (const [CoinArbitraryArrayWithLength](#) &rhs)
Copy constructor.
- [CoinArbitraryArrayWithLength](#) (const [CoinArbitraryArrayWithLength](#) *rhs)
Copy constructor.2.
- [CoinArbitraryArrayWithLength](#) & [operator=](#) (const [CoinArbitraryArrayWithLength](#) &rhs)
Assignment operator.

Protected Attributes

Private member data

- int [lengthInBytes_](#)
Length in bytes.

9.8.1 Detailed Description

arbitrary version

Definition at line 995 of file CoinIndexedVector.hpp.

9.8.2 Constructor & Destructor Documentation

9.8.2.1 `CoinArbitraryArrayWithLength::CoinArbitraryArrayWithLength (int length = 1) [inline]`

Default constructor - NULL.

Definition at line 1026 of file `CoinIndexedVector.hpp`.

9.8.2.2 `CoinArbitraryArrayWithLength::CoinArbitraryArrayWithLength (int length, int size) [inline]`

Alternate Constructor - length in bytes - size_ -1.

Definition at line 1029 of file `CoinIndexedVector.hpp`.

9.8.2.3 `CoinArbitraryArrayWithLength::CoinArbitraryArrayWithLength (int length, int size, int mode) [inline]`

Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.

Definition at line 1035 of file `CoinIndexedVector.hpp`.

9.8.2.4 `CoinArbitraryArrayWithLength::CoinArbitraryArrayWithLength (const CoinArbitraryArrayWithLength & rhs) [inline]`

Copy constructor.

Definition at line 1038 of file `CoinIndexedVector.hpp`.

9.8.2.5 `CoinArbitraryArrayWithLength::CoinArbitraryArrayWithLength (const CoinArbitraryArrayWithLength * rhs) [inline]`

Copy constructor.2.

Definition at line 1041 of file `CoinIndexedVector.hpp`.

9.8.3 Member Function Documentation

9.8.3.1 `int CoinArbitraryArrayWithLength::getSize () const [inline]`

Get the size.

Definition at line 1001 of file `CoinIndexedVector.hpp`.

9.8.3.2 `void** CoinArbitraryArrayWithLength::array () const [inline]`

Get Array.

Definition at line 1004 of file `CoinIndexedVector.hpp`.

9.8.3.3 `void CoinArbitraryArrayWithLength::setSize (int value) [inline]`

Set the size.

Definition at line 1011 of file `CoinIndexedVector.hpp`.

9.8.3.4 `char* CoinArbitraryArrayWithLength::conditionalNew (int length, int sizeWanted) [inline]`

Conditionally gets new array.

Definition at line 1018 of file `CoinIndexedVector.hpp`.

9.8.3.5 CoinArbitraryArrayWithLength& CoinArbitraryArrayWithLength::operator= (const CoinArbitraryArrayWithLength & rhs) [inline]

Assignment operator.

Definition at line 1044 of file CoinIndexedVector.hpp.

9.8.4 Member Data Documentation

9.8.4.1 int CoinArbitraryArrayWithLength::lengthInBytes_ [protected]

Length in bytes.

Definition at line 1052 of file CoinIndexedVector.hpp.

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

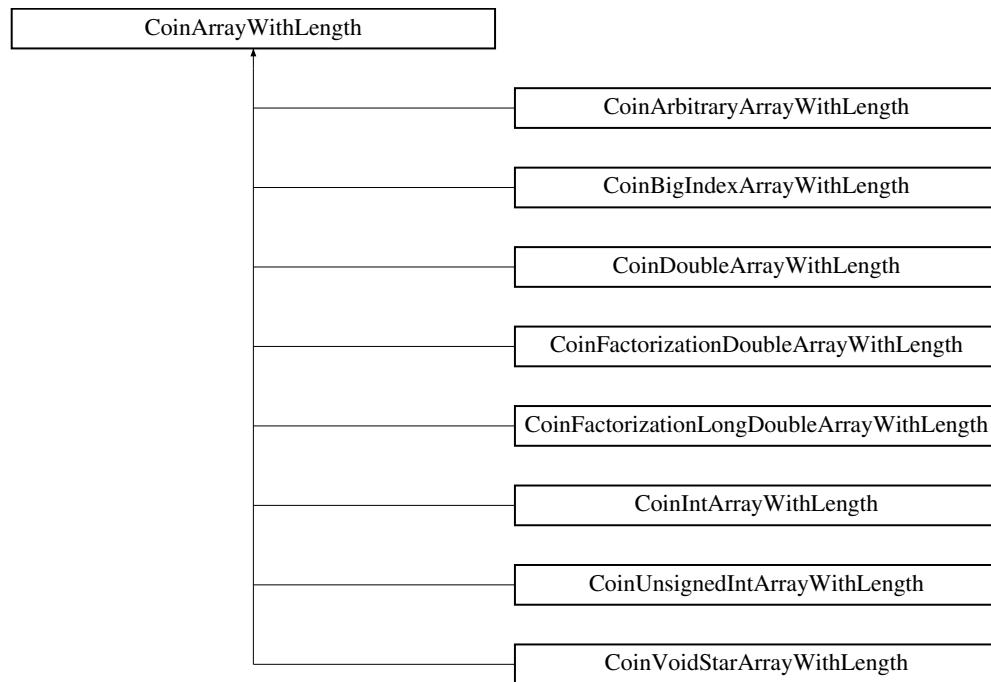
- /home/ted/COIN/trunk/CoinUtils/src/[CoinIndexedVector.hpp](#)

9.9 CoinArrayWithLength Class Reference

Pointer with length in bytes.

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for CoinArrayWithLength:



Public Member Functions

Get methods.

- int [getSize](#) () const

- *Get the size.*
- int `rawSize` () const
- *Get the size.*
- bool `switchedOn` () const
- *See if persistence already on.*
- int `capacity` () const
- *Get the capacity (just read it)*
- void `setCapacity` ()
- *Set the capacity to ≥ 0 if ≤ -2 .*
- const char * `array` () const
- *Get Array.*

Set methods

- void `setSize` (int value)
- *Set the size.*
- void `switchOff` ()
- *Set the size to -1.*
- void `switchOn` (int alignment=3)
- *Set the size to -2 and alignment.*
- void `setPersistence` (int flag, int currentLength)
- *Does what is needed to set persistence.*
- void `clear` ()
- *Zero out array.*
- void `swap` (CoinArrayWithLength &other)
- *Swaps memory between two members.*
- void `extend` (int newSize)
- *Extend a persistent array keeping data (size in bytes)*

Condition methods

- char * `conditionalNew` (long sizeWanted)
- *Conditionally gets new array.*
- void `conditionalDelete` ()
- *Conditionally deletes.*

Constructors and destructors

- CoinArrayWithLength ()
- *Default constructor - NULL.*
- CoinArrayWithLength (int size)
- *Alternate Constructor - length in bytes - size_ -1.*
- CoinArrayWithLength (int size, int mode)
- *Alternate Constructor - length in bytes mode - 0 size_ set to size mode > 0 size_ set to size and zeroed if size <= 0 just does alignment If abs(mode) > 2 then align on that as power of 2.*
- CoinArrayWithLength (const CoinArrayWithLength &rhs)
- *Copy constructor.*
- CoinArrayWithLength (const CoinArrayWithLength *rhs)
- *Copy constructor.2.*
- CoinArrayWithLength & operator= (const CoinArrayWithLength &rhs)
- *Assignment operator.*
- void `copy` (const CoinArrayWithLength &rhs, int numberBytes=-1)
- *Assignment with length (if -1 use internal length)*
- void `allocate` (const CoinArrayWithLength &rhs, int numberBytes)

- Assignment with length - does not copy.*
- `~CoinArrayWithLength ()`
Destructor.
- `void getArray (int size)`
Get array with alignment.
- `void reallyFreeArray ()`
Really get rid of array with alignment.
- `void getCapacity (int numberBytes, int numberIfNeeded=-1)`
Get enough space (if more needed then do at least needed)

Protected Attributes

Private member data

- `char * array_`
Array.
- `CoinBigIndex size_`
Size of array in bytes.
- `int offset_`
Offset of array.
- `int alignment_`
Alignment wanted (power of 2)

9.9.1 Detailed Description

Pointer with length in bytes.

This has a pointer to an array and the number of bytes in array. If number of bytes== -1 then CoinConditionalNew deletes existing pointer and returns new pointer of correct size (and number bytes still -1). CoinConditionalDelete deletes existing pointer and NULLs it. So behavior is as normal (apart from New deleting pointer which will have no effect with good coding practices. If number of bytes >=0 then CoinConditionalNew just returns existing pointer if array big enough otherwise deletes existing pointer, allocates array with spare 1%+64 bytes and updates number of bytes CoinConditionalDelete sets number of bytes = -size-2 and then array returns NULL

Definition at line 511 of file CoinIndexedVector.hpp.

9.9.2 Constructor & Destructor Documentation

9.9.2.1 CoinArrayWithLength::CoinArrayWithLength () [inline]

Default constructor - NULL.

Definition at line 568 of file CoinIndexedVector.hpp.

9.9.2.2 CoinArrayWithLength::CoinArrayWithLength (int size) [inline]

Alternate Constructor - length in bytes - size_ -1.

Definition at line 572 of file CoinIndexedVector.hpp.

9.9.2.3 CoinArrayWithLength::CoinArrayWithLength (int size, int mode)

Alternate Constructor - length in bytes mode - 0 size_ set to size mode>0 size_ set to size and zeroed if size<=0 just does alignment If abs(mode) >2 then align on that as power of 2.

9.9.2.4 CoinArrayWithLength::CoinArrayWithLength (const CoinArrayWithLength & rhs)

Copy constructor.

9.9.2.5 CoinArrayWithLength::CoinArrayWithLength (const CoinArrayWithLength * rhs)

Copy constructor.2.

9.9.2.6 CoinArrayWithLength::~~CoinArrayWithLength ()

Destructor.

9.9.3 Member Function Documentation

9.9.3.1 int CoinArrayWithLength::getSize () const [inline]

Get the size.

Definition at line 517 of file CoinIndexedVector.hpp.

9.9.3.2 int CoinArrayWithLength::rawSize () const [inline]

Get the size.

Definition at line 520 of file CoinIndexedVector.hpp.

9.9.3.3 bool CoinArrayWithLength::switchedOn () const [inline]

See if persistence already on.

Definition at line 523 of file CoinIndexedVector.hpp.

9.9.3.4 int CoinArrayWithLength::capacity () const [inline]

Get the capacity (just read it)

Definition at line 526 of file CoinIndexedVector.hpp.

9.9.3.5 void CoinArrayWithLength::setCapacity () [inline]

Set the capacity to ≥ 0 if ≤ -2 .

Definition at line 529 of file CoinIndexedVector.hpp.

9.9.3.6 const char* CoinArrayWithLength::array () const [inline]

Get Array.

Definition at line 532 of file CoinIndexedVector.hpp.

9.9.3.7 void CoinArrayWithLength::setSize (int value) [inline]

Set the size.

Definition at line 539 of file CoinIndexedVector.hpp.

9.9.3.8 void CoinArrayWithLength::switchOff () [inline]

Set the size to -1.

Definition at line 542 of file CoinIndexedVector.hpp.

9.9.3.9 `void CoinArrayWithLength::switchOn (int alignment = 3) [inline]`

Set the size to -2 and alignment.

Definition at line 545 of file CoinIndexedVector.hpp.

9.9.3.10 `void CoinArrayWithLength::setPersistence (int flag, int currentLength)`

Does what is needed to set persistence.

9.9.3.11 `void CoinArrayWithLength::clear ()`

Zero out array.

9.9.3.12 `void CoinArrayWithLength::swap (CoinArrayWithLength & other)`

Swaps memory between two members.

9.9.3.13 `void CoinArrayWithLength::extend (int newSize)`

Extend a persistent array keeping data (size in bytes)

9.9.3.14 `char* CoinArrayWithLength::conditionalNew (long sizeWanted)`

Conditionally gets new array.

9.9.3.15 `void CoinArrayWithLength::conditionalDelete ()`

Conditionally deletes.

9.9.3.16 `CoinArrayWithLength& CoinArrayWithLength::operator= (const CoinArrayWithLength & rhs)`

Assignment operator.

9.9.3.17 `void CoinArrayWithLength::copy (const CoinArrayWithLength & rhs, int numberBytes = -1)`

Assignment with length (if -1 use internal length)

9.9.3.18 `void CoinArrayWithLength::allocate (const CoinArrayWithLength & rhs, int numberBytes)`

Assignment with length - does not copy.

9.9.3.19 `void CoinArrayWithLength::getArray (int size)`

Get array with alignment.

9.9.3.20 `void CoinArrayWithLength::reallyFreeArray ()`

Really get rid of array with alignment.

9.9.3.21 `void CoinArrayWithLength::getCapacity (int numberBytes, int numberIfNeeded = -1)`

Get enough space (if more needed then do at least needed)

9.9.4 Member Data Documentation

9.9.4.1 `char* CoinArrayWithLength::array_` [protected]

Array.

Definition at line 606 of file `CoinIndexedVector.hpp`.

9.9.4.2 `CoinBigIndex CoinArrayWithLength::size_` [protected]

Size of array in bytes.

Definition at line 608 of file `CoinIndexedVector.hpp`.

9.9.4.3 `int CoinArrayWithLength::offset_` [protected]

Offset of array.

Definition at line 610 of file `CoinIndexedVector.hpp`.

9.9.4.4 `int CoinArrayWithLength::alignment_` [protected]

Alignment wanted (power of 2)

Definition at line 612 of file `CoinIndexedVector.hpp`.

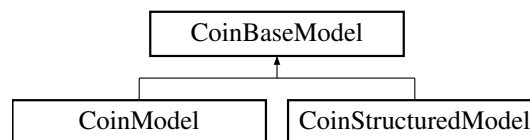
The documentation for this class was generated from the following file:

- `/home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp`

9.10 CoinBaseModel Class Reference

```
#include <CoinModel.hpp>
```

Inheritance diagram for `CoinBaseModel`:



Public Member Functions

Constructors, destructor

- `CoinBaseModel ()`
Default Constructor.
- `CoinBaseModel (const CoinBaseModel &rhs)`
Copy constructor.
- `CoinBaseModel & operator= (const CoinBaseModel &rhs)`
Assignment operator.
- `virtual CoinBaseModel * clone () const =0`
Clone.
- `virtual ~CoinBaseModel ()`
Destructor.

For getting information

- int [numberRows](#) () const
Return number of rows.
- int [numberColumns](#) () const
Return number of columns.
- virtual [CoinBigIndex](#) [numberElements](#) () const =0
Return number of elements.
- double [objectiveOffset](#) () const
Returns the (constant) objective offset This is the RHS entry for the objective row.
- void [setObjectiveOffset](#) (double value)
Set objective offset.
- double [optimizationDirection](#) () const
Direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.
- void [setOptimizationDirection](#) (double value)
Set direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.
- int [logLevel](#) () const
Get print level 0 - off, 1 - errors, 2 - more.
- void [setLogLevel](#) (int value)
Set print level 0 - off, 1 - errors, 2 - more.
- const char * [getProblemName](#) () const
Return the problem name.
- void [setProblemName](#) (const char *name)
Set problem name.
- void [setProblemName](#) (const std::string &name)
Set problem name.
- const std::string & [getRowBlock](#) () const
Return the row block name.
- void [setRowBlock](#) (const std::string &name)
Set row block name.
- const std::string & [getColumnBlock](#) () const
Return the column block name.
- void [setColumnBlock](#) (const std::string &name)
Set column block name.

Protected Attributes

Data members

- int [numberRows_](#)
Current number of rows.
- int [numberColumns_](#)
Current number of columns.
- double [optimizationDirection_](#)
Direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.
- double [objectiveOffset_](#)
Objective offset to be passed on.
- std::string [problemName_](#)
Problem name.
- std::string [rowBlockName_](#)
Rowblock name.
- std::string [columnBlockName_](#)
Columnblock name.
- int [logLevel_](#)
Print level.

9.10.1 Detailed Description

Definition at line 12 of file CoinModel.hpp.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 CoinBaseModel::CoinBaseModel ()

Default Constructor.

9.10.2.2 CoinBaseModel::CoinBaseModel (const CoinBaseModel & rhs)

Copy constructor.

9.10.2.3 virtual CoinBaseModel::~CoinBaseModel () [virtual]

Destructor.

9.10.3 Member Function Documentation

9.10.3.1 CoinBaseModel& CoinBaseModel::operator= (const CoinBaseModel & rhs)

Assignment operator.

9.10.3.2 virtual CoinBaseModel* CoinBaseModel::clone () const [pure virtual]

Clone.

Implemented in [CoinModel](#), and [CoinStructuredModel](#).

9.10.3.3 int CoinBaseModel::numberOfRows () const [inline]

Return number of rows.

Definition at line 38 of file CoinModel.hpp.

9.10.3.4 int CoinBaseModel::numberOfColumns () const [inline]

Return number of columns.

Definition at line 41 of file CoinModel.hpp.

9.10.3.5 virtual CoinBigIndex CoinBaseModel::numberOfElements () const [pure virtual]

Return number of elements.

Implemented in [CoinModel](#), and [CoinStructuredModel](#).

9.10.3.6 double CoinBaseModel::objectiveOffset () const [inline]

Returns the (constant) objective offset This is the RHS entry for the objective row.

Definition at line 48 of file CoinModel.hpp.

9.10.3.7 void CoinBaseModel::setObjectiveOffset (double value) [inline]

Set objective offset.

Definition at line 51 of file CoinModel.hpp.

9.10.3.8 `double CoinBaseModel::optimizationDirection () const [inline]`

Direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.

Definition at line 54 of file CoinModel.hpp.

9.10.3.9 `void CoinBaseModel::setOptimizationDirection (double value) [inline]`

Set direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.

Definition at line 58 of file CoinModel.hpp.

9.10.3.10 `int CoinBaseModel::logLevel () const [inline]`

Get print level 0 - off, 1 - errors, 2 - more.

Definition at line 61 of file CoinModel.hpp.

9.10.3.11 `void CoinBaseModel::setLogLevel (int value)`

Set print level 0 - off, 1 - errors, 2 - more.

9.10.3.12 `const char* CoinBaseModel::getProblemName () const [inline]`

Return the problem name.

Definition at line 66 of file CoinModel.hpp.

9.10.3.13 `void CoinBaseModel::setProblemName (const char * name)`

Set problem name.

9.10.3.14 `void CoinBaseModel::setProblemName (const std::string & name)`

Set problem name.

9.10.3.15 `const std::string& CoinBaseModel::getRowBlock () const [inline]`

Return the row block name.

Definition at line 73 of file CoinModel.hpp.

9.10.3.16 `void CoinBaseModel::setRowBlock (const std::string & name) [inline]`

Set row block name.

Definition at line 76 of file CoinModel.hpp.

9.10.3.17 `const std::string& CoinBaseModel::getColumnBlock () const [inline]`

Return the column block name.

Definition at line 79 of file CoinModel.hpp.

9.10.3.18 `void CoinBaseModel::setColumnBlock (const std::string & name) [inline]`

Set column block name.

Definition at line 82 of file CoinModel.hpp.

9.10.4 Member Data Documentation

9.10.4.1 `int CoinBaseModel::numberOfRows_` [protected]

Current number of rows.

Definition at line 90 of file `CoinModel.hpp`.

9.10.4.2 `int CoinBaseModel::numberOfColumns_` [protected]

Current number of columns.

Definition at line 92 of file `CoinModel.hpp`.

9.10.4.3 `double CoinBaseModel::optimizationDirection_` [protected]

Direction of optimization (1 - minimize, -1 - maximize, 0 - ignore).

Definition at line 94 of file `CoinModel.hpp`.

9.10.4.4 `double CoinBaseModel::objectiveOffset_` [protected]

Objective offset to be passed on.

Definition at line 96 of file `CoinModel.hpp`.

9.10.4.5 `std::string CoinBaseModel::problemName_` [protected]

Problem name.

Definition at line 98 of file `CoinModel.hpp`.

9.10.4.6 `std::string CoinBaseModel::rowBlockName_` [protected]

Rowblock name.

Definition at line 100 of file `CoinModel.hpp`.

9.10.4.7 `std::string CoinBaseModel::columnBlockName_` [protected]

Columnblock name.

Definition at line 102 of file `CoinModel.hpp`.

9.10.4.8 `int CoinBaseModel::logLevel_` [protected]

Print level.

I could have gone for full message handling but this should normally be silent and lightweight. I can always change. 0 - no output 1 - on errors 2 - more detailed

Definition at line 110 of file `CoinModel.hpp`.

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

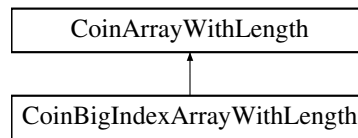
- `/home/ted/COIN/trunk/CoinUtils/src/CoinModel.hpp`

9.11 `CoinBigIndexArrayWithLength` Class Reference

`CoinBigIndex` * version.

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for CoinBigIndexArrayWithLength:



Public Member Functions

Get methods.

- `int getSize () const`
Get the size.
- `CoinBigIndex * array () const`
Get Array.

Set methods

- `void setSize (int value)`
Set the size.

Condition methods

- `CoinBigIndex * conditionalNew (int sizeWanted)`
Conditionally gets new array.

Constructors and destructors

- `CoinBigIndexArrayWithLength ()`
Default constructor - NULL.
- `CoinBigIndexArrayWithLength (int size)`
Alternate Constructor - length in bytes - size_ - 1.
- `CoinBigIndexArrayWithLength (int size, int mode)`
Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.
- `CoinBigIndexArrayWithLength (const CoinBigIndexArrayWithLength &rhs)`
Copy constructor.
- `CoinBigIndexArrayWithLength (const CoinBigIndexArrayWithLength *rhs)`
Copy constructor.2.
- `CoinBigIndexArrayWithLength & operator= (const CoinBigIndexArrayWithLength &rhs)`
Assignment operator.

Additional Inherited Members

9.11.1 Detailed Description

CoinBigIndex * version.

Definition at line 833 of file `CoinIndexedVector.hpp`.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 `CoinBigIndexArrayWithLength::CoinBigIndexArrayWithLength ()` `[inline]`

Default constructor - NULL.

Definition at line 863 of file `CoinIndexedVector.hpp`.

9.11.2.2 `CoinBigIndexArrayWithLength::CoinBigIndexArrayWithLength (int size)` `[inline]`

Alternate Constructor - length in bytes - size_ -1.

Definition at line 866 of file `CoinIndexedVector.hpp`.

9.11.2.3 `CoinBigIndexArrayWithLength::CoinBigIndexArrayWithLength (int size, int mode)` `[inline]`

Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.

Definition at line 872 of file `CoinIndexedVector.hpp`.

9.11.2.4 `CoinBigIndexArrayWithLength::CoinBigIndexArrayWithLength (const CoinBigIndexArrayWithLength & rhs)` `[inline]`

Copy constructor.

Definition at line 875 of file `CoinIndexedVector.hpp`.

9.11.2.5 `CoinBigIndexArrayWithLength::CoinBigIndexArrayWithLength (const CoinBigIndexArrayWithLength * rhs)` `[inline]`

Copy constructor.2.

Definition at line 878 of file `CoinIndexedVector.hpp`.

9.11.3 Member Function Documentation

9.11.3.1 `int CoinBigIndexArrayWithLength::getSize () const` `[inline]`

Get the size.

Definition at line 839 of file `CoinIndexedVector.hpp`.

9.11.3.2 `CoinBigIndex* CoinBigIndexArrayWithLength::array () const` `[inline]`

Get Array.

Definition at line 842 of file `CoinIndexedVector.hpp`.

9.11.3.3 `void CoinBigIndexArrayWithLength::setSize (int value)` `[inline]`

Set the size.

Definition at line 849 of file `CoinIndexedVector.hpp`.

9.11.3.4 `CoinBigIndex* CoinBigIndexArrayWithLength::conditionalNew (int sizeWanted)` `[inline]`

Conditionally gets new array.

Definition at line 856 of file `CoinIndexedVector.hpp`.

9.11.3.5 CoinBigIndexArrayWithLength& CoinBigIndexArrayWithLength::operator= (const CoinBigIndexArrayWithLength & rhs) [inline]

Assignment operator.

Definition at line 881 of file CoinIndexedVector.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp>

9.12 CoinBuild Class Reference

In many cases it is natural to build a model by adding one row at a time.

```
#include <CoinBuild.hpp>
```

Public Member Functions

Useful methods

- [void addRow](#) (int numberInRow, const int *columns, const double *elements, double rowLower=-[COIN_DBL_MAX](#), double rowUpper=[COIN_DBL_MAX](#))
add a row
- [void addColumn](#) (int numberInColumn, const int *rows, const double *elements, double columnLower=0.0, double columnUpper=[COIN_DBL_MAX](#), double objectiveValue=0.0)
add a column
- [void addCol](#) (int numberInColumn, const int *rows, const double *elements, double columnLower=0.0, double columnUpper=[COIN_DBL_MAX](#), double objectiveValue=0.0)
add a column
- int [numberRows](#) () const
Return number of rows or maximum found so far.
- int [numberColumns](#) () const
Return number of columns or maximum found so far.
- [CoinBigIndex numberElements](#) () const
Return number of elements.
- int [row](#) (int whichRow, double &rowLower, double &rowUpper, const int *&indices, const double *&elements) const
Returns number of elements in a row and information in row.
- int [currentRow](#) (double &rowLower, double &rowUpper, const int *&indices, const double *&elements) const
Returns number of elements in current row and information in row Used as rows may be stored in a chain.
- [void setCurrentRow](#) (int whichRow)
Set current row.
- int [currentRow](#) () const
Returns current row number.
- int [column](#) (int whichColumn, double &columnLower, double &columnUpper, double &objectiveValue, const int *&indices, const double *&elements) const
Returns number of elements in a column and information in column.
- int [currentColumn](#) (double &columnLower, double &columnUpper, double &objectiveValue, const int *&indices, const double *&elements) const
Returns number of elements in current column and information in column Used as columns may be stored in a chain.
- [void setCurrentColumn](#) (int whichColumn)
Set current column.
- int [currentColumn](#) () const
Returns current column number.

- `int type () const`
Returns type.

Constructors, destructor

- `CoinBuild ()`
Default constructor.
- `CoinBuild (int type)`
Constructor with type 0==for addRow, 1== for addColumn.
- `~CoinBuild ()`
Destructor.

Copy method

- `CoinBuild (const CoinBuild &)`
The copy constructor.
- `CoinBuild & operator= (const CoinBuild &)`
=

9.12.1 Detailed Description

In many cases it is natural to build a model by adding one row at a time.

In `Coin` this is inefficient so this class gives some help. An instance of `CoinBuild` can be built up more efficiently and then added to the `Clp/OsiModel` in one go.

It may be more efficient to have fewer arrays and re-allocate them but this should give a large gain over `addRow`.

I have now extended it to columns.

Definition at line 27 of file `CoinBuild.hpp`.

9.12.2 Constructor & Destructor Documentation

9.12.2.1 `CoinBuild::CoinBuild ()`

Default constructor.

9.12.2.2 `CoinBuild::CoinBuild (int type)`

Constructor with type 0==for `addRow`, 1== for `addColumn`.

9.12.2.3 `CoinBuild::~~CoinBuild ()`

Destructor.

9.12.2.4 `CoinBuild::CoinBuild (const CoinBuild &)`

The copy constructor.

9.12.3 Member Function Documentation

9.12.3.1 `void CoinBuild::addRow (int numberInRow, const int * columns, const double * elements, double rowLower = -COIN_DBL_MAX, double rowUpper = COIN_DBL_MAX)`

add a row

9.12.3.2 **void** CoinBuild::addColumn (int *numberInColumn*, const int * *rows*, const double * *elements*, double *columnLower* = 0.0, double *columnUpper* = COIN_DBL_MAX, double *objectiveValue* = 0.0)

add a column

9.12.3.3 **void** CoinBuild::addCol (int *numberInColumn*, const int * *rows*, const double * *elements*, double *columnLower* = 0.0, double *columnUpper* = COIN_DBL_MAX, double *objectiveValue* = 0.0) [inline]

add a column

Definition at line 42 of file CoinBuild.hpp.

9.12.3.4 **int** CoinBuild::numberOfRows () const [inline]

Return number of rows or maximum found so far.

Definition at line 48 of file CoinBuild.hpp.

9.12.3.5 **int** CoinBuild::numberOfColumns () const [inline]

Return number of columns or maximum found so far.

Definition at line 51 of file CoinBuild.hpp.

9.12.3.6 **CoinBigIndex** CoinBuild::numberOfElements () const [inline]

Return number of elements.

Definition at line 54 of file CoinBuild.hpp.

9.12.3.7 **int** CoinBuild::row (int *whichRow*, double & *rowLower*, double & *rowUpper*, const int *& *indices*, const double *& *elements*) const

Returns number of elements in a row and information in row.

9.12.3.8 **int** CoinBuild::currentRow (double & *rowLower*, double & *rowUpper*, const int *& *indices*, const double *& *elements*) const

Returns number of elements in current row and information in row Used as rows may be stored in a chain.

9.12.3.9 **void** CoinBuild::setCurrentRow (int *whichRow*)

Set current row.

9.12.3.10 **int** CoinBuild::currentColumn () const

Returns current row number.

9.12.3.11 **int** CoinBuild::column (int *whichColumn*, double & *columnLower*, double & *columnUpper*, double & *objectiveValue*, const int *& *indices*, const double *& *elements*) const

Returns number of elements in a column and information in column.

9.12.3.12 **int** CoinBuild::currentColumn (double & *columnLower*, double & *columnUpper*, double & *objectiveValue*, const int *& *indices*, const double *& *elements*) const

Returns number of elements in current column and information in column Used as columns may be stored in a chain.

9.12.3.13 void CoinBuild::setCurrentColumn (int *whichColumn*)

Set current column.

9.12.3.14 int CoinBuild::currentColumn () const

Returns current column number.

9.12.3.15 int CoinBuild::type () const [inline]

Returns type.

Definition at line 84 of file CoinBuild.hpp.

9.12.3.16 CoinBuild& CoinBuild::operator= (const CoinBuild &)

=

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

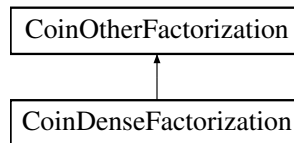
- </home/ted/COIN/trunk/CoinUtils/src/CoinBuild.hpp>

9.13 CoinDenseFactorization Class Reference

This deals with Factorization and Updates This is a simple dense version so other people can write a better one.

```
#include <CoinDenseFactorization.hpp>
```

Inheritance diagram for CoinDenseFactorization:



Public Member Functions

- [void gutsOfDestructor \(\)](#)
The real work of destructor.
- [void gutsOfInitialize \(\)](#)
The real work of constructor.
- [void gutsOfCopy \(const \[CoinDenseFactorization\]\(#\) &other\)](#)
The real work of copy.

Constructors and destructor and copy

- [CoinDenseFactorization \(\)](#)
Default constructor.
- [CoinDenseFactorization \(const \[CoinDenseFactorization\]\(#\) &other\)](#)
Copy constructor.
- [virtual ~CoinDenseFactorization \(\)](#)
Destructor.
- [CoinDenseFactorization & operator= \(const \[CoinDenseFactorization\]\(#\) &other\)](#)

- = copy*
- virtual [CoinOtherFactorization](#) * [clone](#) () const
Clone.

Do factorization - public

- virtual void [getAreas](#) (int [numberRows](#), int [numberColumns](#), [CoinBigIndex](#) maximumL, [CoinBigIndex](#) maximumU)
Gets space for a factorization.
- virtual void [preProcess](#) ()
PreProcesses column ordered copy of basis.
- virtual int [factor](#) ()
Does most of factorization returning status 0 - OK -99 - needs more memory -1 - singular - use numberGoodColumns and redo.
- virtual void [postProcess](#) (const int *sequence, int *pivotVariable)
Does post processing on valid factorization - putting variables on correct rows.
- virtual void [makeNonSingular](#) (int *sequence, int [numberColumns](#))
Makes a non-singular basis by replacing variables.

general stuff such as number of elements

- virtual int [numberElements](#) () const
Total number of elements in factorization.
- double [maximumCoefficient](#) () const
Returns maximum absolute value in factorization.

rank one updates which do exist

- virtual int [replaceColumn](#) ([CoinIndexedVector](#) *regionSparse, int [pivotRow](#), double pivotCheck, bool checkBeforeModifying=false, double acceptablePivot=1.0e-8)
Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If checkBeforeModifying is true will do all accuracy checks before modifying factorization.

various uses of factorization (return code number elements)

which user may want to know about

- virtual int [updateColumnFT](#) ([CoinIndexedVector](#) *regionSparse, [CoinIndexedVector](#) *regionSparse2, bool=false)
Updates one column (FTRAN) from regionSparse2 Tries to do FT update number returned is negative if no room regionSparse starts as zero and is zero at end.
- virtual int [updateColumn](#) ([CoinIndexedVector](#) *regionSparse, [CoinIndexedVector](#) *regionSparse2, bool noPermute=false) const
This version has same effect as above with FTUpdate==false so number returned is always >=0.
- virtual int [updateTwoColumnsFT](#) ([CoinIndexedVector](#) *regionSparse1, [CoinIndexedVector](#) *regionSparse2, [CoinIndexedVector](#) *regionSparse3, bool noPermute=false)
does FTRAN on two columns
- virtual int [updateColumnTranspose](#) ([CoinIndexedVector](#) *regionSparse, [CoinIndexedVector](#) *regionSparse2) const
Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if regionSparse2 packed on input - will be packed on output.

various uses of factorization

*** *Below this user may not want to know about*

which user may not want to know about (left over from my LP code)

- `void clearArrays ()`
Get rid of all memory.
- `virtual int * indices () const`
Returns array to put basis indices in.
- `virtual int * permute () const`
Returns permute in.

Protected Member Functions

- `int checkPivot (double saveFromU, double oldPivot) const`
Returns accuracy status of replaceColumn returns 0=OK, 1=Probably OK, 2=singular.

Friends

- `void CoinDenseFactorizationUnitTest (const std::string &mpsDir)`

Additional Inherited Members

9.13.1 Detailed Description

This deals with Factorization and Updates This is a simple dense version so other people can write a better one.

I am assuming that 32 bits is enough for number of rows or columns, but `CoinBigIndex` may be redefined to get 64 bits.

Definition at line 282 of file `CoinDenseFactorization.hpp`.

9.13.2 Constructor & Destructor Documentation

9.13.2.1 `CoinDenseFactorization::CoinDenseFactorization ()`

Default constructor.

9.13.2.2 `CoinDenseFactorization::CoinDenseFactorization (const CoinDenseFactorization & other)`

Copy constructor.

9.13.2.3 `virtual CoinDenseFactorization::~~CoinDenseFactorization () [virtual]`

Destructor.

9.13.3 Member Function Documentation

9.13.3.1 `CoinDenseFactorization& CoinDenseFactorization::operator= (const CoinDenseFactorization & other)`

= copy

9.13.3.2 `virtual CoinOtherFactorization* CoinDenseFactorization::clone () const [virtual]`

Clone.

Implements [CoinOtherFactorization](#).

9.13.3.3 `virtual void CoinDenseFactorization::getAreas (int numberRows, int numberColumns, CoinBigIndex maximumL, CoinBigIndex maximumU) [virtual]`

Gets space for a factorization.

Implements [CoinOtherFactorization](#).

9.13.3.4 `virtual void CoinDenseFactorization::preProcess () [virtual]`

PreProcesses column ordered copy of basis.

Implements [CoinOtherFactorization](#).

9.13.3.5 `virtual int CoinDenseFactorization::factor () [virtual]`

Does most of factorization returning status 0 - OK -99 - needs more memory -1 - singular - use `numberGoodColumns` and redo.

Implements [CoinOtherFactorization](#).

9.13.3.6 `virtual void CoinDenseFactorization::postProcess (const int * sequence, int * pivotVariable) [virtual]`

Does post processing on valid factorization - putting variables on correct rows.

Implements [CoinOtherFactorization](#).

9.13.3.7 `virtual void CoinDenseFactorization::makeNonSingular (int * sequence, int numberColumns) [virtual]`

Makes a non-singular basis by replacing variables.

Implements [CoinOtherFactorization](#).

9.13.3.8 `virtual int CoinDenseFactorization::numberElements () const [inline],[virtual]`

Total number of elements in factorization.

Implements [CoinOtherFactorization](#).

Definition at line 327 of file `CoinDenseFactorization.hpp`.

9.13.3.9 `double CoinDenseFactorization::maximumCoefficient () const`

Returns maximum absolute value in factorization.

9.13.3.10 `virtual int CoinDenseFactorization::replaceColumn (CoinIndexedVector * regionSparse, int pivotRow, double pivotCheck, bool checkBeforeModifying = false, double acceptablePivot = 1.0e-8) [virtual]`

Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If `checkBeforeModifying` is true will do all accuracy checks before modifying factorization.

Whether to set this depends on speed considerations. You could just do this on first iteration after factorization and thereafter re-factorize partial update already in U

Implements [CoinOtherFactorization](#).

9.13.3.11 `virtual int CoinDenseFactorization::updateColumnFT (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool = false) [inline],[virtual]`

Updates one column (FTRAN) from `regionSparse2` Tries to do FT update number returned is negative if no room `regionSparse` starts as zero and is zero at end.

Note - if `regionSparse2` packed on input - will be packed on output

Implements [CoinOtherFactorization](#).

Definition at line 360 of file CoinDenseFactorization.hpp.

```
9.13.3.12 virtual int CoinDenseFactorization::updateColumn ( CoinIndexedVector * regionSparse, CoinIndexedVector *
               regionSparse2, bool noPermute = false ) const [virtual]
```

This version has same effect as above with FTUpdate==false so number returned is always >=0.

Implements [CoinOtherFactorization](#).

```
9.13.3.13 virtual int CoinDenseFactorization::updateTwoColumnsFT ( CoinIndexedVector * regionSparse1,
               CoinIndexedVector * regionSparse2, CoinIndexedVector * regionSparse3, bool noPermute = false )
               [virtual]
```

does FTRAN on two columns

Implements [CoinOtherFactorization](#).

```
9.13.3.14 virtual int CoinDenseFactorization::updateColumnTranspose ( CoinIndexedVector * regionSparse,
               CoinIndexedVector * regionSparse2 ) const [virtual]
```

Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if region-Sparse2 packed on input - will be packed on output.

Implements [CoinOtherFactorization](#).

```
9.13.3.15 void CoinDenseFactorization::clearArrays ( ) [inline],[virtual]
```

Get rid of all memory.

Reimplemented from [CoinOtherFactorization](#).

Definition at line 387 of file CoinDenseFactorization.hpp.

```
9.13.3.16 virtual int* CoinDenseFactorization::indices ( ) const [inline],[virtual]
```

Returns array to put basis indices in.

Implements [CoinOtherFactorization](#).

Definition at line 390 of file CoinDenseFactorization.hpp.

```
9.13.3.17 virtual int* CoinDenseFactorization::permute ( ) const [inline],[virtual]
```

Returns permute in.

Implements [CoinOtherFactorization](#).

Definition at line 393 of file CoinDenseFactorization.hpp.

```
9.13.3.18 void CoinDenseFactorization::gutsOfDestructor ( )
```

The real work of desstructor.

```
9.13.3.19 void CoinDenseFactorization::gutsOfInitialize ( )
```

The real work of constructor.

```
9.13.3.20 void CoinDenseFactorization::gutsOfCopy ( const CoinDenseFactorization & other )
```

The real work of copy.

9.13.3.21 `int CoinDenseFactorization::checkPivot (double saveFromU, double oldPivot) const` `[protected]`

Returns accuracy status of replaceColumn returns 0=OK, 1=Probably OK, 2=singular.

9.13.4 Friends And Related Function Documentation

9.13.4.1 `void CoinDenseFactorizationUnitTest (const std::string & mpsDir)` `[friend]`

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

- `/home/ted/COIN/trunk/CoinUtils/src/CoinDenseFactorization.hpp`

9.14 CoinDenseVector< T > Class Template Reference

Dense Vector.

```
#include <CoinDenseVector.hpp>
```

Public Member Functions

Get methods.

- `int getNumElements () const`
Get the size.
- `int size () const`
- `const T * getElements () const`
Get element values.
- `T * getElements ()`
Get element values.

Set methods

- `void clear ()`
Reset the vector (i.e. set all elements to zero)
- `CoinDenseVector & operator= (const CoinDenseVector &)`
Assignment operator.
- `T & operator[] (int index) const`
Member of array operator.
- `void setVector (int size, const T *elems)`
Set vector size, and elements.
- `void setConstant (int size, T elems)`
Elements set to have the same scalar value.
- `void setElement (int index, T element)`
Set an existing element in the dense vector The first argument is the "index" into the elements() array.
- `void resize (int newSize, T fill=T())`
Resize the dense vector to be the first newSize elements.
- `void append (const CoinDenseVector &)`
Append a dense vector to this dense vector.

norms, sum and scale

- `T oneNorm () const`
1-norm of vector

- double `twoNorm ()` const
2-norm of vector
- T `infNorm ()` const
infinity-norm of vector
- T `sum ()` const
sum of vector elements
- void `scale (T factor)`
scale vector elements

Arithmetic operators.

- void `operator+= (T value)`
add value to every entry
- void `operator-= (T value)`
subtract value from every entry
- void `operator*= (T value)`
multiply every entry by value
- void `operator/= (T value)`
divide every entry by value

Constructors and destructors

- `CoinDenseVector ()`
Default constructor.
- `CoinDenseVector (int size, const T *elems)`
Alternate Constructors - set elements to vector of Ts.
- `CoinDenseVector (int size, T element=T())`
Alternate Constructors - set elements to same scalar value.
- `CoinDenseVector (const CoinDenseVector &)`
Copy constructors.
- `~CoinDenseVector ()`
Destructor.

9.14.1 Detailed Description

```
template<typename T> class CoinDenseVector< T >
```

Dense Vector.

Stores a dense (or expanded) vector of floating point values. Type of vector elements is controlled by templating. (Some working quantities such as accumulated sums are explicitly declared of type double). This allows the components of the vector integer, single or double precision.

Here is a sample usage:

```
const int ne = 4;
double el[ne] = { 10., 40., 1., 50. }

// Create vector and set its value
CoinDenseVector<double> r(ne,el);

// access each element
assert( r.getElements()[0]==10. );
assert( r.getElements()[1]==40. );
assert( r.getElements()[2]== 1. );
assert( r.getElements()[3]==50. );
```

```
// Test for equality
CoinDenseVector<double> r1;
r1=r;

// Add dense vectors.
// Similarly for subtraction, multiplication,
// and division.
CoinDenseVector<double> add = r + r1;
assert( add[0] == 10.+10. );
assert( add[1] == 40.+40. );
assert( add[2] == 1.+ 1. );
assert( add[3] == 50.+50. );

assert( r.sum() == 10.+40.+1.+50. );
```

Definition at line 67 of file CoinDenseVector.hpp.

9.14.2 Constructor & Destructor Documentation

9.14.2.1 `template<typename T> CoinDenseVector< T >::CoinDenseVector ()`

Default constructor.

9.14.2.2 `template<typename T> CoinDenseVector< T >::CoinDenseVector (int size, const T * elems)`

Alternate Constructors - set elements to vector of Ts.

9.14.2.3 `template<typename T> CoinDenseVector< T >::CoinDenseVector (int size, T element = T ())`

Alternate Constructors - set elements to same scalar value.

9.14.2.4 `template<typename T> CoinDenseVector< T >::CoinDenseVector (const CoinDenseVector< T > &)`

Copy constructors.

9.14.2.5 `template<typename T> CoinDenseVector< T >::~~CoinDenseVector ()`

Destructor.

9.14.3 Member Function Documentation

9.14.3.1 `template<typename T> int CoinDenseVector< T >::getNumElements () const [inline]`

Get the size.

Definition at line 81 of file CoinDenseVector.hpp.

9.14.3.2 `template<typename T> int CoinDenseVector< T >::size () const [inline]`

Definition at line 82 of file CoinDenseVector.hpp.

9.14.3.3 `template<typename T> const T* CoinDenseVector< T >::getElements () const [inline]`

Get element values.

Definition at line 84 of file CoinDenseVector.hpp.

9.14.3.4 `template<typename T> T* CoinDenseVector< T >::getElements () [inline]`

Get element values.

Definition at line 86 of file CoinDenseVector.hpp.

9.14.3.5 `template<typename T> void CoinDenseVector< T >::clear ()`

Reset the vector (i.e. set all elements to zero)

9.14.3.6 `template<typename T> CoinDenseVector& CoinDenseVector< T >::operator= (const CoinDenseVector< T > &)`

Assignment operator.

9.14.3.7 `template<typename T> T& CoinDenseVector< T >::operator[] (int index) const`

Member of array operator.

9.14.3.8 `template<typename T> void CoinDenseVector< T >::setVector (int size, const T * elems)`

Set vector size, and elements.

Size is the length of the elements vector. The element vector is copied into this class instance's member data.

9.14.3.9 `template<typename T> void CoinDenseVector< T >::setConstant (int size, T elems)`

Elements set to have the same scalar value.

9.14.3.10 `template<typename T> void CoinDenseVector< T >::setElement (int index, T element)`

Set an existing element in the dense vector The first argument is the "index" into the elements() array.

9.14.3.11 `template<typename T> void CoinDenseVector< T >::resize (int newSize, T fill = T ())`

Resize the dense vector to be the first newSize elements.

If length is decreased, vector is truncated. If increased new entries, set to new default element

9.14.3.12 `template<typename T> void CoinDenseVector< T >::append (const CoinDenseVector< T > &)`

Append a dense vector to this dense vector.

9.14.3.13 `template<typename T> T CoinDenseVector< T >::oneNorm () const [inline]`

1-norm of vector

Definition at line 128 of file CoinDenseVector.hpp.

9.14.3.14 `template<typename T> double CoinDenseVector< T >::twoNorm () const [inline]`

2-norm of vector

Definition at line 135 of file CoinDenseVector.hpp.

9.14.3.15 `template<typename T> T CoinDenseVector< T >::infNorm () const [inline]`

infinity-norm of vector

Definition at line 144 of file CoinDenseVector.hpp.

9.14.3.16 `template<typename T> T CoinDenseVector< T >::sum () const` `[inline]`

sum of vector elements

Definition at line 151 of file CoinDenseVector.hpp.

9.14.3.17 `template<typename T> void CoinDenseVector< T >::scale (T factor)` `[inline]`

scale vector elements

Definition at line 158 of file CoinDenseVector.hpp.

9.14.3.18 `template<typename T> void CoinDenseVector< T >::operator+= (T value)`

add *value* to every entry

9.14.3.19 `template<typename T> void CoinDenseVector< T >::operator-= (T value)`

subtract *value* from every entry

9.14.3.20 `template<typename T> void CoinDenseVector< T >::operator*= (T value)`

multiply every entry by *value*

9.14.3.21 `template<typename T> void CoinDenseVector< T >::operator/= (T value)`

divide every entry by *value*

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

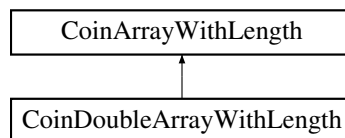
- </home/ted/COIN/trunk/CoinUtils/src/CoinDenseVector.hpp>

9.15 CoinDoubleArrayWithLength Class Reference

double * version

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for CoinDoubleArrayWithLength:



Public Member Functions

Get methods.

- int [getSize](#) () const
Get the size.
- double * [array](#) () const
Get Array.

Set methods

- `void setSize` (int value)
Set the size.

Condition methods

- `double * conditionalNew` (int sizeWanted)
Conditionally gets new array.

Constructors and destructors

- `CoinDoubleArrayWithLength` ()
Default constructor - NULL.
- `CoinDoubleArrayWithLength` (int size)
Alternate Constructor - length in bytes - size_ -1.
- `CoinDoubleArrayWithLength` (int size, int mode)
Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.
- `CoinDoubleArrayWithLength` (const `CoinDoubleArrayWithLength` &rhs)
Copy constructor.
- `CoinDoubleArrayWithLength` (const `CoinDoubleArrayWithLength` *rhs)
Copy constructor.2.
- `CoinDoubleArrayWithLength` & `operator=` (const `CoinDoubleArrayWithLength` &rhs)
Assignment operator.

Additional Inherited Members

9.15.1 Detailed Description

`double * version`

Definition at line 617 of file `CoinIndexedVector.hpp`.

9.15.2 Constructor & Destructor Documentation

9.15.2.1 `CoinDoubleArrayWithLength::CoinDoubleArrayWithLength () [inline]`

Default constructor - NULL.

Definition at line 647 of file `CoinIndexedVector.hpp`.

9.15.2.2 `CoinDoubleArrayWithLength::CoinDoubleArrayWithLength (int size) [inline]`

Alternate Constructor - length in bytes - size_ -1.

Definition at line 650 of file `CoinIndexedVector.hpp`.

9.15.2.3 `CoinDoubleArrayWithLength::CoinDoubleArrayWithLength (int size, int mode) [inline]`

Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.

Definition at line 656 of file `CoinIndexedVector.hpp`.

9.15.2.4 `CoinDoubleArrayWithLength::CoinDoubleArrayWithLength (const CoinDoubleArrayWithLength & rhs) [inline]`

Copy constructor.

Definition at line 659 of file `CoinIndexedVector.hpp`.

9.15.2.5 `CoinDoubleArrayWithLength::CoinDoubleArrayWithLength (const CoinDoubleArrayWithLength * rhs)`
`[inline]`

Copy constructor.2.

Definition at line 662 of file `CoinIndexedVector.hpp`.

9.15.3 Member Function Documentation

9.15.3.1 `int CoinDoubleArrayWithLength::getSize () const` `[inline]`

Get the size.

Definition at line 623 of file `CoinIndexedVector.hpp`.

9.15.3.2 `double* CoinDoubleArrayWithLength::array () const` `[inline]`

Get Array.

Definition at line 626 of file `CoinIndexedVector.hpp`.

9.15.3.3 `void CoinDoubleArrayWithLength::setSize (int value)` `[inline]`

Set the size.

Definition at line 633 of file `CoinIndexedVector.hpp`.

9.15.3.4 `double* CoinDoubleArrayWithLength::conditionalNew (int sizeWanted)` `[inline]`

Conditionally gets new array.

Definition at line 640 of file `CoinIndexedVector.hpp`.

9.15.3.5 `CoinDoubleArrayWithLength& CoinDoubleArrayWithLength::operator= (const CoinDoubleArrayWithLength & rhs)` `[inline]`

Assignment operator.

Definition at line 665 of file `CoinIndexedVector.hpp`.

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

- `/home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp`

9.16 CoinError Class Reference

Error Class thrown by an exception.

```
#include <CoinError.hpp>
```

Public Member Functions

Get error attributes

- `const std::string & message () const`
get message text
- `const std::string & methodName () const`

- *get name of method instantiating error*
• `const std::string & className () const`
get name of class instantiating error (or hint for assert)
- `const std::string & fileName () const`
get name of file for assert
- `int lineNumber () const`
get line number of assert (-1 if not assert)
- `void print (bool doPrint=true) const`
Just print (for asserts)

Constructors and destructors

- `CoinError (std::string message__, std::string methodName__, std::string className__, std::string fileName__ -
=std::string(), int line=-1)`
Alternate Constructor.
- `CoinError (const CoinError &source)`
Copy constructor.
- `CoinError & operator= (const CoinError &rhs)`
Assignment operator.
- `virtual ~CoinError ()`
Destructor.

Static Public Attributes

- `static bool printErrors_`
Whether to print every error.

Friends

- `void CoinErrorUnitTest ()`
A function that tests the methods in the CoinError class.

9.16.1 Detailed Description

Error Class thrown by an exception.

This class is used when exceptions are thrown. It contains:

- message text
- name of method throwing exception
- name of class throwing exception or hint
- name of file if assert
- line number

For asserts class=> optional hint

Definition at line 42 of file CoinError.hpp.

9.16.2 Constructor & Destructor Documentation

9.16.2.1 `CoinError::CoinError (std::string message__, std::string methodName__, std::string className__, std::string fileName_ = std::string(), int line = -1) [inline]`

Alternate Constructor.

Definition at line 99 of file CoinError.hpp.

9.16.2.2 `CoinError::CoinError (const CoinError & source) [inline]`

Copy constructor.

Definition at line 116 of file CoinError.hpp.

9.16.2.3 `virtual CoinError::~~CoinError () [inline],[virtual]`

Destructor.

Definition at line 141 of file CoinError.hpp.

9.16.3 Member Function Documentation

9.16.3.1 `const std::string& CoinError::message () const [inline]`

get message text

Definition at line 65 of file CoinError.hpp.

9.16.3.2 `const std::string& CoinError::methodName () const [inline]`

get name of method instantiating error

Definition at line 68 of file CoinError.hpp.

9.16.3.3 `const std::string& CoinError::className () const [inline]`

get name of class instantiating error (or hint for assert)

Definition at line 71 of file CoinError.hpp.

9.16.3.4 `const std::string& CoinError::fileName () const [inline]`

get name of file for assert

Definition at line 74 of file CoinError.hpp.

9.16.3.5 `int CoinError::lineNumber () const [inline]`

get line number of assert (-1 if not assert)

Definition at line 77 of file CoinError.hpp.

9.16.3.6 `void CoinError::print (bool doPrint = true) const [inline]`

Just print (for asserts)

Definition at line 80 of file CoinError.hpp.

9.16.3.7 CoinError& CoinError::operator= (const CoinError & rhs) [inline]

Assignment operator.

Definition at line 128 of file CoinError.hpp.

9.16.4 Friends And Related Function Documentation

9.16.4.1 void CoinErrorUnitTest () [friend]

A function that tests the methods in the [CoinError](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

9.16.5 Member Data Documentation

9.16.5.1 bool CoinError::printErrors_ [static]

Whether to print every error.

Definition at line 165 of file CoinError.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinError.hpp](#)

9.17 CoinExternalVectorFirstGreater_2< S, T, V > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- [bool operator\(\)](#) (const [CoinPair](#)< S, T > &t1, const [CoinPair](#)< S, T > &t2) const
- [CoinExternalVectorFirstGreater_2](#) (const V *v)

9.17.1 Detailed Description

```
template<class S, class T, class V>class CoinExternalVectorFirstGreater_2< S, T, V >
```

Function operator.

Compare based on the entries of an external vector, i.e., returns true if `vec[t1.first] > vec[t2.first]`. Note that to use this comparison operator `.first` must be a data type automatically convertible to int.

Definition at line 120 of file CoinSort.hpp.

9.17.2 Constructor & Destructor Documentation

9.17.2.1 `template<class S , class T , class V > CoinExternalVectorFirstGreater_2< S, T, V
>::CoinExternalVectorFirstGreater_2 (const V * v) [inline]`

Definition at line 129 of file CoinSort.hpp.

9.17.3 Member Function Documentation

9.17.3.1 `template<class S , class T , class V > bool CoinExternalVectorFirstGreater_2< S, T, V >::operator() (const
CoinPair< S, T > & t1, const CoinPair< S, T > & t2) const [inline]`

Definition at line 126 of file CoinSort.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp>

9.18 CoinExternalVectorFirstGreater_3< S, T, U, V > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- `bool operator() (const CoinTriple< S, T, U > &t1, const CoinTriple< S, T, U > &t2) const`
- `CoinExternalVectorFirstGreater_3 (const V *v)`

9.18.1 Detailed Description

```
template<class S, class T, class U, class V>class CoinExternalVectorFirstGreater_3< S, T, U, V >
```

Function operator.

Compare based on the entries of an external vector, i.e., returns true if `vec[t1.first > vect2.first`. Note that to use this comparison operator `.first` must be a data type automatically convertible to int.

Definition at line 551 of file CoinSort.hpp.

9.18.2 Constructor & Destructor Documentation

9.18.2.1 `template<class S , class T , class U , class V > CoinExternalVectorFirstGreater_3< S, T, U, V
>::CoinExternalVectorFirstGreater_3 (const V * v) [inline]`

Definition at line 560 of file CoinSort.hpp.

9.18.3 Member Function Documentation

9.18.3.1 `template<class S , class T , class U , class V > bool CoinExternalVectorFirstGreater_3< S, T, U, V >::operator() (const
CoinTriple< S, T, U > & t1, const CoinTriple< S, T, U > & t2) const [inline]`

Definition at line 557 of file CoinSort.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp](#)

9.19 CoinExternalVectorFirstLess_2< S, T, V > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- `bool operator()` (const [CoinPair](#)< S, T > &t1, const [CoinPair](#)< S, T > &t2) const
- [CoinExternalVectorFirstLess_2](#) (const V *v)

9.19.1 Detailed Description

```
template<class S, class T, class V>class CoinExternalVectorFirstLess_2< S, T, V >
```

Function operator.

Compare based on the entries of an external vector, i.e., returns true if `vec[t1].first < vect2.first`. Note that to use this comparison operator `.first` must be a data type automatically convertible to int.

Definition at line 102 of file [CoinSort.hpp](#).

9.19.2 Constructor & Destructor Documentation

```
9.19.2.1 template<class S , class T , class V > CoinExternalVectorFirstLess_2< S, T, V
>::CoinExternalVectorFirstLess_2 ( const V * v ) [inline]
```

Definition at line 111 of file [CoinSort.hpp](#).

9.19.3 Member Function Documentation

```
9.19.3.1 template<class S , class T , class V > bool CoinExternalVectorFirstLess_2< S, T, V >::operator() ( const
CoinPair< S, T > & t1, const CoinPair< S, T > & t2 ) const [inline]
```

Definition at line 108 of file [CoinSort.hpp](#).

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp](#)

9.20 CoinExternalVectorFirstLess_3< S, T, U, V > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- `bool operator()` (const [CoinTriple](#)< S, T, U > &t1, const [CoinTriple](#)< S, T, U > &t2) const
- [CoinExternalVectorFirstLess_3](#) (const V *v)

9.20.1 Detailed Description

```
template<class S, class T, class U, class V>class CoinExternalVectorFirstLess_3< S, T, U, V >
```

Function operator.

Compare based on the entries of an external vector, i.e., returns true if `vec[t1].first < vec[t2].first`. Note that to use this comparison operator `.first` must be a data type automatically convertible to `int`.

Definition at line 533 of file `CoinSort.hpp`.

9.20.2 Constructor & Destructor Documentation

```
9.20.2.1 template<class S , class T , class U , class V > CoinExternalVectorFirstLess_3< S, T, U, V
>::CoinExternalVectorFirstLess_3( const V * v ) [inline]
```

Definition at line 542 of file `CoinSort.hpp`.

9.20.3 Member Function Documentation

```
9.20.3.1 template<class S , class T , class U , class V > bool CoinExternalVectorFirstLess_3< S, T, U, V >::operator() (
const CoinTriple< S, T, U > & t1, const CoinTriple< S, T, U > & t2 ) const [inline]
```

Definition at line 539 of file `CoinSort.hpp`.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp](#)

9.21 CoinFactorization Class Reference

This deals with Factorization and Updates.

```
#include <CoinFactorization.hpp>
```

Public Member Functions

Constructors and destructor and copy

- [CoinFactorization](#) ()
Default constructor.
- [CoinFactorization](#) (const [CoinFactorization](#) &other)
Copy constructor.
- [~CoinFactorization](#) ()
Destructor.
- [void almostDestructor](#) ()
Delete all stuff (leaves as after [CoinFactorization\(\)](#))
- [void show_self](#) () const
Debug show object (shows one representation)
- [int saveFactorization](#) (const char *file) const
Debug - save on file - 0 if no error.
- [int restoreFactorization](#) (const char *file, bool factor=false)
Debug - restore from file - 0 if no error on file.

- `void sort ()` const
Debug - sort so can compare.
- `CoinFactorization & operator=` (const `CoinFactorization` &other)
= copy

Do factorization

- `int factorize` (const `CoinPackedMatrix` &matrix, int rowsBasic[], int columnsBasic[], double `areaFactor`=0.0)
When part of LP - given by basic variables.
- `int factorize` (int `numberRows`, int `numberColumns`, `CoinBigIndex` `numberElements`, `CoinBigIndex` `maximumL`, `CoinBigIndex` `maximumU`, const int `indicesRow`[], const int `indicesColumn`[], const double `elements`[], int `permutation`[], double `areaFactor`=0.0)
When given as triplets.
- `int factorizePart1` (int `numberRows`, int `numberColumns`, `CoinBigIndex` `estimateNumberElements`, int *`indicesRow`[], int *`indicesColumn`[], `CoinFactorizationDouble` *`elements`[], double `areaFactor`=0.0)
Two part version for maximum flexibility This part creates arrays for user to fill.
- `int factorizePart2` (int `permutation`[], int `exactNumberElements`)
This is part two of factorization Arrays belong to factorization and were returned by part 1 If status okay, permutation has pivot rows - this is only needed If status is singular, then basic variables have pivot row and ones thrown out have -1 returns 0 -okay, -1 singular, -99 memory.
- `double conditionNumber ()` const
Condition number - product of pivots after factorization.

general stuff such as permutation or status

- `int status ()` const
Returns status.
- `void setStatus` (int value)
Sets status.
- `int pivots ()` const
Returns number of pivots since factorization.
- `void setPivots` (int value)
Sets number of pivots since factorization.
- `int * permute ()` const
Returns address of permute region.
- `int * pivotColumn ()` const
Returns address of pivotColumn region (also used for permuting)
- `CoinFactorizationDouble * pivotRegion ()` const
Returns address of pivot region.
- `int * permuteBack ()` const
Returns address of permuteBack region.
- `int * pivotColumnBack ()` const
Returns address of pivotColumnBack region (also used for permuting) Now uses firstCount to save memory allocation.
- `CoinBigIndex * startRowL ()` const
Start of each row in L.
- `CoinBigIndex * startColumnL ()` const
Start of each column in L.
- `int * indexColumnL ()` const
Index of column in row for L.
- `int * indexRowL ()` const
Row indices of L.
- `CoinFactorizationDouble * elementByRowL ()` const
Elements in L (row copy)
- `int numberOfRowsExtra ()` const
Number of Rows after iterating.

- `void setNumberRows` (int value)
Set number of Rows after factorization.
- `int numberRows` () const
Number of Rows after factorization.
- `CoinBigIndex numberL` () const
Number in L.
- `CoinBigIndex baseL` () const
Base of L.
- `int maximumRowsExtra` () const
Maximum of Rows after iterating.
- `int numberColumns` () const
Total number of columns in factorization.
- `int numberElements` () const
Total number of elements in factorization.
- `int numberForrestTomlin` () const
Length of FT vector.
- `int numberGoodColumns` () const
Number of good columns in factorization.
- `double areaFactor` () const
Whether larger areas needed.
- `void areaFactor` (double value)
- `double adjustedAreaFactor` () const
Returns areaFactor but adjusted for dense.
- `void relaxAccuracyCheck` (double value)
Allows change of pivot accuracy check 1.0 == none > 1.0 relaxed.
- `double getAccuracyCheck` () const
- `int messageLevel` () const
Level of detail of messages.
- `void messageLevel` (int value)
- `int maximumPivots` () const
Maximum number of pivots between factorizations.
- `void maximumPivots` (int value)
- `int denseThreshold` () const
Gets dense threshold.
- `void setDenseThreshold` (int value)
Sets dense threshold.
- `double pivotTolerance` () const
Pivot tolerance.
- `void pivotTolerance` (double value)
- `double zeroTolerance` () const
Zero tolerance.
- `void zeroTolerance` (double value)
- `double slackValue` () const
Whether slack value is +1 or -1.
- `void slackValue` (double value)
- `double maximumCoefficient` () const
Returns maximum absolute value in factorization.
- `bool forrestTomlin` () const
true if Forrest Tomlin update, false if PFI
- `void setForrestTomlin` (bool value)
- `bool spaceForForrestTomlin` () const
True if FT update and space.

some simple stuff

- `int numberDense` () const

- Returns number of dense rows.*
- `CoinBigIndex numberElementsU () const`
Returns number in U area.
- `void setNumberElementsU (CoinBigIndex value)`
Setss number in U area.
- `CoinBigIndex lengthAreaU () const`
Returns length of U area.
- `CoinBigIndex numberElementsL () const`
Returns number in L area.
- `CoinBigIndex lengthAreaL () const`
Returns length of L area.
- `CoinBigIndex numberElementsR () const`
Returns number in R area.
- `CoinBigIndex numberCompressions () const`
Number of compressions done.
- `int * numberInRow () const`
Number of entries in each row.
- `int * numberInColumn () const`
Number of entries in each column.
- `CoinFactorizationDouble * elementU () const`
Elements of U.
- `int * indexRowU () const`
Row indices of U.
- `CoinBigIndex * startColumnU () const`
Start of each column in U.
- `int maximumColumnsExtra ()`
Maximum number of Columns after iterating.
- `int biasLU () const`
L to U bias 0 - U bias, 1 - some U bias, 2 some L bias, 3 L bias.
- `void setBiasLU (int value)`
- `int persistenceFlag () const`
Array persistence flag If 0 then as now (delete/new) 1 then only do arrays if bigger needed 2 as 1 but give a bit extra if bigger needed.
- `void setPersistenceFlag (int value)`

rank one updates which do exist

- `int replaceColumn (CoinIndexedVector *regionSparse, int pivotRow, double pivotCheck, bool checkBeforeModifying=false, double acceptablePivot=1.0e-8)`
Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If checkBeforeModifying is true will do all accuracy checks before modifying factorization.
- `void replaceColumnU (CoinIndexedVector *regionSparse, CoinBigIndex *deleted, int internalPivotRow)`
Combines BtranU and delete elements If deleted is NULL then delete elements otherwise store where elements are.

various uses of factorization (return code number elements)

*** Below this user may not want to know about

which user may not want to know about (left over from my LP code)

- `int updateColumnFT (CoinIndexedVector *regionSparse, CoinIndexedVector *regionSparse2)`
Updates one column (FTRAN) from regionSparse2 Tries to do FT update number returned is negative if no room regionSparse starts as zero and is zero at end.
- `int updateColumn (CoinIndexedVector *regionSparse, CoinIndexedVector *regionSparse2, bool noPermute=false) const`
This version has same effect as above with FTUpdate==false so number returned is always >=0.

- `int updateTwoColumnsFT (CoinIndexedVector *regionSparse1, CoinIndexedVector *regionSparse2, CoinIndexedVector *regionSparse3, bool noPermuteRegion3=false)`
Updates one column (FTRAN) from region2 Tries to do FT update number returned is negative if no room.
- `int updateColumnTranspose (CoinIndexedVector *regionSparse, CoinIndexedVector *regionSparse2) const`
Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if region-Sparse2 packed on input - will be packed on output.
- `void goSparse ()`
makes a row copy of L for speed and to allow very sparse problems
- `int sparseThreshold () const`
get sparse threshold
- `void sparseThreshold (int value)`
set sparse threshold
- `void clearArrays ()`
Get rid of all memory.

various updates - none of which have been written!

- `int add (CoinBigIndex numberElements, int indicesRow[], int indicesColumn[], double elements[])`
Adds given elements to Basis and updates factorization, can increase size of basis.
- `int addColumn (CoinBigIndex numberElements, int indicesRow[], double elements[])`
Adds one Column to basis, can increase size of basis.
- `int addRow (CoinBigIndex numberElements, int indicesColumn[], double elements[])`
Adds one Row to basis, can increase size of basis.
- `int deleteColumn (int Row)`
Deletes one Column from basis, returns rank.
- `int deleteRow (int Row)`
Deletes one Row from basis, returns rank.
- `int replaceRow (int whichRow, int numberElements, const int indicesColumn[], const double elements[])`
Replaces one Row in basis, At present assumes just a singleton on row is in basis returns 0=OK, 1=Probably OK, 2=singular, 3 no space.
- `void emptyRows (int numberToEmpty, const int which[])`
Takes out all entries for given rows.

used by ClpFactorization

- `void checkSparse ()`
See if worth going sparse.
- `bool collectStatistics () const`
For statistics.
- `void setCollectStatistics (bool onOff) const`
For statistics.
- `void gutsOfDestructor (int type=1)`
The real work of constructors etc 0 just scalars, 1 bit normal.
- `void gutsOfInitialize (int type)`
1 bit - tolerances etc, 2 more, 4 dummy arrays
- `void gutsOfCopy (const CoinFactorization &other)`
- `void resetStatistics ()`
Reset all sparsity etc statistics.

Protected Attributes

data

- `double pivotTolerance_`

- *Pivot tolerance.*
- double [zeroTolerance_](#)
Zero tolerance.
- double [slackValue_](#)
Whether slack value is +1 or -1.
- double [areaFactor_](#)
How much to multiply areas by.
- double [relaxCheck_](#)
Relax check on accuracy in replaceColumn.
- int [numberRows_](#)
Number of Rows in factorization.
- int [numberRowsExtra_](#)
Number of Rows after iterating.
- int [maximumRowsExtra_](#)
Maximum number of Rows after iterating.
- int [numberColumns_](#)
Number of Columns in factorization.
- int [numberColumnsExtra_](#)
Number of Columns after iterating.
- int [maximumColumnsExtra_](#)
Maximum number of Columns after iterating.
- int [numberGoodU_](#)
Number factorized in U (not row singletons)
- int [numberGoodL_](#)
Number factorized in L.
- int [maximumPivots_](#)
Maximum number of pivots before factorization.
- int [numberPivots_](#)
Number pivots since last factorization.
- [CoinBigIndex totalElements_](#)
Number of elements in U (to go) or while iterating total overall.
- [CoinBigIndex factorElements_](#)
Number of elements after factorization.
- [CoinIntArrayWithLength pivotColumn_](#)
Pivot order for each Column.
- [CoinIntArrayWithLength permute_](#)
Permutation vector for pivot row order.
- [CoinIntArrayWithLength permuteBack_](#)
DePermutation vector for pivot row order.
- [CoinIntArrayWithLength pivotColumnBack_](#)
Inverse Pivot order for each Column.
- int [status_](#)
Status of factorization.
- int [numberTrials_](#)
0 - no increasing rows - no permutations, 1 - no increasing rows but permutations 2 - increasing rows
- [CoinBigIndexArrayWithLength startRowU_](#)
Start of each Row as pointer.
- [CoinIntArrayWithLength numberInRow_](#)
Number in each Row.
- [CoinIntArrayWithLength numberInColumn_](#)
Number in each Column.
- [CoinIntArrayWithLength numberInColumnPlus_](#)
Number in each Column including pivoted.
- [CoinIntArrayWithLength firstCount_](#)
First Row/Column with count of k, can tell which by offset - Rows then Columns.

- [CoinIntArrayWithLength nextCount_](#)
Next Row/Column with count.
- [CoinIntArrayWithLength lastCount_](#)
Previous Row/Column with count.
- [CoinIntArrayWithLength nextColumn_](#)
Next Column in memory order.
- [CoinIntArrayWithLength lastColumn_](#)
Previous Column in memory order.
- [CoinIntArrayWithLength nextRow_](#)
Next Row in memory order.
- [CoinIntArrayWithLength lastRow_](#)
Previous Row in memory order.
- [CoinIntArrayWithLength saveColumn_](#)
Columns left to do in a single pivot.
- [CoinIntArrayWithLength markRow_](#)
Marks rows to be updated.
- [int messageLevel_](#)
Detail in messages.
- [int biggerDimension_](#)
Larger of row and column size.
- [CoinIntArrayWithLength indexColumnU_](#)
Base address for U (may change)
- [CoinIntArrayWithLength pivotRowL_](#)
Pivots for L.
- [CoinFactorizationDoubleArrayWithLength pivotRegion_](#)
Inverses of pivot values.
- [int numberSlacks_](#)
Number of slacks at beginning of U.
- [int numberU_](#)
Number in U.
- [CoinBigIndex maximumU_](#)
Maximum space used in U.
- [CoinBigIndex lengthU_](#)
Base of U is always 0.
- [CoinBigIndex lengthAreaU_](#)
Length of area reserved for U.
- [CoinFactorizationDoubleArrayWithLength elementU_](#)
Elements of U.
- [CoinIntArrayWithLength indexRowU_](#)
Row indices of U.
- [CoinBigIndexArrayWithLength startColumnU_](#)
Start of each column in U.
- [CoinBigIndexArrayWithLength convertRowToColumnU_](#)
Converts rows to columns in U.
- [CoinBigIndex numberL_](#)
Number in L.
- [CoinBigIndex baseL_](#)
Base of L.
- [CoinBigIndex lengthL_](#)
Length of L.
- [CoinBigIndex lengthAreaL_](#)
Length of area reserved for L.
- [CoinFactorizationDoubleArrayWithLength elementL_](#)
Elements of L.
- [CoinIntArrayWithLength indexRowL_](#)

- Row indices of L.*
- [CoinBigIndexArrayWithLength startColumnL_](#)
 - Start of each column in L.*
- bool [doForrestTomlin_](#)
 - true if Forrest Tomlin update, false if PFI*
- int [numberR_](#)
 - Number in R.*
- [CoinBigIndex lengthR_](#)
 - Length of R stuff.*
- [CoinBigIndex lengthAreaR_](#)
 - length of area reserved for R*
- [CoinFactorizationDouble * elementR_](#)
 - Elements of R.*
- int * [indexRowR_](#)
 - Row indices for R.*
- [CoinBigIndexArrayWithLength startColumnR_](#)
 - Start of columns for R.*
- double * [denseArea_](#)
 - Dense area.*
- int * [densePermute_](#)
 - Dense permutation.*
- int [numberDense_](#)
 - Number of dense rows.*
- int [denseThreshold_](#)
 - Dense threshold.*
- [CoinFactorizationDoubleArrayWithLength workArea_](#)
 - First work area.*
- [CoinUnsignedIntArrayWithLength workArea2_](#)
 - Second work area.*
- [CoinBigIndex numberCompressions_](#)
 - Number of compressions done.*
- double [ftranCountInput_](#)
 - Below are all to collect.*
- double [ftranCountAfterL_](#)
- double [ftranCountAfterR_](#)
- double [ftranCountAfterU_](#)
- double [btranCountInput_](#)
- double [btranCountAfterU_](#)
- double [btranCountAfterR_](#)
- double [btranCountAfterL_](#)
- int [numberFtranCounts_](#)
 - We can roll over factorizations.*
- int [numberBtranCounts_](#)
- double [ftranAverageAfterL_](#)
 - While these are average ratios collected over last period.*
- double [ftranAverageAfterR_](#)
- double [ftranAverageAfterU_](#)
- double [btranAverageAfterU_](#)
- double [btranAverageAfterR_](#)
- double [btranAverageAfterL_](#)
- bool [collectStatistics_](#)
 - For statistics.*
- int [sparseThreshold_](#)
 - Below this use sparse technology - if 0 then no L row copy.*
- int [sparseThreshold2_](#)
 - And one for "sparsish".*

- [CoinBigIndexArrayWithLength startRowL_](#)
Start of each row in L.
- [CoinIntArrayWithLength indexColumnL_](#)
Index of column in row for L.
- [CoinFactorizationDoubleArrayWithLength elementByRowL_](#)
Elements in L (row copy)
- [CoinIntArrayWithLength sparse_](#)
Sparse regions.
- int [biasLU_](#)
L to U bias 0 - U bias, 1 - some U bias, 2 some L bias, 3 L bias.
- int [persistenceFlag_](#)
Array persistence flag If 0 then as now (delete/new) 1 then only do arrays if bigger needed 2 as 1 but give a bit extra if bigger needed.

Friends

- [void CoinFactorizationUnitTest](#) (const std::string &mpsDir)

used by factorization

- [void getAreas](#) (int [numberRows](#), int [numberColumns](#), [CoinBigIndex](#) maximumL, [CoinBigIndex](#) maximumU)
Gets space for a factorization, called by constructors.
- [void preProcess](#) (int state, int possibleDuplicates=-1)
PreProcesses raw triplet data.
- int [factor](#) ()
Does most of factorization.
- int [replaceColumnPFI](#) ([CoinIndexedVector](#) *regionSparse, int pivotRow, double alpha)
Replaces one Column to basis for PFI returns 0=OK, 1=Probably OK, 2=singular, 3=no room.
- int [factorSparse](#) ()
Does sparse phase of factorization return code is <0 error, 0= finished.
- int [factorSparseSmall](#) ()
Does sparse phase of factorization (for smaller problems) return code is <0 error, 0= finished.
- int [factorSparseLarge](#) ()
Does sparse phase of factorization (for larger problems) return code is <0 error, 0= finished.
- int [factorDense](#) ()
Does dense phase of factorization return code is <0 error, 0= finished.
- bool [pivotOneOtherRow](#) (int pivotRow, int [pivotColumn](#))
Pivots when just one other row so faster?
- bool [pivotRowSingleton](#) (int pivotRow, int [pivotColumn](#))
Does one pivot on Row Singleton in factorization.
- bool [pivotColumnSingleton](#) (int pivotRow, int [pivotColumn](#))
Does one pivot on Column Singleton in factorization.
- bool [getColumnSpace](#) (int iColumn, int extraNeeded)
Gets space for one Column with given length, may have to do compression (returns True if successful), also moves existing vector, extraNeeded is over and above present.
- bool [reorderU](#) ()
Reorders U so contiguous and in order (if there is space) Returns true if it could.
- bool [getColumnSpacelaterateR](#) (int iColumn, double value, int iRow)
getColumnSpacelaterateR.

- [CoinBigIndex getColumnSpaceltrate](#) (int iColumn, double value, int iRow)
getColumnSpaceltrate.
- [bool getRowSpace](#) (int iRow, int extraNeeded)
*Gets space for one Row with given length,
may have to do compression (returns True if successful), also moves existing vector*
- [bool getRowSpaceltrate](#) (int iRow, int extraNeeded)
*Gets space for one Row with given length while iterating,
may have to do compression (returns True if successful), also moves existing vector*
- [void checkConsistency](#) ()
Checks that row and column copies look OK.
- [void addLink](#) (int index, int count)
Adds a link in chain of equal counts.
- [void deleteLink](#) (int index)
Deletes a link in chain of equal counts.
- [void separateLinks](#) (int count, bool rowsFirst)
Separate out links with same row/column count.
- [void cleanup](#) ()
Cleans up at end of factorization.
- [void updateColumnL](#) ([CoinIndexedVector](#) *region, int *indexIn) const
Updates part of column (FTRANL)
- [void updateColumnLDensish](#) ([CoinIndexedVector](#) *region, int *indexIn) const
Updates part of column (FTRANL) when densish.
- [void updateColumnLSparse](#) ([CoinIndexedVector](#) *region, int *indexIn) const
Updates part of column (FTRANL) when sparse.
- [void updateColumnLSparsish](#) ([CoinIndexedVector](#) *region, int *indexIn) const
Updates part of column (FTRANL) when sparsish.
- [void updateColumnR](#) ([CoinIndexedVector](#) *region) const
Updates part of column (FTRANR) without FT update.
- [void updateColumnRFT](#) ([CoinIndexedVector](#) *region, int *indexIn)
Updates part of column (FTRANR) with FT update.
- [void updateColumnU](#) ([CoinIndexedVector](#) *region, int *indexIn) const
Updates part of column (FTRANU)
- [void updateColumnUSparse](#) ([CoinIndexedVector](#) *regionSparse, int *indexIn) const
Updates part of column (FTRANU) when sparse.
- [void updateColumnUSparsish](#) ([CoinIndexedVector](#) *regionSparse, int *indexIn) const
Updates part of column (FTRANU) when sparsish.
- [int updateColumnUDensish](#) (double *[COIN_RESTRICT](#) region, int *[COIN_RESTRICT](#) regionIndex) const
Updates part of column (FTRANU)
- [void updateTwoColumnsUDensish](#) (int &numberNonZero1, double *[COIN_RESTRICT](#) region1, int *[COIN_RESTRICT](#) index1, int &numberNonZero2, double *[COIN_RESTRICT](#) region2, int *[COIN_RESTRICT](#) index2) const
Updates part of 2 columns (FTRANU) real work.
- [void updateColumnPFI](#) ([CoinIndexedVector](#) *regionSparse) const
Updates part of column PFI (FTRAN) (after rest)
- [void permuteBack](#) ([CoinIndexedVector](#) *regionSparse, [CoinIndexedVector](#) *outVector) const
Permutes back at end of updateColumn.
- [void updateColumnTransposePFI](#) ([CoinIndexedVector](#) *region) const
Updates part of column transpose PFI (BTRAN) (before rest)
- [void updateColumnTransposeU](#) ([CoinIndexedVector](#) *region, int smallestIndex) const

- Updates part of column transpose (BTRANU), assumes index is sorted i.e.*
- `void updateColumnTransposeUSparsish (CoinIndexedVector *region, int smallestIndex) const`
Updates part of column transpose (BTRANU) when sparsish, assumes index is sorted i.e.
- `void updateColumnTransposeUDensish (CoinIndexedVector *region, int smallestIndex) const`
Updates part of column transpose (BTRANU) when densish, assumes index is sorted i.e.
- `void updateColumnTransposeUSparse (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANU) when sparse, assumes index is sorted i.e.
- `void updateColumnTransposeUByColumn (CoinIndexedVector *region, int smallestIndex) const`
Updates part of column transpose (BTRANU) by column assumes index is sorted i.e.
- `void updateColumnTransposeR (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANR)
- `void updateColumnTransposeRDensish (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANR) when dense.
- `void updateColumnTransposeRSparse (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANR) when sparse.
- `void updateColumnTransposeL (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANL)
- `void updateColumnTransposeLDensish (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANL) when densish by column.
- `void updateColumnTransposeLByRow (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANL) when densish by row.
- `void updateColumnTransposeLSparsish (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANL) when sparsish by row.
- `void updateColumnTransposeLSparse (CoinIndexedVector *region) const`
Updates part of column transpose (BTRANL) when sparse (by Row)
- `int checkPivot (double saveFromU, double oldPivot) const`
Returns accuracy status of replaceColumn returns 0=OK, 1=Probably OK, 2=singular.
- `template<class T >`
`bool pivot (int pivotRow, int pivotColumn, CoinBigIndex pivotRowPosition, CoinBigIndex pivotColumnPosition, CoinFactorizationDouble work[], unsigned int workArea2[], int increment2, T markRow[], int largeInteger)`

9.21.1 Detailed Description

This deals with Factorization and Updates.

This class started with a parallel simplex code I was writing in the mid 90's. The need for parallelism led to many complications and I have simplified as much as I could to get back to this.

I was aiming at problems where I might get speed-up so I was looking at dense problems or ones with structure. This led to permuting input and output vectors and to increasing the number of rows each rank-one update. This is still in as a minor overhead.

I have also put in handling for hyper-sparsity. I have taken out all outer loop unrolling, dense matrix handling and most of the book-keeping for slacks. Also I always use FTRAN approach to updating even if factorization fairly dense. All these could improve performance.

I blame some of the coding peculiarities on the history of the code but mostly it is just because I can't do elegant code (or useful comments).

I am assuming that 32 bits is enough for number of rows or columns, but CoinBigIndex may be redefined to get 64 bits.

Definition at line 50 of file CoinFactorization.hpp.

9.21.2 Constructor & Destructor Documentation

9.21.2.1 `CoinFactorization::CoinFactorization ()`

Default constructor.

9.21.2.2 `CoinFactorization::CoinFactorization (const CoinFactorization & other)`

Copy constructor.

9.21.2.3 `CoinFactorization::~~CoinFactorization ()`

Destructor.

9.21.3 Member Function Documentation

9.21.3.1 `void CoinFactorization::almostDestructor ()`

Delete all stuff (leaves as after [CoinFactorization\(\)](#))

9.21.3.2 `void CoinFactorization::show_self () const`

Debug show object (shows one representation)

9.21.3.3 `int CoinFactorization::saveFactorization (const char * file) const`

Debug - save on file - 0 if no error.

9.21.3.4 `int CoinFactorization::restoreFactorization (const char * file, bool factor = false)`

Debug - restore from file - 0 if no error on file.

If factor true then factorizes as if called from `ClpFactorization`

9.21.3.5 `void CoinFactorization::sort () const`

Debug - sort so can compare.

9.21.3.6 `CoinFactorization& CoinFactorization::operator= (const CoinFactorization & other)`

= copy

9.21.3.7 `int CoinFactorization::factorize (const CoinPackedMatrix & matrix, int rowsBasic[], int columnsBasic[], double areaFactor = 0.0)`

When part of LP - given by basic variables.

Actually does factorization. Arrays passed in have non negative value to say basic. If status is okay, basic variables have pivot row - this is only needed If status is singular, then basic variables have pivot row and ones thrown out have -1 returns 0 -okay, -1 singular, -2 too many in basis, -99 memory

9.21.3.8 `int CoinFactorization::factorize (int numberOfRows, int numberColumns, CoinBigIndex numberElements, CoinBigIndex maximumL, CoinBigIndex maximumU, const int indicesRow[], const int indicesColumn[], const double elements[], int permutation[], double areaFactor = 0.0)`

When given as triplets.

Actually does factorization. maximumL is guessed maximum size of L part of final factorization, maximumU of U part.

These are multiplied by areaFactor which can be computed by user or internally. Arrays are copied in. I could add flag to delete arrays to save a bit of memory. If status okay, permutation has pivot rows - this is only needed If status is singular, then basic variables have pivot row and ones thrown out have -1 returns 0 -okay, -1 singular, -99 memory

9.21.3.9 `int CoinFactorization::factorizePart1 (int numberRows, int numberColumns, CoinBigIndex estimateNumberElements, int * indicesRow[], int * indicesColumn[], CoinFactorizationDouble * elements[], double areaFactor = 0.0)`

Two part version for maximum flexibility This part creates arrays for user to fill.

estimateNumberElements is safe estimate of number returns 0 -okay, -99 memory

9.21.3.10 `int CoinFactorization::factorizePart2 (int permutation[], int exactNumberElements)`

This is part two of factorization Arrays belong to factorization and were returned by part 1 If status okay, permutation has pivot rows - this is only needed If status is singular, then basic variables have pivot row and ones thrown out have -1 returns 0 -okay, -1 singular, -99 memory.

9.21.3.11 `double CoinFactorization::conditionNumber () const`

Condition number - product of pivots after factorization.

9.21.3.12 `int CoinFactorization::status () const [inline]`

Returns status.

Definition at line 137 of file CoinFactorization.hpp.

9.21.3.13 `void CoinFactorization::setStatus (int value) [inline]`

Sets status.

Definition at line 141 of file CoinFactorization.hpp.

9.21.3.14 `int CoinFactorization::pivots () const [inline]`

Returns number of pivots since factorization.

Definition at line 144 of file CoinFactorization.hpp.

9.21.3.15 `void CoinFactorization::setPivots (int value) [inline]`

Sets number of pivots since factorization.

Definition at line 148 of file CoinFactorization.hpp.

9.21.3.16 `int* CoinFactorization::permute () const [inline]`

Returns address of permute region.

Definition at line 151 of file CoinFactorization.hpp.

9.21.3.17 `int* CoinFactorization::pivotColumn () const [inline]`

Returns address of pivotColumn region (also used for permuting)

Definition at line 155 of file CoinFactorization.hpp.

9.21.3.18 `CoinFactorizationDouble* CoinFactorization::pivotRegion () const [inline]`

Returns address of pivot region.

Definition at line 159 of file CoinFactorization.hpp.

9.21.3.19 `int* CoinFactorization::permuteBack () const [inline]`

Returns address of permuteBack region.

Definition at line 163 of file CoinFactorization.hpp.

9.21.3.20 `int* CoinFactorization::pivotColumnBack () const [inline]`

Returns address of pivotColumnBack region (also used for permuting) Now uses firstCount to save memory allocation.

Definition at line 168 of file CoinFactorization.hpp.

9.21.3.21 `CoinBigIndex* CoinFactorization::startRowL () const [inline]`

Start of each row in L.

Definition at line 173 of file CoinFactorization.hpp.

9.21.3.22 `CoinBigIndex* CoinFactorization::startColumnL () const [inline]`

Start of each column in L.

Definition at line 177 of file CoinFactorization.hpp.

9.21.3.23 `int* CoinFactorization::indexColumnL () const [inline]`

Index of column in row for L.

Definition at line 181 of file CoinFactorization.hpp.

9.21.3.24 `int* CoinFactorization::indexRowL () const [inline]`

Row indices of L.

Definition at line 185 of file CoinFactorization.hpp.

9.21.3.25 `CoinFactorizationDouble* CoinFactorization::elementByRowL () const [inline]`

Elements in L (row copy)

Definition at line 189 of file CoinFactorization.hpp.

9.21.3.26 `int CoinFactorization::numberOfRowsExtra () const [inline]`

Number of Rows after iterating.

Definition at line 193 of file CoinFactorization.hpp.

9.21.3.27 `void CoinFactorization::setNumberOfRows (int value) [inline]`

Set number of Rows after factorization.

Definition at line 197 of file CoinFactorization.hpp.

9.21.3.28 `int CoinFactorization::numberOfRows () const [inline]`

Number of Rows after factorization.

Definition at line 200 of file CoinFactorization.hpp.

9.21.3.29 CoinBigIndex CoinFactorization::numberL () const [inline]

Number in L.

Definition at line 204 of file CoinFactorization.hpp.

9.21.3.30 CoinBigIndex CoinFactorization::baseL () const [inline]

Base of L.

Definition at line 208 of file CoinFactorization.hpp.

9.21.3.31 int CoinFactorization::maximumRowsExtra () const [inline]

Maximum of Rows after iterating.

Definition at line 211 of file CoinFactorization.hpp.

9.21.3.32 int CoinFactorization::numberColumns () const [inline]

Total number of columns in factorization.

Definition at line 215 of file CoinFactorization.hpp.

9.21.3.33 int CoinFactorization::numberElements () const [inline]

Total number of elements in factorization.

Definition at line 219 of file CoinFactorization.hpp.

9.21.3.34 int CoinFactorization::numberForrestTomlin () const [inline]

Length of FT vector.

Definition at line 223 of file CoinFactorization.hpp.

9.21.3.35 int CoinFactorization::numberGoodColumns () const [inline]

Number of good columns in factorization.

Definition at line 227 of file CoinFactorization.hpp.

9.21.3.36 double CoinFactorization::areaFactor () const [inline]

Whether larger areas needed.

Definition at line 231 of file CoinFactorization.hpp.

9.21.3.37 void CoinFactorization::areaFactor (double *value*) [inline]

Definition at line 234 of file CoinFactorization.hpp.

9.21.3.38 double CoinFactorization::adjustedAreaFactor () const

Returns areaFactor but adjusted for dense.

9.21.3.39 void CoinFactorization::relaxAccuracyCheck (double *value*) [inline]

Allows change of pivot accuracy check 1.0 == none >1.0 relaxed.

Definition at line 240 of file CoinFactorization.hpp.

9.21.3.40 **double** CoinFactorization::getAccuracyCheck () const [inline]

Definition at line 242 of file CoinFactorization.hpp.

9.21.3.41 **int** CoinFactorization::messageLevel () const [inline]

Level of detail of messages.

Definition at line 245 of file CoinFactorization.hpp.

9.21.3.42 **void** CoinFactorization::messageLevel (int *value*)

9.21.3.43 **int** CoinFactorization::maximumPivots () const [inline]

Maximum number of pivots between factorizations.

Definition at line 250 of file CoinFactorization.hpp.

9.21.3.44 **void** CoinFactorization::maximumPivots (int *value*)

9.21.3.45 **int** CoinFactorization::denseThreshold () const [inline]

Gets dense threshold.

Definition at line 256 of file CoinFactorization.hpp.

9.21.3.46 **void** CoinFactorization::setDenseThreshold (int *value*) [inline]

Sets dense threshold.

Definition at line 259 of file CoinFactorization.hpp.

9.21.3.47 **double** CoinFactorization::pivotTolerance () const [inline]

Pivot tolerance.

Definition at line 262 of file CoinFactorization.hpp.

9.21.3.48 **void** CoinFactorization::pivotTolerance (double *value*)

9.21.3.49 **double** CoinFactorization::zeroTolerance () const [inline]

Zero tolerance.

Definition at line 267 of file CoinFactorization.hpp.

9.21.3.50 **void** CoinFactorization::zeroTolerance (double *value*)

9.21.3.51 **double** CoinFactorization::slackValue () const [inline]

Whether slack value is +1 or -1.

Definition at line 273 of file CoinFactorization.hpp.

9.21.3.52 **void** CoinFactorization::slackValue (double *value*)

9.21.3.53 **double** CoinFactorization::maximumCoefficient () const

Returns maximum absolute value in factorization.

9.21.3.54 `bool CoinFactorization::forrestTomlin () const [inline]`

true if Forrest Tomlin update, false if PFI

Definition at line 281 of file CoinFactorization.hpp.

9.21.3.55 `void CoinFactorization::setForrestTomlin (bool value) [inline]`

Definition at line 283 of file CoinFactorization.hpp.

9.21.3.56 `bool CoinFactorization::spaceForForrestTomlin () const [inline]`

True if FT update and space.

Definition at line 286 of file CoinFactorization.hpp.

9.21.3.57 `int CoinFactorization::numberDense () const [inline]`

Returns number of dense rows.

Definition at line 298 of file CoinFactorization.hpp.

9.21.3.58 `CoinBigIndex CoinFactorization::numberElementsU () const [inline]`

Returns number in U area.

Definition at line 302 of file CoinFactorization.hpp.

9.21.3.59 `void CoinFactorization::setNumberElementsU (CoinBigIndex value) [inline]`

Setss number in U area.

Definition at line 306 of file CoinFactorization.hpp.

9.21.3.60 `CoinBigIndex CoinFactorization::lengthAreaU () const [inline]`

Returns length of U area.

Definition at line 309 of file CoinFactorization.hpp.

9.21.3.61 `CoinBigIndex CoinFactorization::numberElementsL () const [inline]`

Returns number in L area.

Definition at line 313 of file CoinFactorization.hpp.

9.21.3.62 `CoinBigIndex CoinFactorization::lengthAreaL () const [inline]`

Returns length of L area.

Definition at line 317 of file CoinFactorization.hpp.

9.21.3.63 `CoinBigIndex CoinFactorization::numberElementsR () const [inline]`

Returns number in R area.

Definition at line 321 of file CoinFactorization.hpp.

9.21.3.64 `CoinBigIndex CoinFactorization::numberCompressions () const [inline]`

Number of compressions done.

Definition at line 325 of file CoinFactorization.hpp.

9.21.3.65 `int* CoinFactorization::numberInRow () const [inline]`

Number of entries in each row.

Definition at line 328 of file CoinFactorization.hpp.

9.21.3.66 `int* CoinFactorization::numberInColumn () const [inline]`

Number of entries in each column.

Definition at line 331 of file CoinFactorization.hpp.

9.21.3.67 `CoinFactorizationDouble* CoinFactorization::elementU () const [inline]`

Elements of U.

Definition at line 334 of file CoinFactorization.hpp.

9.21.3.68 `int* CoinFactorization::indexRowU () const [inline]`

Row indices of U.

Definition at line 337 of file CoinFactorization.hpp.

9.21.3.69 `CoinBigIndex* CoinFactorization::startColumnU () const [inline]`

Start of each column in U.

Definition at line 340 of file CoinFactorization.hpp.

9.21.3.70 `int CoinFactorization::maximumColumnsExtra () [inline]`

Maximum number of Columns after iterating.

Definition at line 343 of file CoinFactorization.hpp.

9.21.3.71 `int CoinFactorization::biasLU () const [inline]`

L to U bias 0 - U bias, 1 - some U bias, 2 some L bias, 3 L bias.

Definition at line 348 of file CoinFactorization.hpp.

9.21.3.72 `void CoinFactorization::setBiasLU (int value) [inline]`

Definition at line 350 of file CoinFactorization.hpp.

9.21.3.73 `int CoinFactorization::persistenceFlag () const [inline]`

Array persistence flag If 0 then as now (delete/new) 1 then only do arrays if bigger needed 2 as 1 but give a bit extra if bigger needed.

Definition at line 357 of file CoinFactorization.hpp.

9.21.3.74 `void CoinFactorization::setPersistenceFlag (int value)`

9.21.3.75 `int CoinFactorization::replaceColumn (CoinIndexedVector * regionSparse, int pivotRow, double pivotCheck, bool checkBeforeModifying = false, double acceptablePivot = 1.0e-8)`

Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If checkBeforeModifying is true will do all accuracy checks before modifying factorization.

Whether to set this depends on speed considerations. You could just do this on first iteration after factorization and thereafter re-factorize partial update already in U

9.21.3.76 `void CoinFactorization::replaceColumnU (CoinIndexedVector * regionSparse, CoinBigIndex * deleted, int internalPivotRow)`

Combines BtranU and delete elements If deleted is NULL then delete elements otherwise store where elements are.

9.21.3.77 `int CoinFactorization::updateColumnFT (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2)`

Updates one column (FTRAN) from regionSparse2 Tries to do FT update number returned is negative if no room region-Sparse starts as zero and is zero at end.

Note - if regionSparse2 packed on input - will be packed on output

9.21.3.78 `int CoinFactorization::updateColumn (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool noPermute = false) const`

This version has same effect as above with FTUpdate==false so number returned is always ≥ 0 .

9.21.3.79 `int CoinFactorization::updateTwoColumnsFT (CoinIndexedVector * regionSparse1, CoinIndexedVector * regionSparse2, CoinIndexedVector * regionSparse3, bool noPermuteRegion3 = false)`

Updates one column (FTRAN) from region2 Tries to do FT update number returned is negative if no room.

Also updates region3 region1 starts as zero and is zero at end

9.21.3.80 `int CoinFactorization::updateColumnTranspose (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2) const`

Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if region-Sparse2 packed on input - will be packed on output.

9.21.3.81 `void CoinFactorization::goSparse ()`

makes a row copy of L for speed and to allow very sparse problems

9.21.3.82 `int CoinFactorization::sparseThreshold () const [inline]`

get sparse threshold

Definition at line 420 of file CoinFactorization.hpp.

9.21.3.83 `void CoinFactorization::sparseThreshold (int value)`

set sparse threshold

9.21.3.84 `void CoinFactorization::clearArrays () [inline]`

Get rid of all memory.

Definition at line 431 of file CoinFactorization.hpp.

9.21.3.85 `int CoinFactorization::add (CoinBigIndex numberElements, int indicesRow[], int indicesColumn[], double elements[])`

Adds given elements to Basis and updates factorization, can increase size of basis.

Returns rank

9.21.3.86 `int CoinFactorization::addColumn (CoinBigIndex numberElements, int indicesRow[], double elements[])`

Adds one Column to basis, can increase size of basis.

Returns rank

9.21.3.87 `int CoinFactorization::addRow (CoinBigIndex numberElements, int indicesColumn[], double elements[])`

Adds one Row to basis, can increase size of basis.

Returns rank

9.21.3.88 `int CoinFactorization::deleteColumn (int Row)`

Deletes one Column from basis, returns rank.

9.21.3.89 `int CoinFactorization::deleteRow (int Row)`

Deletes one Row from basis, returns rank.

9.21.3.90 `int CoinFactorization::replaceRow (int whichRow, int numberElements, const int indicesColumn[], const double elements[])`

Replaces one Row in basis, At present assumes just a singleton on row is in basis returns 0=OK, 1=Probably OK, 2=singular, 3 no space.

9.21.3.91 `void CoinFactorization::emptyRows (int numberToEmpty, const int which[])`

Takes out all entries for given rows.

9.21.3.92 `void CoinFactorization::checkSparse ()`

See if worth going sparse.

9.21.3.93 `bool CoinFactorization::collectStatistics () const [inline]`

For statistics.

Definition at line 471 of file CoinFactorization.hpp.

9.21.3.94 `void CoinFactorization::setCollectStatistics (bool onOff) const [inline]`

For statistics.

Definition at line 474 of file CoinFactorization.hpp.

9.21.3.95 `void CoinFactorization::gutsOfDestructor (int type = 1)`

The real work of constructors etc 0 just scalars, 1 bit normal.

9.21.3.96 `void CoinFactorization::gutsOfInitialize (int type)`

1 bit - tolerances etc, 2 more, 4 dummy arrays

9.21.3.97 void CoinFactorization::gutsOfCopy (const CoinFactorization & *other*)

9.21.3.98 void CoinFactorization::resetStatistics ()

Reset all sparsity etc statistics.

9.21.3.99 void CoinFactorization::getAreas (int *numberOfRows*, int *numberOfColumns*, CoinBigIndex *maximumL*, CoinBigIndex *maximumU*)

Gets space for a factorization, called by constructors.

9.21.3.100 void CoinFactorization::preProcess (int *state*, int *possibleDuplicates* = -1)

PreProcesses raw triplet data.

state is 0 - triplets, 1 - some counts etc , 2 - ..

9.21.3.101 int CoinFactorization::factor ()

Does most of factorization.

9.21.3.102 int CoinFactorization::factorSparse () [protected]

Does sparse phase of factorization return code is <0 error, 0= finished.

9.21.3.103 int CoinFactorization::factorSparseSmall () [protected]

Does sparse phase of factorization (for smaller problems) return code is <0 error, 0= finished.

9.21.3.104 int CoinFactorization::factorSparseLarge () [protected]

Does sparse phase of factorization (for larger problems) return code is <0 error, 0= finished.

9.21.3.105 int CoinFactorization::factorDense () [protected]

Does dense phase of factorization return code is <0 error, 0= finished.

9.21.3.106 bool CoinFactorization::pivotOneOtherRow (int *pivotRow*, int *pivotColumn*) [protected]

Pivots when just one other row so faster?

9.21.3.107 bool CoinFactorization::pivotRowSingleton (int *pivotRow*, int *pivotColumn*) [protected]

Does one pivot on Row Singleton in factorization.

9.21.3.108 bool CoinFactorization::pivotColumnSingleton (int *pivotRow*, int *pivotColumn*) [protected]

Does one pivot on Column Singleton in factorization.

9.21.3.109 bool CoinFactorization::getColumnSpace (int *iColumn*, int *extraNeeded*) [protected]

Gets space for one Column with given length, may have to do compression (returns True if successful), also moves existing vector, extraNeeded is over and above present.

9.21.3.110 bool CoinFactorization::reorderU () [protected]

Reorders U so contiguous and in order (if there is space) Returns true if it could.

9.21.3.111 **bool** CoinFactorization::getColumnSpacelaterR (int *iColumn*, double *value*, int *iRow*) [protected]

getColumnSpacelaterR.

Gets space for one extra R element in Column may have to do compression (returns true) also moves existing vector

9.21.3.112 **CoinBigIndex** CoinFactorization::getColumnSpacelater (int *iColumn*, double *value*, int *iRow*) [protected]

getColumnSpacelater.

Gets space for one extra U element in Column may have to do compression (returns true) also moves existing vector. Returns -1 if no memory or where element was put Used by replaceRow (turns off R version)

9.21.3.113 **bool** CoinFactorization::getRowSpace (int *iRow*, int *extraNeeded*) [protected]

Gets space for one Row with given length,

may have to do compression (returns True if successful), also moves existing vector

9.21.3.114 **bool** CoinFactorization::getRowSpacelater (int *iRow*, int *extraNeeded*) [protected]

Gets space for one Row with given length while iterating,

may have to do compression (returns True if successful), also moves existing vector

9.21.3.115 **void** CoinFactorization::checkConsistency () [protected]

Checks that row and column copies look OK.

9.21.3.116 **void** CoinFactorization::addLink (int *index*, int *count*) [inline],[protected]

Adds a link in chain of equal counts.

Definition at line 560 of file CoinFactorization.hpp.

9.21.3.117 **void** CoinFactorization::deleteLink (int *index*) [inline],[protected]

Deletes a link in chain of equal counts.

Definition at line 576 of file CoinFactorization.hpp.

9.21.3.118 **void** CoinFactorization::separateLinks (int *count*, bool *rowsFirst*) [protected]

Separate out links with same row/column count.

9.21.3.119 **void** CoinFactorization::cleanup () [protected]

Cleans up at end of factorization.

9.21.3.120 **void** CoinFactorization::updateColumnL (CoinIndexedVector * *region*, int * *indexIn*) const [protected]

Updates part of column (FTRANL)

9.21.3.121 **void** CoinFactorization::updateColumnLDensish (CoinIndexedVector * *region*, int * *indexIn*) const
[protected]

Updates part of column (FTRANL) when densish.

9.21.3.122 **void** CoinFactorization::updateColumnLSparse (CoinIndexedVector * *region*, int * *indexIn*) const
[protected]

Updates part of column (FTRANL) when sparse.

9.21.3.123 **void** CoinFactorization::updateColumnLSparsish (CoinIndexedVector * *region*, int * *indexIn*) const
[protected]

Updates part of column (FTRANL) when sparsish.

9.21.3.124 **void** CoinFactorization::updateColumnR (CoinIndexedVector * *region*) const [protected]

Updates part of column (FTRANR) without FT update.

9.21.3.125 **void** CoinFactorization::updateColumnRFT (CoinIndexedVector * *region*, int * *indexIn*) [protected]

Updates part of column (FTRANR) with FT update.

Also stores update after L and R

9.21.3.126 **void** CoinFactorization::updateColumnU (CoinIndexedVector * *region*, int * *indexIn*) const [protected]

Updates part of column (FTRANU)

9.21.3.127 **void** CoinFactorization::updateColumnUSparse (CoinIndexedVector * *regionSparse*, int * *indexIn*) const
[protected]

Updates part of column (FTRANU) when sparse.

9.21.3.128 **void** CoinFactorization::updateColumnUSparsish (CoinIndexedVector * *regionSparse*, int * *indexIn*) const
[protected]

Updates part of column (FTRANU) when sparsish.

9.21.3.129 **int** CoinFactorization::updateColumnUDensish (double *COIN_RESTRICT *region*, int *COIN_RESTRICT
regionIndex) const [protected]

Updates part of column (FTRANU)

9.21.3.130 **void** CoinFactorization::updateTwoColumnsUDensish (int & *numberNonZero1*, double *COIN_RESTRICT
region1, int *COIN_RESTRICT *index1*, int & *numberNonZero2*, double *COIN_RESTRICT *region2*, int
*COIN_RESTRICT *index2*) const [protected]

Updates part of 2 columns (FTRANU) real work.

9.21.3.131 **void** CoinFactorization::updateColumnPFI (CoinIndexedVector * *regionSparse*) const [protected]

Updates part of column PFI (FTRAN) (after rest)

9.21.3.132 **void** CoinFactorization::permuteBack (CoinIndexedVector * *regionSparse*, CoinIndexedVector * *outVector*)
const [protected]

Permutes back at end of updateColumn.

9.21.3.133 **void** CoinFactorization::updateColumnTransposePFI (CoinIndexedVector * *region*) const [protected]

Updates part of column transpose PFI (BTRAN) (before rest)

9.21.3.134 void CoinFactorization::updateColumnTransposeU (CoinIndexedVector * *region*, int *smallestIndex*) const
[protected]

Updates part of column transpose (BTRANU), assumes index is sorted i.e.
region is correct

9.21.3.135 void CoinFactorization::updateColumnTransposeUSparsish (CoinIndexedVector * *region*, int *smallestIndex*) const
[protected]

Updates part of column transpose (BTRANU) when sparsish, assumes index is sorted i.e.
region is correct

9.21.3.136 void CoinFactorization::updateColumnTransposeUDensish (CoinIndexedVector * *region*, int *smallestIndex*) const
[protected]

Updates part of column transpose (BTRANU) when densish, assumes index is sorted i.e.
region is correct

9.21.3.137 void CoinFactorization::updateColumnTransposeUSparse (CoinIndexedVector * *region*) const [protected]

Updates part of column transpose (BTRANU) when sparse, assumes index is sorted i.e.
region is correct

9.21.3.138 void CoinFactorization::updateColumnTransposeUByColumn (CoinIndexedVector * *region*, int *smallestIndex*) const [protected]

Updates part of column transpose (BTRANU) by column assumes index is sorted i.e.
region is correct

9.21.3.139 void CoinFactorization::updateColumnTransposeR (CoinIndexedVector * *region*) const [protected]

Updates part of column transpose (BTRANR)

9.21.3.140 void CoinFactorization::updateColumnTransposeRDensish (CoinIndexedVector * *region*) const
[protected]

Updates part of column transpose (BTRANR) when dense.

9.21.3.141 void CoinFactorization::updateColumnTransposeRSparse (CoinIndexedVector * *region*) const [protected]

Updates part of column transpose (BTRANR) when sparse.

9.21.3.142 void CoinFactorization::updateColumnTransposeL (CoinIndexedVector * *region*) const [protected]

Updates part of column transpose (BTRANL)

9.21.3.143 void CoinFactorization::updateColumnTransposeLDensish (CoinIndexedVector * *region*) const
[protected]

Updates part of column transpose (BTRANL) when densish by column.

9.21.3.144 void CoinFactorization::updateColumnTransposeLByRow (CoinIndexedVector * *region*) const [protected]

Updates part of column transpose (BTRANL) when densish by row.

9.21.3.145 **void** CoinFactorization::updateColumnTransposeLSparsish (**CoinIndexedVector** * *region*) **const** [protected]

Updates part of column transpose (BTRANL) when sparsish by row.

9.21.3.146 **void** CoinFactorization::updateColumnTransposeLSparse (**CoinIndexedVector** * *region*) **const** [protected]

Updates part of column transpose (BTRANL) when sparse (by Row)

9.21.3.147 **int** CoinFactorization::replaceColumnPFI (**CoinIndexedVector** * *regionSparse*, **int** *pivotRow*, **double** *alpha*)

Replaces one Column to basis for PFI returns 0=OK, 1=Probably OK, 2=singular, 3=no room.

In this case region is not empty - it is incoming variable (updated)

9.21.3.148 **int** CoinFactorization::checkPivot (**double** *saveFromU*, **double** *oldPivot*) **const** [protected]

Returns accuracy status of replaceColumn returns 0=OK, 1=Probably OK, 2=singular.

9.21.3.149 **template**<class **T** > **bool** CoinFactorization::pivot (**int** *pivotRow*, **int** *pivotColumn*, **CoinBigIndex** *pivotRowPosition*, **CoinBigIndex** *pivotColumnPosition*, **CoinFactorizationDouble** *work*[], **unsigned int** *workArea2*[], **int** *increment2*, **T** *markRow*[], **int** *largelInteger*) [inline], [protected]

Definition at line 703 of file CoinFactorization.hpp.

9.21.4 Friends And Related Function Documentation

9.21.4.1 **void** CoinFactorizationUnitTest (**const** **std::string** & *mpsDir*) [friend]

9.21.5 Member Data Documentation

9.21.5.1 **double** CoinFactorization::pivotTolerance_ [protected]

Pivot tolerance.

Definition at line 1187 of file CoinFactorization.hpp.

9.21.5.2 **double** CoinFactorization::zeroTolerance_ [protected]

Zero tolerance.

Definition at line 1189 of file CoinFactorization.hpp.

9.21.5.3 **double** CoinFactorization::slackValue_ [protected]

Whether slack value is +1 or -1.

Definition at line 1192 of file CoinFactorization.hpp.

9.21.5.4 **double** CoinFactorization::areaFactor_ [protected]

How much to multiply areas by.

Definition at line 1199 of file CoinFactorization.hpp.

9.21.5.5 **double** CoinFactorization::relaxCheck_ [protected]

Relax check on accuracy in replaceColumn.

Definition at line 1201 of file CoinFactorization.hpp.

9.21.5.6 `int CoinFactorization::numberRows_` `[protected]`

Number of Rows in factorization.

Definition at line 1203 of file CoinFactorization.hpp.

9.21.5.7 `int CoinFactorization::numberRowsExtra_` `[protected]`

Number of Rows after iterating.

Definition at line 1205 of file CoinFactorization.hpp.

9.21.5.8 `int CoinFactorization::maximumRowsExtra_` `[protected]`

Maximum number of Rows after iterating.

Definition at line 1207 of file CoinFactorization.hpp.

9.21.5.9 `int CoinFactorization::numberColumns_` `[protected]`

Number of Columns in factorization.

Definition at line 1209 of file CoinFactorization.hpp.

9.21.5.10 `int CoinFactorization::numberColumnsExtra_` `[protected]`

Number of Columns after iterating.

Definition at line 1211 of file CoinFactorization.hpp.

9.21.5.11 `int CoinFactorization::maximumColumnsExtra_` `[protected]`

Maximum number of Columns after iterating.

Definition at line 1213 of file CoinFactorization.hpp.

9.21.5.12 `int CoinFactorization::numberGoodU_` `[protected]`

Number factorized in U (not row singletons)

Definition at line 1215 of file CoinFactorization.hpp.

9.21.5.13 `int CoinFactorization::numberGoodL_` `[protected]`

Number factorized in L.

Definition at line 1217 of file CoinFactorization.hpp.

9.21.5.14 `int CoinFactorization::maximumPivots_` `[protected]`

Maximum number of pivots before factorization.

Definition at line 1219 of file CoinFactorization.hpp.

9.21.5.15 `int CoinFactorization::numberPivots_` `[protected]`

Number pivots since last factorization.

Definition at line 1221 of file CoinFactorization.hpp.

9.21.5.16 CoinBigIndex CoinFactorization::totalElements_ [protected]

Number of elements in U (to go) or while iterating total overall.

Definition at line 1224 of file CoinFactorization.hpp.

9.21.5.17 CoinBigIndex CoinFactorization::factorElements_ [protected]

Number of elements after factorization.

Definition at line 1226 of file CoinFactorization.hpp.

9.21.5.18 CoinIntArrayWithLength CoinFactorization::pivotColumn_ [protected]

Pivot order for each Column.

Definition at line 1228 of file CoinFactorization.hpp.

9.21.5.19 CoinIntArrayWithLength CoinFactorization::permute_ [protected]

Permutation vector for pivot row order.

Definition at line 1230 of file CoinFactorization.hpp.

9.21.5.20 CoinIntArrayWithLength CoinFactorization::permuteBack_ [protected]

DePermutation vector for pivot row order.

Definition at line 1232 of file CoinFactorization.hpp.

9.21.5.21 CoinIntArrayWithLength CoinFactorization::pivotColumnBack_ [protected]

Inverse Pivot order for each Column.

Definition at line 1234 of file CoinFactorization.hpp.

9.21.5.22 int CoinFactorization::status_ [protected]

Status of factorization.

Definition at line 1236 of file CoinFactorization.hpp.

9.21.5.23 int CoinFactorization::numberTrials_ [protected]

0 - no increasing rows - no permutations, 1 - no increasing rows but permutations 2 - increasing rows

- taken out as always 2 Number of trials before rejection

Definition at line 1245 of file CoinFactorization.hpp.

9.21.5.24 CoinBigIndexArrayWithLength CoinFactorization::startRowU_ [protected]

Start of each Row as pointer.

Definition at line 1247 of file CoinFactorization.hpp.

9.21.5.25 CoinIntArrayWithLength CoinFactorization::numberInRow_ [protected]

Number in each Row.

Definition at line 1250 of file CoinFactorization.hpp.

9.21.5.26 CoinIntArrayWithLength CoinFactorization::numberInColumn_ [protected]

Number in each Column.

Definition at line 1253 of file CoinFactorization.hpp.

9.21.5.27 CoinIntArrayWithLength CoinFactorization::numberInColumnPlus_ [protected]

Number in each Column including pivoted.

Definition at line 1256 of file CoinFactorization.hpp.

9.21.5.28 CoinIntArrayWithLength CoinFactorization::firstCount_ [protected]

First Row/Column with count of k, can tell which by offset - Rows then Columns.

Definition at line 1260 of file CoinFactorization.hpp.

9.21.5.29 CoinIntArrayWithLength CoinFactorization::nextCount_ [protected]

Next Row/Column with count.

Definition at line 1263 of file CoinFactorization.hpp.

9.21.5.30 CoinIntArrayWithLength CoinFactorization::lastCount_ [protected]

Previous Row/Column with count.

Definition at line 1266 of file CoinFactorization.hpp.

9.21.5.31 CoinIntArrayWithLength CoinFactorization::nextColumn_ [protected]

Next Column in memory order.

Definition at line 1269 of file CoinFactorization.hpp.

9.21.5.32 CoinIntArrayWithLength CoinFactorization::lastColumn_ [protected]

Previous Column in memory order.

Definition at line 1272 of file CoinFactorization.hpp.

9.21.5.33 CoinIntArrayWithLength CoinFactorization::nextRow_ [protected]

Next Row in memory order.

Definition at line 1275 of file CoinFactorization.hpp.

9.21.5.34 CoinIntArrayWithLength CoinFactorization::lastRow_ [protected]

Previous Row in memory order.

Definition at line 1278 of file CoinFactorization.hpp.

9.21.5.35 CoinIntArrayWithLength CoinFactorization::saveColumn_ [protected]

Columns left to do in a single pivot.

Definition at line 1281 of file CoinFactorization.hpp.

9.21.5.36 CoinIntArrayWithLength CoinFactorization::markRow_ [protected]

Marks rows to be updated.

Definition at line 1284 of file CoinFactorization.hpp.

9.21.5.37 int CoinFactorization::messageLevel_ [protected]

Detail in messages.

Definition at line 1287 of file CoinFactorization.hpp.

9.21.5.38 int CoinFactorization::biggerDimension_ [protected]

Larger of row and column size.

Definition at line 1290 of file CoinFactorization.hpp.

9.21.5.39 CoinIntArrayWithLength CoinFactorization::indexColumnU_ [protected]

Base address for U (may change)

Definition at line 1293 of file CoinFactorization.hpp.

9.21.5.40 CoinIntArrayWithLength CoinFactorization::pivotRowL_ [protected]

Pivots for L.

Definition at line 1296 of file CoinFactorization.hpp.

9.21.5.41 CoinFactorizationDoubleArrayWithLength CoinFactorization::pivotRegion_ [protected]

Inverses of pivot values.

Definition at line 1299 of file CoinFactorization.hpp.

9.21.5.42 int CoinFactorization::numberSlacks_ [protected]

Number of slacks at beginning of U.

Definition at line 1302 of file CoinFactorization.hpp.

9.21.5.43 int CoinFactorization::numberU_ [protected]

Number in U.

Definition at line 1305 of file CoinFactorization.hpp.

9.21.5.44 CoinBigIndex CoinFactorization::maximumU_ [protected]

Maximum space used in U.

Definition at line 1308 of file CoinFactorization.hpp.

9.21.5.45 CoinBigIndex CoinFactorization::lengthU_ [protected]

Base of U is always 0.

Length of U

Definition at line 1314 of file CoinFactorization.hpp.

9.21.5.46 CoinBigIndex CoinFactorization::lengthAreaU_ [protected]

Length of area reserved for U.

Definition at line 1317 of file CoinFactorization.hpp.

9.21.5.47 CoinFactorizationDoubleArrayWithLength CoinFactorization::elementU_ [protected]

Elements of U.

Definition at line 1320 of file CoinFactorization.hpp.

9.21.5.48 CoinIntArrayWithLength CoinFactorization::indexRowU_ [protected]

Row indices of U.

Definition at line 1323 of file CoinFactorization.hpp.

9.21.5.49 CoinBigIndexArrayWithLength CoinFactorization::startColumnU_ [protected]

Start of each column in U.

Definition at line 1326 of file CoinFactorization.hpp.

9.21.5.50 CoinBigIndexArrayWithLength CoinFactorization::convertRowToColumnU_ [protected]

Converts rows to columns in U.

Definition at line 1329 of file CoinFactorization.hpp.

9.21.5.51 CoinBigIndex CoinFactorization::numberL_ [protected]

Number in L.

Definition at line 1332 of file CoinFactorization.hpp.

9.21.5.52 CoinBigIndex CoinFactorization::baseL_ [protected]

Base of L.

Definition at line 1335 of file CoinFactorization.hpp.

9.21.5.53 CoinBigIndex CoinFactorization::lengthL_ [protected]

Length of L.

Definition at line 1338 of file CoinFactorization.hpp.

9.21.5.54 CoinBigIndex CoinFactorization::lengthAreaL_ [protected]

Length of area reserved for L.

Definition at line 1341 of file CoinFactorization.hpp.

9.21.5.55 CoinFactorizationDoubleArrayWithLength CoinFactorization::elementL_ [protected]

Elements of L.

Definition at line 1344 of file CoinFactorization.hpp.

9.21.5.56 CoinIntArrayWithLength CoinFactorization::indexRowL_ [protected]

Row indices of L.

Definition at line 1347 of file CoinFactorization.hpp.

9.21.5.57 CoinBigIndexArrayWithLength CoinFactorization::startColumnL_ [protected]

Start of each column in L.

Definition at line 1350 of file CoinFactorization.hpp.

9.21.5.58 bool CoinFactorization::doForrestTomlin_ [protected]

true if Forrest Tomlin update, false if PFI

Definition at line 1353 of file CoinFactorization.hpp.

9.21.5.59 int CoinFactorization::numberR_ [protected]

Number in R.

Definition at line 1356 of file CoinFactorization.hpp.

9.21.5.60 CoinBigIndex CoinFactorization::lengthR_ [protected]

Length of R stuff.

Definition at line 1359 of file CoinFactorization.hpp.

9.21.5.61 CoinBigIndex CoinFactorization::lengthAreaR_ [protected]

length of area reserved for R

Definition at line 1362 of file CoinFactorization.hpp.

9.21.5.62 CoinFactorizationDouble* CoinFactorization::elementR_ [protected]

Elements of R.

Definition at line 1365 of file CoinFactorization.hpp.

9.21.5.63 int* CoinFactorization::indexRowR_ [protected]

Row indices for R.

Definition at line 1368 of file CoinFactorization.hpp.

9.21.5.64 CoinBigIndexArrayWithLength CoinFactorization::startColumnR_ [protected]

Start of columns for R.

Definition at line 1371 of file CoinFactorization.hpp.

9.21.5.65 double* CoinFactorization::denseArea_ [protected]

Dense area.

Definition at line 1374 of file CoinFactorization.hpp.

9.21.5.66 `int* CoinFactorization::densePermute_` `[protected]`

Dense permutation.

Definition at line 1377 of file `CoinFactorization.hpp`.

9.21.5.67 `int CoinFactorization::numberDense_` `[protected]`

Number of dense rows.

Definition at line 1380 of file `CoinFactorization.hpp`.

9.21.5.68 `int CoinFactorization::denseThreshold_` `[protected]`

Dense threshold.

Definition at line 1383 of file `CoinFactorization.hpp`.

9.21.5.69 `CoinFactorizationDoubleArrayWithLength CoinFactorization::workArea_` `[protected]`

First work area.

Definition at line 1386 of file `CoinFactorization.hpp`.

9.21.5.70 `CoinUnsignedIntArrayWithLength CoinFactorization::workArea2_` `[protected]`

Second work area.

Definition at line 1389 of file `CoinFactorization.hpp`.

9.21.5.71 `CoinBigIndex CoinFactorization::numberCompressions_` `[protected]`

Number of compressions done.

Definition at line 1392 of file `CoinFactorization.hpp`.

9.21.5.72 `double CoinFactorization::ftranCountInput_` `[mutable], [protected]`

Below are all to collect.

Definition at line 1395 of file `CoinFactorization.hpp`.

9.21.5.73 `double CoinFactorization::ftranCountAfterL_` `[mutable], [protected]`

Definition at line 1396 of file `CoinFactorization.hpp`.

9.21.5.74 `double CoinFactorization::ftranCountAfterR_` `[mutable], [protected]`

Definition at line 1397 of file `CoinFactorization.hpp`.

9.21.5.75 `double CoinFactorization::ftranCountAfterU_` `[mutable], [protected]`

Definition at line 1398 of file `CoinFactorization.hpp`.

9.21.5.76 `double CoinFactorization::btranCountInput_` `[mutable], [protected]`

Definition at line 1399 of file `CoinFactorization.hpp`.

9.21.5.77 `double CoinFactorization::btranCountAfterU_` `[mutable], [protected]`

Definition at line 1400 of file `CoinFactorization.hpp`.

9.21.5.78 `double CoinFactorization::btranCountAfterR_` [mutable], [protected]

Definition at line 1401 of file CoinFactorization.hpp.

9.21.5.79 `double CoinFactorization::btranCountAfterL_` [mutable], [protected]

Definition at line 1402 of file CoinFactorization.hpp.

9.21.5.80 `int CoinFactorization::numberFtranCounts_` [mutable], [protected]

We can roll over factorizations.

Definition at line 1405 of file CoinFactorization.hpp.

9.21.5.81 `int CoinFactorization::numberBtranCounts_` [mutable], [protected]

Definition at line 1406 of file CoinFactorization.hpp.

9.21.5.82 `double CoinFactorization::ftranAverageAfterL_` [protected]

While these are average ratios collected over last period.

Definition at line 1409 of file CoinFactorization.hpp.

9.21.5.83 `double CoinFactorization::ftranAverageAfterR_` [protected]

Definition at line 1410 of file CoinFactorization.hpp.

9.21.5.84 `double CoinFactorization::ftranAverageAfterU_` [protected]

Definition at line 1411 of file CoinFactorization.hpp.

9.21.5.85 `double CoinFactorization::btranAverageAfterU_` [protected]

Definition at line 1412 of file CoinFactorization.hpp.

9.21.5.86 `double CoinFactorization::btranAverageAfterR_` [protected]

Definition at line 1413 of file CoinFactorization.hpp.

9.21.5.87 `double CoinFactorization::btranAverageAfterL_` [protected]

Definition at line 1414 of file CoinFactorization.hpp.

9.21.5.88 `bool CoinFactorization::collectStatistics_` [mutable], [protected]

For statistics.

Definition at line 1417 of file CoinFactorization.hpp.

9.21.5.89 `int CoinFactorization::sparseThreshold_` [protected]

Below this use sparse technology - if 0 then no L row copy.

Definition at line 1420 of file CoinFactorization.hpp.

9.21.5.90 `int CoinFactorization::sparseThreshold2_` [protected]

And one for "sparsish".

Definition at line 1423 of file CoinFactorization.hpp.

9.21.5.91 **CoinBigIndexArrayWithLength** CoinFactorization::startRowL_ [protected]

Start of each row in L.

Definition at line 1426 of file CoinFactorization.hpp.

9.21.5.92 **CoinIntArrayWithLength** CoinFactorization::indexColumnL_ [protected]

Index of column in row for L.

Definition at line 1429 of file CoinFactorization.hpp.

9.21.5.93 **CoinFactorizationDoubleArrayWithLength** CoinFactorization::elementByRowL_ [protected]

Elements in L (row copy)

Definition at line 1432 of file CoinFactorization.hpp.

9.21.5.94 **CoinIntArrayWithLength** CoinFactorization::sparse_ [mutable], [protected]

Sparse regions.

Definition at line 1435 of file CoinFactorization.hpp.

9.21.5.95 **int** CoinFactorization::biasLU_ [protected]

L to U bias 0 - U bias, 1 - some U bias, 2 some L bias, 3 L bias.

Definition at line 1439 of file CoinFactorization.hpp.

9.21.5.96 **int** CoinFactorization::persistenceFlag_ [protected]

Array persistence flag If 0 then as now (delete/new) 1 then only do arrays if bigger needed 2 as 1 but give a bit extra if bigger needed.

Definition at line 1445 of file CoinFactorization.hpp.

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

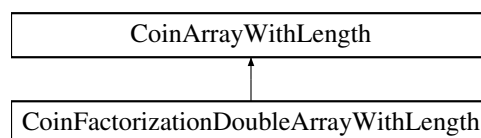
- </home/ted/COIN/trunk/CoinUtils/src/CoinFactorization.hpp>

9.22 **CoinFactorizationDoubleArrayWithLength** Class Reference

CoinFactorizationDouble * version.

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for CoinFactorizationDoubleArrayWithLength:



Public Member Functions

Get methods.

- int `getSize` () const
Get the size.
- `CoinFactorizationDouble` * `array` () const
Get Array.

Set methods

- void `setSize` (int value)
Set the size.

Condition methods

- `CoinFactorizationDouble` * `conditionalNew` (int sizeWanted)
Conditionally gets new array.

Constructors and destructors

- `CoinFactorizationDoubleArrayWithLength` ()
Default constructor - NULL.
- `CoinFactorizationDoubleArrayWithLength` (int size)
Alternate Constructor - length in bytes - size_ -1.
- `CoinFactorizationDoubleArrayWithLength` (int size, int mode)
Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.
- `CoinFactorizationDoubleArrayWithLength` (const `CoinFactorizationDoubleArrayWithLength` &rhs)
Copy constructor.
- `CoinFactorizationDoubleArrayWithLength` (const `CoinFactorizationDoubleArrayWithLength` *rhs)
Copy constructor.2.
- `CoinFactorizationDoubleArrayWithLength` & `operator=` (const `CoinFactorizationDoubleArrayWithLength` &rhs)
Assignment operator.

Additional Inherited Members

9.22.1 Detailed Description

`CoinFactorizationDouble` * version.

Definition at line 671 of file `CoinIndexedVector.hpp`.

9.22.2 Constructor & Destructor Documentation

9.22.2.1 `CoinFactorizationDoubleArrayWithLength::CoinFactorizationDoubleArrayWithLength ()` `[inline]`

Default constructor - NULL.

Definition at line 701 of file `CoinIndexedVector.hpp`.

9.22.2.2 `CoinFactorizationDoubleArrayWithLength::CoinFactorizationDoubleArrayWithLength (int size)` `[inline]`

Alternate Constructor - length in bytes - size_ -1.

Definition at line 704 of file `CoinIndexedVector.hpp`.

9.22.2.3 **CoinFactorizationDoubleArrayWithLength::CoinFactorizationDoubleArrayWithLength (int *size*, int *mode*)** `[inline]`

Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.

Definition at line 710 of file CoinIndexedVector.hpp.

9.22.2.4 **CoinFactorizationDoubleArrayWithLength::CoinFactorizationDoubleArrayWithLength (const CoinFactorizationDoubleArrayWithLength & *rhs*)** `[inline]`

Copy constructor.

Definition at line 713 of file CoinIndexedVector.hpp.

9.22.2.5 **CoinFactorizationDoubleArrayWithLength::CoinFactorizationDoubleArrayWithLength (const CoinFactorizationDoubleArrayWithLength * *rhs*)** `[inline]`

Copy constructor.2.

Definition at line 716 of file CoinIndexedVector.hpp.

9.22.3 Member Function Documentation

9.22.3.1 **int CoinFactorizationDoubleArrayWithLength::getSize () const** `[inline]`

Get the size.

Definition at line 677 of file CoinIndexedVector.hpp.

9.22.3.2 **CoinFactorizationDouble* CoinFactorizationDoubleArrayWithLength::array () const** `[inline]`

Get Array.

Definition at line 680 of file CoinIndexedVector.hpp.

9.22.3.3 **void CoinFactorizationDoubleArrayWithLength::setSize (int *value*)** `[inline]`

Set the size.

Definition at line 687 of file CoinIndexedVector.hpp.

9.22.3.4 **CoinFactorizationDouble* CoinFactorizationDoubleArrayWithLength::conditionalNew (int *sizeWanted*)** `[inline]`

Conditionally gets new array.

Definition at line 694 of file CoinIndexedVector.hpp.

9.22.3.5 **CoinFactorizationDoubleArrayWithLength& CoinFactorizationDoubleArrayWithLength::operator= (const CoinFactorizationDoubleArrayWithLength & *rhs*)** `[inline]`

Assignment operator.

Definition at line 719 of file CoinIndexedVector.hpp.

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

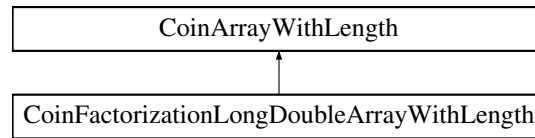
- /home/ted/COIN/trunk/CoinUtils/src/[CoinIndexedVector.hpp](#)

9.23 CoinFactorizationLongDoubleArrayWithLength Class Reference

CoinFactorizationLongDouble * version.

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for CoinFactorizationLongDoubleArrayWithLength:



Public Member Functions

Get methods.

- `int getSize () const`
Get the size.
- `long double * array () const`
Get Array.

Set methods

- `void setSize (int value)`
Set the size.

Condition methods

- `long double * conditionalNew (int sizeWanted)`
Conditionally gets new array.

Constructors and destructors

- `CoinFactorizationLongDoubleArrayWithLength ()`
Default constructor - NULL.
- `CoinFactorizationLongDoubleArrayWithLength (int size)`
Alternate Constructor - length in bytes - size_ - 1.
- `CoinFactorizationLongDoubleArrayWithLength (int size, int mode)`
Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.
- `CoinFactorizationLongDoubleArrayWithLength (const CoinFactorizationLongDoubleArrayWithLength &rhs)`
Copy constructor.
- `CoinFactorizationLongDoubleArrayWithLength (const CoinFactorizationLongDoubleArrayWithLength *rhs)`
Copy constructor.2.
- `CoinFactorizationLongDoubleArrayWithLength & operator= (const CoinFactorizationLongDoubleArrayWithLength &rhs)`
Assignment operator.

Additional Inherited Members

9.23.1 Detailed Description

CoinFactorizationLongDouble * version.

Definition at line 725 of file CoinIndexedVector.hpp.

9.23.2 Constructor & Destructor Documentation

9.23.2.1 `CoinFactorizationLongDoubleArrayWithLength::CoinFactorizationLongDoubleArrayWithLength () [inline]`

Default constructor - NULL.

Definition at line 755 of file `CoinIndexedVector.hpp`.

9.23.2.2 `CoinFactorizationLongDoubleArrayWithLength::CoinFactorizationLongDoubleArrayWithLength (int size) [inline]`

Alternate Constructor - length in bytes - size_ -1.

Definition at line 758 of file `CoinIndexedVector.hpp`.

9.23.2.3 `CoinFactorizationLongDoubleArrayWithLength::CoinFactorizationLongDoubleArrayWithLength (int size, int mode) [inline]`

Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.

Definition at line 764 of file `CoinIndexedVector.hpp`.

9.23.2.4 `CoinFactorizationLongDoubleArrayWithLength::CoinFactorizationLongDoubleArrayWithLength (const CoinFactorizationLongDoubleArrayWithLength & rhs) [inline]`

Copy constructor.

Definition at line 767 of file `CoinIndexedVector.hpp`.

9.23.2.5 `CoinFactorizationLongDoubleArrayWithLength::CoinFactorizationLongDoubleArrayWithLength (const CoinFactorizationLongDoubleArrayWithLength * rhs) [inline]`

Copy constructor.2.

Definition at line 770 of file `CoinIndexedVector.hpp`.

9.23.3 Member Function Documentation

9.23.3.1 `int CoinFactorizationLongDoubleArrayWithLength::getSize () const [inline]`

Get the size.

Definition at line 731 of file `CoinIndexedVector.hpp`.

9.23.3.2 `long double* CoinFactorizationLongDoubleArrayWithLength::array () const [inline]`

Get Array.

Definition at line 734 of file `CoinIndexedVector.hpp`.

9.23.3.3 `void CoinFactorizationLongDoubleArrayWithLength::setSize (int value) [inline]`

Set the size.

Definition at line 741 of file `CoinIndexedVector.hpp`.

9.23.3.4 `long double* CoinFactorizationLongDoubleArrayWithLength::conditionalNew (int sizeWanted) [inline]`

Conditionally gets new array.

Definition at line 748 of file `CoinIndexedVector.hpp`.

9.23.3.5 **CoinFactorizationLongDoubleArrayWithLength& CoinFactorizationLongDoubleArrayWithLength::operator= (const CoinFactorizationLongDoubleArrayWithLength & rhs) [inline]**

Assignment operator.

Definition at line 773 of file CoinIndexedVector.hpp.

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

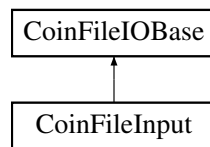
- /home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp

9.24 CoinFileInput Class Reference

Abstract base class for file input classes.

```
#include <CoinFileIO.hpp>
```

Inheritance diagram for CoinFileInput:



Public Member Functions

- [CoinFileInput](#) (const std::string &fileName)
Constructor (don't use this, use the create method instead).
- virtual [~CoinFileInput](#) ()
Destructor.
- virtual int [read](#) (void *buffer, int size)=0
Read a block of data from the file, similar to fread.
- virtual char * [gets](#) (char *buffer, int size)=0
Reads up to (size-1) characters and stores them into the buffer, similar to fgets.

Static Public Member Functions

- static bool [haveGzipSupport](#) ()
indicates whether [CoinFileInput](#) supports gzip'ed files
- static bool [haveBzip2Support](#) ()
indicates whether [CoinFileInput](#) supports bzip2'ed files
- static [CoinFileInput](#) * [create](#) (const std::string &fileName)
Factory method, that creates a [CoinFileInput](#) (more precisely a subclass of it) for the file specified.

Related Functions

(Note that these are not member functions.)

- bool [fileAbsPath](#) (const std::string &path)

Test if the given string looks like an absolute file path.

- `bool fileCoinReadable (std::string &name, const std::string &dfltPrefix=std::string(""))`

Test if the file is readable, using likely versions of the file name, and return the name that worked.

Additional Inherited Members

9.24.1 Detailed Description

Abstract base class for file input classes.

Definition at line 37 of file CoinFileIO.hpp.

9.24.2 Constructor & Destructor Documentation

9.24.2.1 CoinFileInput::CoinFileInput (const std::string & fileName)

Constructor (don't use this, use the create method instead).

Parameters

<i>fileName</i>	The name of the file used by this object.
-----------------	---

9.24.2.2 virtual CoinFileInput::~~CoinFileInput () [virtual]

Destructor.

9.24.3 Member Function Documentation

9.24.3.1 static bool CoinFileInput::haveGzipSupport () [static]

indicates whether [CoinFileInput](#) supports gzip'ed files

9.24.3.2 static bool CoinFileInput::haveBzip2Support () [static]

indicates whether [CoinFileInput](#) supports bzip2'ed files

9.24.3.3 static CoinFileInput* CoinFileInput::create (const std::string & fileName) [static]

Factory method, that creates a [CoinFileInput](#) (more precisely a subclass of it) for the file specified.

This method reads the first few bytes of the file and determines if this is a compressed or a plain file and returns the correct subclass to handle it. If the file does not exist or uses a compression not compiled in an exception is thrown.

Parameters

<i>fileName</i>	The file that should be read.
-----------------	-------------------------------

9.24.3.4 virtual int CoinFileInput::read (void * buffer, int size) [pure virtual]

Read a block of data from the file, similar to fread.

Parameters

<i>buffer</i>	Address of a buffer to store the data into.
<i>size</i>	Number of bytes to read (buffer should be large enough).

Returns

Number of bytes read.

9.24.3.5 virtual char* CoinFileInput::gets (char * *buffer*, int *size*) [pure virtual]

Reads up to (size-1) characters and stores them into the buffer, similar to fgets.

Reading ends, when EOF or a newline occurs or (size-1) characters have been read. The resulting string is terminated with '\0'. If reading ends due to an encountered newline, the '

' is put into the buffer, before the '\0' is appended.

Parameters

<i>buffer</i>	The buffer to put the string into.
<i>size</i>	The size of the buffer in characters.

Returns

buffer on success, or 0 if no characters have been read.

9.24.4 Friends And Related Function Documentation

9.24.4.1 bool fileAbsPath (const std::string & *path*) [related]

Test if the given string looks like an absolute file path.

The criteria are:

- unix: string begins with '/'
- windows: string begins with '\ ' or with 'drv:' (drive specifier)

9.24.4.2 bool fileCoinReadable (std::string & *name*, const std::string & *dfltPrefix* = std::string("")) [related]

Test if the file is readable, using likely versions of the file name, and return the name that worked.

The file name is constructed from *name* using the following rules:

- An absolute path is not modified.
- If the name begins with '~', an attempt is made to replace '~' with the value of the environment variable HOME.
- If a default prefix (*dfltPrefix*) is provided, it is prepended to the name.

If the constructed file name cannot be opened, and CoinUtils was built with support for compressed files, fileCoinReadable will try any standard extensions for supported compressed files.

The value returned in *name* is the file name that actually worked.

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

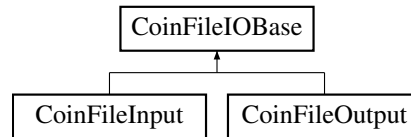
- /home/ted/COIN/trunk/CoinUtils/src/CoinFileIO.hpp

9.25 CoinFileIOBase Class Reference

Base class for FileIO classes.

```
#include <CoinFileIO.hpp>
```

Inheritance diagram for CoinFileIOBase:



Public Member Functions

- [CoinFileIOBase](#) (const std::string &fileName)
Constructor.
- [~CoinFileIOBase](#) ()
Destructor.
- const char * [getFileName](#) () const
Return the name of the file used by this object.
- std::string [getReadType](#) () const
Return the method of reading being used.

Protected Attributes

- std::string [readType_](#)

9.25.1 Detailed Description

Base class for FileIO classes.

Definition at line 11 of file CoinFileIO.hpp.

9.25.2 Constructor & Destructor Documentation

9.25.2.1 CoinFileIOBase::CoinFileIOBase (const std::string & fileName)

Constructor.

Parameters

<i>fileName</i>	The name of the file used by this object.
-----------------	---

9.25.2.2 CoinFileIOBase::~~CoinFileIOBase ()

Destructor.

9.25.3 Member Function Documentation

9.25.3.1 `const char* CoinFileIOBase::getFileName () const`

Return the name of the file used by this object.

9.25.3.2 `std::string CoinFileIOBase::getReadType () const` `[inline]`

Return the method of reading being used.

Definition at line 25 of file CoinFileIO.hpp.

9.25.4 Member Data Documentation

9.25.4.1 `std::string CoinFileIOBase::readType_` `[protected]`

Definition at line 28 of file CoinFileIO.hpp.

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

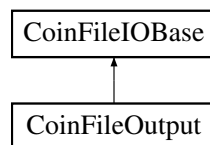
- `/home/ted/COIN/trunk/CoinUtils/src/CoinFileIO.hpp`

9.26 CoinFileOutput Class Reference

Abstract base class for file output classes.

```
#include <CoinFileIO.hpp>
```

Inheritance diagram for CoinFileOutput:



Public Types

- enum `Compression` { `COMPRESS_NONE` = 0, `COMPRESS_GZIP` = 1, `COMPRESS_BZIP2` = 2 }
The compression method.

Public Member Functions

- `CoinFileOutput` (const std::string &fileName)
Constructor (don't use this, use the create method instead).
- virtual `~CoinFileOutput` ()
Destructor.
- virtual int `write` (const void *buffer, int size)=0
Write a block of data to the file, similar to fwrite.
- virtual bool `puts` (const char *s)
Write a string to the file (like fputs).

- bool `puts` (const std::string &s)
Convenience method: just a 'puts(s.c_str())'.

Static Public Member Functions

- static bool `compressionSupported` (`Compression` compression)
Returns whether the specified compression method is supported (i.e.
- static `CoinFileOutput` * `create` (const std::string &fileName, `Compression` compression)
Factory method, that creates a `CoinFileOutput` (more precisely a subclass of it) for the file specified.

Additional Inherited Members

9.26.1 Detailed Description

Abstract base class for file output classes.

Definition at line 80 of file `CoinFileIO.hpp`.

9.26.2 Member Enumeration Documentation

9.26.2.1 enum `CoinFileOutput::Compression`

The compression method.

Enumerator

`COMPRESS_NONE` No compression.
`COMPRESS_GZIP` gzip compression.
`COMPRESS_BZIP2` bzip2 compression.

Definition at line 85 of file `CoinFileIO.hpp`.

9.26.3 Constructor & Destructor Documentation

9.26.3.1 `CoinFileOutput::CoinFileOutput (const std::string & fileName)`

Constructor (don't use this, use the create method instead).

Parameters

<i>fileName</i>	The name of the file used by this object.
-----------------	---

9.26.3.2 `virtual CoinFileOutput::~~CoinFileOutput () [virtual]`

Destructor.

9.26.4 Member Function Documentation

9.26.4.1 `static bool CoinFileOutput::compressionSupported (Compression compression) [static]`

Returns whether the specified compression method is supported (i.e.
 was compiled into COIN).

9.26.4.2 `static CoinFileOutput* CoinFileOutput::create (const std::string & fileName, Compression compression)`
`[static]`

Factory method, that creates a [CoinFileOutput](#) (more precisely a subclass of it) for the file specified.

If the compression method is not supported an exception is thrown (so use `compressionSupported` first, if this is a problem). The reason for not providing direct access to the subclasses (and using such a method instead) is that depending on the build configuration some of the classes are not available (or functional). This way we can handle all required `ifdefs` here instead of polluting other files.

Parameters

<i>fileName</i>	The file that should be read.
<i>compression</i>	Compression method used.

9.26.4.3 `virtual int CoinFileOutput::write (const void * buffer, int size)` `[pure virtual]`

Write a block of data to the file, similar to `fwrite`.

Parameters

<i>buffer</i>	Address of a buffer containing the data to be written.
<i>size</i>	Number of bytes to write.

Returns

Number of bytes written.

9.26.4.4 `virtual bool CoinFileOutput::puts (const char * s)` `[virtual]`

Write a string to the file (like `fputs`).

Just as with `fputs` no trailing newline is inserted! The terminating `'\0'` is not written to the file. The default implementation determines the length of the string and calls `write` on it.

Parameters

<i>s</i>	The zero terminated string to be written.
----------	---

Returns

true on success, false on error.

9.26.4.5 `bool CoinFileOutput::puts (const std::string & s)` `[inline]`

Convenience method: just a `'puts(s.c_str())'`.

Definition at line 131 of file `CoinFileIO.hpp`.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinFileIO.hpp](#)

9.27 CoinFirstAbsGreater_2< S, T > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- bool [operator\(\)](#) ([CoinPair](#)< S, T > t1, [CoinPair](#)< S, T > t2) const
Compare function.

9.27.1 Detailed Description

```
template<class S, class T>class CoinFirstAbsGreater_2< S, T >
```

Function operator.

Returns true if `abs(t1.first) > abs(t2.first)` (i.e., decreasing).

Definition at line 85 of file `CoinSort.hpp`.

9.27.2 Member Function Documentation

```
9.27.2.1 template<class S , class T > bool CoinFirstAbsGreater_2< S, T >::operator() ( CoinPair< S, T > t1, CoinPair< S, T > t2 ) const [inline]
```

Compare function.

Definition at line 88 of file `CoinSort.hpp`.

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

- `/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp`

9.28 CoinFirstAbsGreater_3< S, T, U > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (const [CoinTriple](#)< S, T, U > &t1, const [CoinTriple](#)< S, T, U > &t2) const
Compare function.

9.28.1 Detailed Description

```
template<class S, class T, class U>class CoinFirstAbsGreater_3< S, T, U >
```

Function operator.

Returns true if `abs(t1.first) > abs(t2.first)` (i.e., decreasing).

Definition at line 515 of file `CoinSort.hpp`.

9.28.2 Member Function Documentation

9.28.2.1 `template<class S , class T , class U > bool CoinFirstAbsGreater_3< S, T, U >::operator() (const CoinTriple< S, T, U > & t1, const CoinTriple< S, T, U > & t2) const [inline]`

Compare function.

Definition at line 518 of file CoinSort.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp>

9.29 CoinFirstAbsLess_2< S, T > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- `bool operator() (const CoinPair< S, T > &t1, const CoinPair< S, T > &t2) const`
Compare function.

9.29.1 Detailed Description

```
template<class S, class T>class CoinFirstAbsLess_2< S, T >
```

Function operator.

Returns true if `abs(t1.first) < abs(t2.first)` (i.e., increasing).

Definition at line 70 of file CoinSort.hpp.

9.29.2 Member Function Documentation

9.29.2.1 `template<class S , class T > bool CoinFirstAbsLess_2< S, T >::operator() (const CoinPair< S, T > & t1, const CoinPair< S, T > & t2) const [inline]`

Compare function.

Definition at line 73 of file CoinSort.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp>

9.30 CoinFirstAbsLess_3< S, T, U > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (const [CoinTriple](#)< S, T, U > &t1, const [CoinTriple](#)< S, T, U > &t2) const
Compare function.

9.30.1 Detailed Description

template<class S, class T, class U>class [CoinFirstAbsLess_3](#)< S, T, U >

Function operator.

Returns true if abs(t1.first) < abs(t2.first) (i.e., increasing).

Definition at line 500 of file [CoinSort.hpp](#).

9.30.2 Member Function Documentation

9.30.2.1 template<class S, class T, class U> bool [CoinFirstAbsLess_3](#)< S, T, U >::operator() (const [CoinTriple](#)< S, T, U > & t1, const [CoinTriple](#)< S, T, U > & t2) const [inline]

Compare function.

Definition at line 503 of file [CoinSort.hpp](#).

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinSort.hpp](#)

9.31 [CoinFirstGreater_2](#)< S, T > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (const [CoinPair](#)< S, T > &t1, const [CoinPair](#)< S, T > &t2) const
Compare function.

9.31.1 Detailed Description

template<class S, class T>class [CoinFirstGreater_2](#)< S, T >

Function operator.

Returns true if t1.first > t2.first (i.e, decreasing).

Definition at line 59 of file [CoinSort.hpp](#).

9.31.2 Member Function Documentation

9.31.2.1 `template<class S, class T> bool CoinFirstGreater_2< S, T >::operator() (const CoinPair< S, T > & t1, const CoinPair< S, T > & t2) const [inline]`

Compare function.

Definition at line 62 of file CoinSort.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp>

9.32 CoinFirstGreater_3< S, T, U > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- `bool operator() (const CoinTriple< S, T, U > &t1, const CoinTriple< S, T, U > &t2) const`
Compare function.

9.32.1 Detailed Description

```
template<class S, class T, class U> class CoinFirstGreater_3< S, T, U >
```

Function operator.

Returns true if t1.first > t2.first (i.e, decreasing).

Definition at line 489 of file CoinSort.hpp.

9.32.2 Member Function Documentation

9.32.2.1 `template<class S, class T, class U> bool CoinFirstGreater_3< S, T, U >::operator() (const CoinTriple< S, T, U > & t1, const CoinTriple< S, T, U > & t2) const [inline]`

Compare function.

Definition at line 492 of file CoinSort.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp>

9.33 CoinFirstLess_2< S, T > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- `bool operator() (const CoinPair< S, T > &t1, const CoinPair< S, T > &t2) const`
Compare function.

9.33.1 Detailed Description

```
template<class S, class T>class CoinFirstLess_2< S, T >
```

Function operator.

Returns true if `t1.first < t2.first` (i.e., increasing).

Definition at line 48 of file `CoinSort.hpp`.

9.33.2 Member Function Documentation

```
9.33.2.1 template<class S, class T > bool CoinFirstLess_2< S, T >::operator() ( const CoinPair< S, T > & t1, const
CoinPair< S, T > & t2 ) const [inline]
```

Compare function.

Definition at line 51 of file `CoinSort.hpp`.

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

- `/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp`

9.34 CoinFirstLess_3< S, T, U > Class Template Reference

Function operator.

```
#include <CoinSort.hpp>
```

Public Member Functions

- `bool operator() (const CoinTriple< S, T, U > &t1, const CoinTriple< S, T, U > &t2) const`
Compare function.

9.34.1 Detailed Description

```
template<class S, class T, class U>class CoinFirstLess_3< S, T, U >
```

Function operator.

Returns true if `t1.first < t2.first` (i.e., increasing).

Definition at line 478 of file `CoinSort.hpp`.

9.34.2 Member Function Documentation

9.34.2.1 `template<class S , class T , class U > bool CoinFirstLess_3< S, T, U >::operator() (const CoinTriple< S, T, U > & t1, const CoinTriple< S, T, U > & t2) const [inline]`

Compare function.

Definition at line 481 of file CoinSort.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp](#)

9.35 CoinLpIO::CoinHashLink Struct Reference

```
#include <CoinLpIO.hpp>
```

Public Attributes

- int [index](#)
- int [next](#)

9.35.1 Detailed Description

Definition at line 589 of file CoinLpIO.hpp.

9.35.2 Member Data Documentation

9.35.2.1 int CoinLpIO::CoinHashLink::index

Definition at line 590 of file CoinLpIO.hpp.

9.35.2.2 int CoinLpIO::CoinHashLink::next

Definition at line 590 of file CoinLpIO.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinLpIO.hpp](#)

9.36 CoinMpsIO::CoinHashLink Struct Reference

```
#include <CoinMpsIO.hpp>
```

Public Attributes

- int [index](#)
- int [next](#)

9.36.1 Detailed Description

Definition at line 897 of file CoinMpsIO.hpp.

9.36.2 Member Data Documentation

9.36.2.1 `int CoinMpslO::CoinHashLink::index`

Definition at line 898 of file `CoinMpslO.hpp`.

9.36.2.2 `int CoinMpslO::CoinHashLink::next`

Definition at line 898 of file `CoinMpslO.hpp`.

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

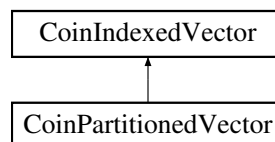
- `/home/ted/COIN/trunk/CoinUtils/src/CoinMpslO.hpp`

9.37 CoinIndexedVector Class Reference

Indexed Vector.

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for `CoinIndexedVector`:



Public Member Functions

Get methods.

- `int getNumElements () const`
Get the size.
- `const int * getIndices () const`
Get indices of elements.
- `int * getIndices ()`
Get element values.
- `double * denseVector () const`
Get the vector as a dense vector.
- `void setDenseVector (double *array)`
For very temporary use when user needs to borrow a dense vector.
- `void setIndexVector (int *array)`
For very temporary use when user needs to borrow an index vector.
- `double & operator[] (int i) const`
Access the i'th element of the full storage vector.

Set methods

- `void setNumElements (int value)`
Set the size.
- `void clear ()`
Reset the vector (as if were just created an empty vector). This leaves arrays!
- `void empty ()`

- Reset the vector (as if were just created an empty vector)*

 - `CoinIndexedVector & operator=` (const `CoinIndexedVector` &)

Assignment operator.
- `CoinIndexedVector & operator=` (const `CoinPackedVectorBase` &rhs)

Assignment operator from a `CoinPackedVectorBase`.
- `void copy` (const `CoinIndexedVector` &rhs, double multiplier=1.0)

Copy the contents of one vector into another.
- `void borrowVector` (int size, int numberIndices, int *inds, double *elems)

Borrow ownership of the arguments to this vector.
- `void returnVector` ()

Return ownership of the arguments to this vector.
- `void setVector` (int numberIndices, const int *inds, const double *elems)

Set vector numberIndices, indices, and elements.
- `void setVector` (int size, int numberIndices, const int *inds, const double *elems)

Set vector size, indices, and elements.
- `void setConstant` (int size, const int *inds, double elems)

Elements set to have the same scalar value.
- `void setFull` (int size, const double *elems)

Indices are not specified and are taken to be 0,1,...,size-1.
- `void setElement` (int index, double element)

Set an existing element in the indexed vector The first argument is the "index" into the elements() array.
- `void insert` (int index, double element)

Insert an element into the vector.
- `void quickInsert` (int index, double element)

Insert a nonzero element into the vector.
- `void add` (int index, double element)

Insert or if exists add an element into the vector Any resulting zero elements will be made tiny.
- `void quickAdd` (int index, double element)

Insert or if exists add an element into the vector Any resulting zero elements will be made tiny.
- `void quickAddNonZero` (int index, double element)

Insert or if exists add an element into the vector Any resulting zero elements will be made tiny.
- `void zero` (int index)

Makes nonzero tiny.
- int `clean` (double tolerance)

set all small values to zero and return number remaining
- int `cleanAndPack` (double tolerance)

Same but packs down.
- int `cleanAndPackSafe` (double tolerance)

Same but packs down and is safe (i.e. if order is odd)
- `void setPacked` ()

Mark as packed.
- `void checkClear` ()

For debug check vector is clear i.e. no elements.
- `void checkClean` ()

For debug check vector is clean i.e. elements match indices.
- int `scan` ()

Scan dense region and set up indices (returns number found)
- int `scan` (int start, int end)

Scan dense region from start to < end and set up indices returns number found.
- int `scan` (double tolerance)

Scan dense region and set up indices (returns number found).
- int `scan` (int start, int end, double tolerance)

Scan dense region from start to < end and set up indices returns number found.
- int `scanAndPack` ()

These are same but pack down.

- `int scanAndPack` (int start, int end)
- `int scanAndPack` (double tolerance)
- `int scanAndPack` (int start, int end, double tolerance)
- `void createPacked` (int number, const int *indices, const double *elements)
Create packed array.
- `void createUnpacked` (int number, const int *indices, const double *elements)
Create unpacked array.
- `void createOneUnpackedElement` (int index, double element)
Create unpacked singleton.
- `void expand` ()
This is mainly for testing - goes from packed to indexed.
- `void append` (const `CoinPackedVectorBase` &caboose)
Append a `CoinPackedVector` to the end.
- `void append` (const `CoinIndexedVector` &caboose)
Append a `CoinIndexedVector` to the end (with extra space)
- `void append` (`CoinIndexedVector` &other, int adjustIndex, bool zapElements=false)
Append a `CoinIndexedVector` to the end and modify indices.
- `void swap` (int i, int j)
Swap values in positions i and j of indices and elements.
- `void truncate` (int newSize)
Throw away all entries in rows >= newSize.
- `void print` () const
Print out.

Arithmetic operators.

- `void operator+=` (double value)
add value to every entry
- `void operator-=` (double value)
subtract value from every entry
- `void operator*=` (double value)
multiply every entry by value
- `void operator/=` (double value)
*divide every entry by value (** 0 vanishes)*

Comparison operators on two indexed vectors

- `bool operator==` (const `CoinPackedVectorBase` &rhs) const
Equal.
- `bool operator!=` (const `CoinPackedVectorBase` &rhs) const
Not equal.
- `bool operator==` (const `CoinIndexedVector` &rhs) const
Equal.
- `bool operator!=` (const `CoinIndexedVector` &rhs) const
Not equal.
- `int isApproximatelyEqual` (const `CoinIndexedVector` &rhs, double tolerance=1.0e-8) const
Equal with a tolerance (returns -1 or position of inequality).

Index methods

- `int getMaxIndex` () const
Get value of maximum index.
- `int getMinIndex` () const
Get value of minimum index.

Sorting

- `void sort ()`
Sort the indexed storage vector (increasing indices).
- `void sortIncrIndex ()`
- `void sortDecrIndex ()`
- `void sortIncrElement ()`
- `void sortDecrElement ()`
- `void sortPacked ()`

Arithmetic operators on packed vectors.

NOTE: These methods operate on those positions where at least one of the arguments has a value listed.

At those positions the appropriate operation is executed, Otherwise the result of the operation is considered 0.

NOTE 2: Because these methods return an object (they can't return a reference, though they could return a pointer...) they are very inefficient...

- `CoinIndexedVector operator+ (const CoinIndexedVector &op2)`
Return the sum of two indexed vectors.
- `CoinIndexedVector operator- (const CoinIndexedVector &op2)`
Return the difference of two indexed vectors.
- `CoinIndexedVector operator* (const CoinIndexedVector &op2)`
Return the element-wise product of two indexed vectors.
- `CoinIndexedVector operator/ (const CoinIndexedVector &op2)`
Return the element-wise ratio of two indexed vectors (0.0/0.0 => 0.0) (0 vanishes)
- `void operator+= (const CoinIndexedVector &op2)`
The sum of two indexed vectors.
- `void operator-= (const CoinIndexedVector &op2)`
The difference of two indexed vectors.
- `void operator*= (const CoinIndexedVector &op2)`
The element-wise product of two indexed vectors.
- `void operator/= (const CoinIndexedVector &op2)`
The element-wise ratio of two indexed vectors (0.0/0.0 => 0.0) (0 vanishes)

Memory usage

- `void reserve (int n)`
Reserve space.
- `int capacity () const`
capacity returns the size which could be accomodated without having to reallocate storage.
- `void setPackedMode (bool yesNo)`
Sets packed mode.
- `bool packedMode () const`
Gets packed mode.

Constructors and destructors

- `CoinIndexedVector ()`
Default constructor.
- `CoinIndexedVector (int size, const int *inds, const double *elems)`
Alternate Constructors - set elements to vector of doubles.
- `CoinIndexedVector (int size, const int *inds, double element)`
Alternate Constructors - set elements to same scalar value.
- `CoinIndexedVector (int size, const double *elements)`
Alternate Constructors - construct full storage with indices 0 through size-1.

- [CoinIndexedVector](#) (int size)
Alternate Constructors - just size.
- [CoinIndexedVector](#) (const [CoinIndexedVector](#) &)
Copy constructor.
- [CoinIndexedVector](#) (const [CoinIndexedVector](#) *)
Copy constructor.2.
- [CoinIndexedVector](#) (const [CoinPackedVectorBase](#) &rhs)
Copy constructor from a PackedVectorBase.
- [~CoinIndexedVector](#) ()
Destructor.

Protected Attributes

Private member data

- int * [indices_](#)
Vector indices.
- double * [elements_](#)
Vector elements.
- int [nElements_](#)
Size of indices and packed elements vectors.
- int [capacity_](#)
Amount of memory allocated for indices_, and elements_.
- int [offset_](#)
Offset to get where new allocated array.
- bool [packedMode_](#)
If true then is operating in packed mode.

Friends

- [void CoinIndexedVectorUnitTest](#) ()
A function that tests the methods in the [CoinIndexedVector](#) class.

9.37.1 Detailed Description

Indexed Vector.

This stores values unpacked but apart from that is a bit like [CoinPackedVector](#). It is designed to be lightweight in normal use.

It now has a "packed" mode when it is even more like [CoinPackedVector](#)

Indices array has capacity_ extra chars which are zeroed and can be used for any purpose - but must be re-zeroed

Stores vector of indices and associated element values. Supports sorting of indices.

Does not support negative indices.

Does NOT support testing for duplicates

`getElements` is no longer supported

Here is a sample usage:

```

const int ne = 4;
int inx[ne] = { 1, 4, 0, 2 };
double el[ne] = { 10., 40., 1., 50. };

// Create vector and set its value
CoinIndexedVector r(ne,inx,el);

// access as a full storage vector
assert( r[ 0]==1. );
assert( r[ 1]==10.);
assert( r[ 2]==50.);
assert( r[ 3]==0. );
assert( r[ 4]==40.);

// sort Elements in increasing order
r.sortIncrElement();

// access each index and element
assert( r.getIndices () [0]== 0 );
assert( r.getIndices () [1]== 1 );
assert( r.getIndices () [2]== 4 );
assert( r.getIndices () [3]== 2 );

// access as a full storage vector
assert( r[ 0]==1. );
assert( r[ 1]==10.);
assert( r[ 2]==50.);
assert( r[ 3]==0. );
assert( r[ 4]==40.);

// Tests for equality and equivalence
CoinIndexedVector r1;
r1=r;
assert( r==r1 );
assert( r.equivalent(r1) );
r.sortIncrElement();
assert( r!=r1 );
assert( r.equivalent(r1) );

// Add indexed vectors.
// Similarly for subtraction, multiplication,
// and division.
CoinIndexedVector add = r + r1;
assert( add[0] == 1.+ 1. );
assert( add[1] == 10.+10. );
assert( add[2] == 50.+50. );
assert( add[3] == 0.+ 0. );
assert( add[4] == 40.+40. );

assert( r.sum() == 10.+40.+1.+50. );

```

Definition at line 104 of file CoinIndexedVector.hpp.

9.37.2 Constructor & Destructor Documentation

9.37.2.1 CoinIndexedVector::CoinIndexedVector ()

Default constructor.

9.37.2.2 CoinIndexedVector::CoinIndexedVector (int size, const int * inds, const double * elems)

Alternate Constructors - set elements to vector of doubles.

9.37.2.3 `CoinIndexedVector::CoinIndexedVector (int size, const int * inds, double element)`

Alternate Constructors - set elements to same scalar value.

9.37.2.4 `CoinIndexedVector::CoinIndexedVector (int size, const double * elements)`

Alternate Constructors - construct full storage with indices 0 through size-1.

9.37.2.5 `CoinIndexedVector::CoinIndexedVector (int size)`

Alternate Constructors - just size.

9.37.2.6 `CoinIndexedVector::CoinIndexedVector (const CoinIndexedVector &)`

Copy constructor.

9.37.2.7 `CoinIndexedVector::CoinIndexedVector (const CoinIndexedVector *)`

Copy constructor.2.

9.37.2.8 `CoinIndexedVector::CoinIndexedVector (const CoinPackedVectorBase & rhs)`

Copy constructor *from a PackedVectorBase*.

9.37.2.9 `CoinIndexedVector::~~CoinIndexedVector ()`

Destructor.

9.37.3 Member Function Documentation

9.37.3.1 `int CoinIndexedVector::getNumElements () const [inline]`

Get the size.

Definition at line 111 of file `CoinIndexedVector.hpp`.

9.37.3.2 `const int* CoinIndexedVector::getIndices () const [inline]`

Get indices of elements.

Definition at line 113 of file `CoinIndexedVector.hpp`.

9.37.3.3 `int* CoinIndexedVector::getIndices () [inline]`

Get element values.

Get indices of elements

Definition at line 117 of file `CoinIndexedVector.hpp`.

9.37.3.4 `double* CoinIndexedVector::denseVector () const [inline]`

Get the vector as a dense vector.

This is normal storage method. The user should not not delete [] this.

Definition at line 121 of file `CoinIndexedVector.hpp`.

9.37.3.5 `void CoinIndexedVector::setDenseVector (double * array) [inline]`

For very temporary use when user needs to borrow a dense vector.

Definition at line 123 of file CoinIndexedVector.hpp.

9.37.3.6 `void CoinIndexedVector::setIndexVector (int * array) [inline]`

For very temporary use when user needs to borrow an index vector.

Definition at line 126 of file CoinIndexedVector.hpp.

9.37.3.7 `double& CoinIndexedVector::operator[] (int i) const`

Access the i'th element of the full storage vector.

9.37.3.8 `void CoinIndexedVector::setNumElements (int value) [inline]`

Set the size.

Definition at line 140 of file CoinIndexedVector.hpp.

9.37.3.9 `void CoinIndexedVector::clear ()`

Reset the vector (as if were just created an empty vector). This leaves arrays!

9.37.3.10 `void CoinIndexedVector::empty ()`

Reset the vector (as if were just created an empty vector)

9.37.3.11 `CoinIndexedVector& CoinIndexedVector::operator= (const CoinIndexedVector &)`

Assignment operator.

9.37.3.12 `CoinIndexedVector& CoinIndexedVector::operator= (const CoinPackedVectorBase & rhs)`

Assignment operator from a [CoinPackedVectorBase](#).

NOTE: This assumes no duplicates

9.37.3.13 `void CoinIndexedVector::copy (const CoinIndexedVector & rhs, double multiplier = 1.0)`

Copy the contents of one vector into another.

If multiplier is 1 It is the equivalent of = but if vectors are same size does not re-allocate memory just clears and copies

9.37.3.14 `void CoinIndexedVector::borrowVector (int size, int numberIndices, int * inds, double * elems)`

Borrow ownership of the arguments to this vector.

Size is the length of the unpacked elements vector.

9.37.3.15 `void CoinIndexedVector::returnVector ()`

Return ownership of the arguments to this vector.

State after is empty .

9.37.3.16 `void CoinIndexedVector::setVector (int numberIndices, const int * inds, const double * elems)`

Set vector numberIndices, indices, and elements.

NumberIndices is the length of both the indices and elements vectors. The indices and elements vectors are copied into this class instance's member data. Assumed to have no duplicates

9.37.3.17 void CoinIndexedVector::setVector (int size, int numberIndices, const int * inds, const double * elems)

Set vector size, indices, and elements.

Size is the length of the unpacked elements vector. The indices and elements vectors are copied into this class instance's member data. We do not check for duplicate indices

9.37.3.18 void CoinIndexedVector::setConstant (int size, const int * inds, double elems)

Elements set to have the same scalar value.

9.37.3.19 void CoinIndexedVector::setFull (int size, const double * elems)

Indices are not specified and are taken to be 0,1,...,size-1.

9.37.3.20 void CoinIndexedVector::setElement (int index, double element)

Set an existing element in the indexed vector The first argument is the "index" into the elements() array.

9.37.3.21 void CoinIndexedVector::insert (int index, double element)

Insert an element into the vector.

9.37.3.22 void CoinIndexedVector::quickInsert (int index, double element) [inline]

Insert a nonzero element into the vector.

Definition at line 193 of file CoinIndexedVector.hpp.

9.37.3.23 void CoinIndexedVector::add (int index, double element)

Insert or if exists add an element into the vector Any resulting zero elements will be made tiny.

9.37.3.24 void CoinIndexedVector::quickAdd (int index, double element) [inline]

Insert or if exists add an element into the vector Any resulting zero elements will be made tiny.

This version does no checking

Definition at line 206 of file CoinIndexedVector.hpp.

9.37.3.25 void CoinIndexedVector::quickAddNonZero (int index, double element) [inline]

Insert or if exists add an element into the vector Any resulting zero elements will be made tiny.

This knows element is nonzero This version does no checking

Definition at line 225 of file CoinIndexedVector.hpp.

9.37.3.26 void CoinIndexedVector::zero (int index) [inline]

Makes nonzero tiny.

This version does no checking

Definition at line 243 of file CoinIndexedVector.hpp.

9.37.3.27 `int CoinIndexedVector::clean (double tolerance)`

set all small values to zero and return number remaining

- $< \text{tolerance} \Rightarrow 0.0$

9.37.3.28 `int CoinIndexedVector::cleanAndPack (double tolerance)`

Same but packs down.

9.37.3.29 `int CoinIndexedVector::cleanAndPackSafe (double tolerance)`

Same but packs down and is safe (i.e. if order is odd)

9.37.3.30 `void CoinIndexedVector::setPacked () [inline]`

Mark as packed.

Definition at line 256 of file CoinIndexedVector.hpp.

9.37.3.31 `void CoinIndexedVector::checkClear ()`

For debug check vector is clear i.e. no elements.

9.37.3.32 `void CoinIndexedVector::checkClean ()`

For debug check vector is clean i.e. elements match indices.

9.37.3.33 `int CoinIndexedVector::scan ()`

Scan dense region and set up indices (returns number found)

9.37.3.34 `int CoinIndexedVector::scan (int start, int end)`

Scan dense region from start to $<$ end and set up indices returns number found.

9.37.3.35 `int CoinIndexedVector::scan (double tolerance)`

Scan dense region and set up indices (returns number found).

Only ones \geq tolerance

9.37.3.36 `int CoinIndexedVector::scan (int start, int end, double tolerance)`

Scan dense region from start to $<$ end and set up indices returns number found.

Only \geq tolerance

9.37.3.37 `int CoinIndexedVector::scanAndPack ()`

These are same but pack down.

9.37.3.38 `int CoinIndexedVector::scanAndPack (int start, int end)`**9.37.3.39** `int CoinIndexedVector::scanAndPack (double tolerance)`**9.37.3.40** `int CoinIndexedVector::scanAndPack (int start, int end, double tolerance)`

9.37.3.41 `void CoinIndexedVector::createPacked (int number, const int * indices, const double * elements)`

Create packed array.

9.37.3.42 `void CoinIndexedVector::createUnpacked (int number, const int * indices, const double * elements)`

Create unpacked array.

9.37.3.43 `void CoinIndexedVector::createOneUnpackedElement (int index, double element)`

Create unpacked singleton.

9.37.3.44 `void CoinIndexedVector::expand ()`

This is mainly for testing - goes from packed to indexed.

9.37.3.45 `void CoinIndexedVector::append (const CoinPackedVectorBase & caboose)`

Append a [CoinPackedVector](#) to the end.

9.37.3.46 `void CoinIndexedVector::append (const CoinIndexedVector & caboose)`

Append a [CoinIndexedVector](#) to the end (with extra space)

9.37.3.47 `void CoinIndexedVector::append (CoinIndexedVector & other, int adjustIndex, bool zapElements = false)`

Append a [CoinIndexedVector](#) to the end and modify indices.

9.37.3.48 `void CoinIndexedVector::swap (int i, int j)`

Swap values in positions *i* and *j* of indices and elements.

9.37.3.49 `void CoinIndexedVector::truncate (int newSize)`

Throw away all entries in rows \geq *newSize*.

9.37.3.50 `void CoinIndexedVector::print () const`

Print out.

9.37.3.51 `void CoinIndexedVector::operator+= (double value)`

add *value* to every entry

9.37.3.52 `void CoinIndexedVector::operator-= (double value)`

subtract *value* from every entry

9.37.3.53 `void CoinIndexedVector::operator*= (double value)`

multiply every entry by *value*

9.37.3.54 `void CoinIndexedVector::operator/= (double value)`

divide every entry by *value* (** 0 vanishes)

9.37.3.55 **bool** CoinIndexedVector::operator==(const CoinPackedVectorBase & rhs) const

Equal.

Returns true if vectors have same length and corresponding element of each vector is equal.

9.37.3.56 **bool** CoinIndexedVector::operator!=(const CoinPackedVectorBase & rhs) const

Not equal.

9.37.3.57 **bool** CoinIndexedVector::operator==(const CoinIndexedVector & rhs) const

Equal.

Returns true if vectors have same length and corresponding element of each vector is equal.

9.37.3.58 **bool** CoinIndexedVector::operator!=(const CoinIndexedVector & rhs) const

Not equal.

9.37.3.59 **int** CoinIndexedVector::isApproximatelyEqual (const CoinIndexedVector & rhs, double tolerance = 1.0e-8) const

Equal with a tolerance (returns -1 or position of inequality).

9.37.3.60 **int** CoinIndexedVector::getMaxIndex () const

Get value of maximum index.

9.37.3.61 **int** CoinIndexedVector::getMinIndex () const

Get value of minimum index.

9.37.3.62 **void** CoinIndexedVector::sort () [inline]

Sort the indexed storage vector (increasing indices).

Definition at line 354 of file CoinIndexedVector.hpp.

9.37.3.63 **void** CoinIndexedVector::sortIncrIndex () [inline]

Definition at line 357 of file CoinIndexedVector.hpp.

9.37.3.64 **void** CoinIndexedVector::sortDecrIndex ()

9.37.3.65 **void** CoinIndexedVector::sortIncrElement ()

9.37.3.66 **void** CoinIndexedVector::sortDecrElement ()

9.37.3.67 **void** CoinIndexedVector::sortPacked ()

9.37.3.68 **CoinIndexedVector** CoinIndexedVector::operator+ (const CoinIndexedVector & op2)

Return the sum of two indexed vectors.

9.37.3.69 **CoinIndexedVector** CoinIndexedVector::operator- (const CoinIndexedVector & op2)

Return the difference of two indexed vectors.

9.37.3.70 `CoinIndexedVector CoinIndexedVector::operator* (const CoinIndexedVector & op2)`

Return the element-wise product of two indexed vectors.

9.37.3.71 `CoinIndexedVector CoinIndexedVector::operator/ (const CoinIndexedVector & op2)`

Return the element-wise ratio of two indexed vectors (0.0/0.0 => 0.0) (0 vanishes)

9.37.3.72 `void CoinIndexedVector::operator+= (const CoinIndexedVector & op2)`

The sum of two indexed vectors.

9.37.3.73 `void CoinIndexedVector::operator-= (const CoinIndexedVector & op2)`

The difference of two indexed vectors.

9.37.3.74 `void CoinIndexedVector::operator*= (const CoinIndexedVector & op2)`

The element-wise product of two indexed vectors.

9.37.3.75 `void CoinIndexedVector::operator/= (const CoinIndexedVector & op2)`

The element-wise ratio of two indexed vectors (0.0/0.0 => 0.0) (0 vanishes)

9.37.3.76 `void CoinIndexedVector::reserve (int n)`

Reserve space.

If one knows the eventual size of the indexed vector, then it may be more efficient to reserve the space.

9.37.3.77 `int CoinIndexedVector::capacity () const [inline]`

capacity returns the size which could be accommodated without having to reallocate storage.

Definition at line 420 of file `CoinIndexedVector.hpp`.

9.37.3.78 `void CoinIndexedVector::setPackedMode (bool yesNo) [inline]`

Sets packed mode.

Definition at line 422 of file `CoinIndexedVector.hpp`.

9.37.3.79 `bool CoinIndexedVector::packedMode () const [inline]`

Gets packed mode.

Definition at line 425 of file `CoinIndexedVector.hpp`.

9.37.4 Friends And Related Function Documentation

9.37.4.1 `void CoinIndexedVectorUnitTest () [friend]`

A function that tests the methods in the [CoinIndexedVector](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

9.37.5 Member Data Documentation

9.37.5.1 `int* CoinIndexedVector::indices_` [protected]

Vector indices.

Definition at line 473 of file `CoinIndexedVector.hpp`.

9.37.5.2 `double* CoinIndexedVector::elements_` [protected]

Vector elements.

Definition at line 475 of file `CoinIndexedVector.hpp`.

9.37.5.3 `int CoinIndexedVector::nElements_` [protected]

Size of indices and packed elements vectors.

Definition at line 477 of file `CoinIndexedVector.hpp`.

9.37.5.4 `int CoinIndexedVector::capacity_` [protected]

Amount of memory allocated for `indices_`, and `elements_`.

Definition at line 479 of file `CoinIndexedVector.hpp`.

9.37.5.5 `int CoinIndexedVector::offset_` [protected]

Offset to get where new allocated array.

Definition at line 481 of file `CoinIndexedVector.hpp`.

9.37.5.6 `bool CoinIndexedVector::packedMode_` [protected]

If true then is operating in packed mode.

Definition at line 483 of file `CoinIndexedVector.hpp`.

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

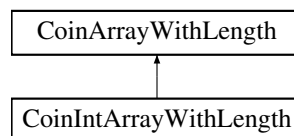
- `/home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp`

9.38 CoinIntArrayWithLength Class Reference

`int * version`

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for `CoinIntArrayWithLength`:



Public Member Functions

Get methods.

- `int getSize () const`
Get the size.
- `int * array () const`
Get Array.

Set methods

- `void setSize (int value)`
Set the size.

Condition methods

- `int * conditionalNew (int sizeWanted)`
Conditionally gets new array.

Constructors and destructors

- `CoinIntArrayWithLength ()`
Default constructor - NULL.
- `CoinIntArrayWithLength (int size)`
Alternate Constructor - length in bytes - size_ -1.
- `CoinIntArrayWithLength (int size, int mode)`
Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.
- `CoinIntArrayWithLength (const CoinIntArrayWithLength &rhs)`
Copy constructor.
- `CoinIntArrayWithLength (const CoinIntArrayWithLength *rhs)`
Copy constructor.2.
- `CoinIntArrayWithLength & operator= (const CoinIntArrayWithLength &rhs)`
Assignment operator.

Additional Inherited Members

9.38.1 Detailed Description

`int * version`

Definition at line 779 of file `CoinIndexedVector.hpp`.

9.38.2 Constructor & Destructor Documentation

9.38.2.1 `CoinIntArrayWithLength::CoinIntArrayWithLength () [inline]`

Default constructor - NULL.

Definition at line 809 of file `CoinIndexedVector.hpp`.

9.38.2.2 `CoinIntArrayWithLength::CoinIntArrayWithLength (int size) [inline]`

Alternate Constructor - length in bytes - size_ -1.

Definition at line 812 of file `CoinIndexedVector.hpp`.

9.38.2.3 `CoinIntArrayWithLength::CoinIntArrayWithLength (int size, int mode)` `[inline]`

Alternate Constructor - length in bytes mode - 0 *size_* set to size 1 *size_* set to size and zeroed.

Definition at line 818 of file `CoinIndexedVector.hpp`.

9.38.2.4 `CoinIntArrayWithLength::CoinIntArrayWithLength (const CoinIntArrayWithLength & rhs)` `[inline]`

Copy constructor.

Definition at line 821 of file `CoinIndexedVector.hpp`.

9.38.2.5 `CoinIntArrayWithLength::CoinIntArrayWithLength (const CoinIntArrayWithLength * rhs)` `[inline]`

Copy constructor.2.

Definition at line 824 of file `CoinIndexedVector.hpp`.

9.38.3 Member Function Documentation

9.38.3.1 `int CoinIntArrayWithLength::getSize () const` `[inline]`

Get the size.

Definition at line 785 of file `CoinIndexedVector.hpp`.

9.38.3.2 `int* CoinIntArrayWithLength::array () const` `[inline]`

Get Array.

Definition at line 788 of file `CoinIndexedVector.hpp`.

9.38.3.3 `void CoinIntArrayWithLength::setSize (int value)` `[inline]`

Set the size.

Definition at line 795 of file `CoinIndexedVector.hpp`.

9.38.3.4 `int* CoinIntArrayWithLength::conditionalNew (int sizeWanted)` `[inline]`

Conditionally gets new array.

Definition at line 802 of file `CoinIndexedVector.hpp`.

9.38.3.5 `CoinIntArrayWithLength& CoinIntArrayWithLength::operator= (const CoinIntArrayWithLength & rhs)`
`[inline]`

Assignment operator.

Definition at line 827 of file `CoinIndexedVector.hpp`.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp](#)

9.39 CoinLpIO Class Reference

Class to read and write Lp files.

```
#include <CoinLpIO.hpp>
```

Classes

- struct [CoinHashLink](#)

Public Member Functions

- [void convertBoundToSense](#) (const double lower, const double upper, char &sense, double &right, double &range) const
A quick inlined function to convert from lb/ub style constraint definition to sense/rhs/range style.

Constructor and Destructor

- [CoinLpIO](#) ()
Default Constructor.
- [void gutsOfDestructor](#) ()
Does the heavy lifting for destruct and assignment.
- [void gutsOfCopy](#) (const [CoinLpIO](#) &)
Does the heavy lifting for copy and assignment.
- [CoinLpIO](#) & [operator=](#) (const [CoinLpIO](#) &rhs)
assignment operator
- [CoinLpIO](#) (const [CoinLpIO](#) &)
Copy constructor.
- [~CoinLpIO](#) ()
Destructor.
- [void freePreviousNames](#) (const int section)
Free the vector previous_names_[section] and set card_previous_names_[section] to 0.
- [void freeAll](#) ()
Free all memory (except memory related to hash tables and objName_).

Queries

- const char * [getProblemName](#) () const
Get the problem name.
- [void setProblemName](#) (const char *name)
Set problem name.
- int [getNumCols](#) () const
Get number of columns.
- int [getNumRows](#) () const
Get number of rows.
- int [getNumElements](#) () const
Get number of nonzero elements.
- const double * [getColLower](#) () const
Get pointer to array[getNumCols()] of column lower bounds.
- const double * [getColUpper](#) () const
Get pointer to array[getNumCols()] of column upper bounds.
- const double * [getRowLower](#) () const
Get pointer to array[getNumRows()] of row lower bounds.
- const double * [getRowUpper](#) () const
Get pointer to array[getNumRows()] of row upper bounds.
- const char * [getRowSense](#) () const
Get pointer to array[getNumRows()] of constraint senses.
- const double * [getRightHandSide](#) () const
Get pointer to array[getNumRows()] of constraint right-hand sides.
- const double * [getRowRange](#) () const

- Get pointer to array[getNumRows()] of row ranges.*
- const double * [getObjCoefficients](#) () const
 - Get pointer to array[getNumCols()] of objective function coefficients.*
- const [CoinPackedMatrix](#) * [getMatrixByRow](#) () const
 - Get pointer to row-wise copy of the coefficient matrix.*
- const [CoinPackedMatrix](#) * [getMatrixByCol](#) () const
 - Get pointer to column-wise copy of the coefficient matrix.*
- const char * [getObjName](#) () const
 - Get objective function name.*
- void [getPreviousRowNames](#) (char const *const *prev, int *card_prev) const
 - Get pointer to array[*card_prev] of previous row names.*
- void [getPreviousColNames](#) (char const *const *prev, int *card_prev) const
 - Get pointer to array[*card_prev] of previous column names.*
- char const *const * [getRowNames](#) () const
 - Get pointer to array[getNumRows()+1] of row names, including objective function name as last entry.*
- char const *const * [getColNames](#) () const
 - Get pointer to array[getNumCols()] of column names.*
- const char * [rowName](#) (int index) const
 - Return the row name for the specified index.*
- const char * [columnName](#) (int index) const
 - Return the column name for the specified index.*
- int [rowIndex](#) (const char *name) const
 - Return the index for the specified row name.*
- int [columnIndex](#) (const char *name) const
 - Return the index for the specified column name.*
- double [objectiveOffset](#) () const
 - Returns the (constant) objective offset.*
- void [setObjectiveOffset](#) (double value)
 - Set objective offset.*
- bool [isInteger](#) (int columnNumber) const
 - Return true if a column is an integer (binary or general integer) variable.*
- const char * [integerColumns](#) () const
 - Get characteristic vector of integer variables.*

Parameters

- double [getInfinity](#) () const
 - Get infinity.*
- void [setInfinity](#) (const double)
 - Set infinity.*
- double [getEpsilon](#) () const
 - Get epsilon.*
- void [setEpsilon](#) (const double)
 - Set epsilon.*
- int [getNumberAcross](#) () const
 - Get numberAcross, the number of monomials to be printed per line.*
- void [setNumberAcross](#) (const int)
 - Set numberAcross.*
- int [getDecimals](#) () const
 - Get decimals, the number of digits to write after the decimal point.*
- void [setDecimals](#) (const int)
 - Set decimals.*

Public methods

- `void setLpDataWithoutRowAndColNames` (const `CoinPackedMatrix` &m, const double *collb, const double *colub, const double *obj_coeff, const char *integrality, const double *rowlb, const double *rowub)
Set the data of the object.
- `int is_invalid_name` (const char *buff, const bool ranged) const
Return 0 if buff is a valid name for a row, a column or objective function, return a positive number otherwise.
- `int are_invalid_names` (char const *const *vnames, const int card_vnames, const bool check_ranged) const
Return 0 if each of the card_vnames entries of vnames is a valid name, return a positive number otherwise.
- `void setDefaultRowNames` ()
Set objective function name to the default "obj" and row names to the default "cons0", "cons1", ...
- `void setDefaultColNames` ()
Set column names to the default "x0", "x1", ...
- `void setLpDataRowAndColNames` (char const *const *const rownames, char const *const *const colnames)
Set the row and column names.
- `int writeLp` (const char *filename, const double epsilon, const int numberAcross, const int decimals, const bool useRowNames=true)
Write the data in Lp format in the file with name filename.
- `int writeLp` (FILE *fp, const double epsilon, const int numberAcross, const int decimals, const bool useRowNames=true)
Write the data in Lp format in the file pointed to by the parameter fp.
- `int writeLp` (const char *filename, const bool useRowNames=true)
Write the data in Lp format in the file with name filename.
- `int writeLp` (FILE *fp, const bool useRowNames=true)
Write the data in Lp format in the file pointed to by the parameter fp.
- `void readLp` (const char *filename, const double epsilon)
Read the data in Lp format from the file with name filename, using the given value for epsilon.
- `void readLp` (const char *filename)
Read the data in Lp format from the file with name filename.
- `void readLp` (FILE *fp, const double epsilon)
Read the data in Lp format from the file stream, using the given value for epsilon.
- `void readLp` (FILE *fp)
Read the data in Lp format from the file stream.
- `void print` () const
Dump the data. Low level method for debugging.

Message handling

- `void passInMessageHandler` (`CoinMessageHandler` *handler)
Pass in Message handler.
- `void newLanguage` (`CoinMessages::Language` language)
Set the language for messages.
- `void setLanguage` (`CoinMessages::Language` language)
Set the language for messages.
- `CoinMessageHandler * messageHandler` () const
Return the message handler.
- `CoinMessages messages` ()
Return the messages.
- `CoinMessages * messagesPointer` ()
Return the messages pointer.

Protected Member Functions

- `void startHash` (char const *const *const names, const `COINColumnIndex` number, int section)
Build the hash table for the given names.

- `void stopHash` (int section)
Delete hash storage.
- `COINColumnIndex findHash` (const char *name, int section) const
Return the index of the given name, return -1 if the name is not found.
- `void insertHash` (const char *thisName, int section)
Insert thisName in the hash table if not present yet; does nothing if the name is already in.
- `void out_coeff` (FILE *fp, double v, int print_1) const
Write a coefficient.
- `int find_obj` (FILE *fp) const
Locate the objective function.
- `int is_subject_to` (const char *buff) const
Return an integer indicating if the keyword "subject to" or one of its variants has been read.
- `int first_is_number` (const char *buff) const
Return 1 if the first character of buff is a number.
- `int is_comment` (const char *buff) const
Return 1 if the first character of buff is '/' or '
- `void skip_comment` (char *buff, FILE *fp) const
Read the file fp until buff contains an end of line.
- `void scan_next` (char *buff, FILE *fp) const
Put in buff the next string that is not part of a comment.
- `int is_free` (const char *buff) const
Return 1 if buff is the keyword "free" or one of its variants.
- `int is_inf` (const char *buff) const
Return 1 if buff is the keyword "inf" or one of its variants.
- `int is_sense` (const char *buff) const
Return an integer indicating the inequality sense read.
- `int is_keyword` (const char *buff) const
Return an integer indicating if one of the keywords "Bounds", "Integers", "Generals", "Binaries", "End", or one of their variants has been read.
- `int read_monom_obj` (FILE *fp, double *coeff, char **name, int *cnt, char **obj_name)
Read a monomial of the objective function.
- `int read_monom_row` (FILE *fp, char *start_str, double *coeff, char **name, int cnt_coeff) const
Read a monomial of a constraint.
- `void realloc_coeff` (double **coeff, char ***colNames, int *maxcoeff) const
Reallocate vectors related to number of coefficients.
- `void realloc_row` (char ***rowNames, int **start, double **rhs, double **rowlow, double **rowup, int *maxrow) const
Reallocate vectors related to rows.
- `void realloc_col` (double **collow, double **colup, char **is_int, int *maxcol) const
Reallocate vectors related to columns.
- `void read_row` (FILE *fp, char *buff, double **pcoeff, char ***pcolNames, int *cnt_coeff, int *maxcoeff, double *rhs, double *rowlow, double *rowup, int *cnt_row, double inf) const
Read a constraint.
- `void checkRowNames` ()
Check that current objective name and all row names are distinct including row names obtained by adding "_low" for ranged constraints.
- `void checkColNames` ()
Check that current column names are distinct.

Protected Attributes

- char * [problemName_](#)
Problem name.
- [CoinMessageHandler](#) * [handler_](#)
Message handler.
- bool [defaultHandler_](#)
Flag to say if the message handler is the default handler.
- [CoinMessages](#) [messages_](#)
Messages.
- int [numberRows_](#)
Number of rows.
- int [numberColumns_](#)
Number of columns.
- int [numberElements_](#)
Number of elements.
- [CoinPackedMatrix](#) * [matrixByColumn_](#)
Pointer to column-wise copy of problem matrix coefficients.
- [CoinPackedMatrix](#) * [matrixByRow_](#)
Pointer to row-wise copy of problem matrix coefficients.
- double * [rowlower_](#)
Pointer to dense vector of row lower bounds.
- double * [rowupper_](#)
Pointer to dense vector of row upper bounds.
- double * [collower_](#)
Pointer to dense vector of column lower bounds.
- double * [colupper_](#)
Pointer to dense vector of column upper bounds.
- double * [rhs_](#)
Pointer to dense vector of row rhs.
- double * [rowrange_](#)
Pointer to dense vector of slack variable upper bounds for ranged constraints (undefined for non-ranged constraints)
- char * [rowsense_](#)
Pointer to dense vector of row senses.
- double * [objective_](#)
Pointer to dense vector of objective coefficients.
- double [objectiveOffset_](#)
Constant offset for objective value.
- char * [integerType_](#)
Pointer to dense vector specifying if a variable is continuous (0) or integer (1).
- char * [fileName_](#)
Current file name.
- double [infinity_](#)
Value to use for infinity.
- double [epsilon_](#)
Value to use for epsilon.
- int [numberAcross_](#)

Number of monomials printed in a row.

- int [decimals_](#)

Number of decimals printed for coefficients.

- char * [objName_](#)

Objective function name.

- char ** [previous_names_](#) [2]

Row names (including objective function name) and column names when [stopHash\(\)](#) for the corresponding section was last called or for initial names (deemed invalid) read from a file.

- int [card_previous_names_](#) [2]

[card_previous_names_\[section\]](#) holds the number of entries in the vector [previous_names_\[section\]](#).

- char ** [names_](#) [2]

Row names (including objective function name) and column names (linked to Hash tables).

- int [maxHash_](#) [2]

Maximum number of entries in a hash table section.

- int [numberHash_](#) [2]

Number of entries in a hash table section.

- [CoinHashLink](#) * [hash_](#) [2]

Hash tables with two sections.

Friends

- [void CoinLpIOUnitTest](#) (const std::string &lpDir)

9.39.1 Detailed Description

Class to read and write Lp files.

Lp file format:

/ this is a comment

\ this too

Min

obj: $x_0 + x_1 + 3x_2 - 4.5x_{yr} + 1$

s.t.

cons1: $x_0 - x_2 - 2.3x_4 \leq 4.2$ / this is another comment

c2: $x_1 + x_2 \geq 1$

cc: $x_1 + x_2 + x_{yr} = 2$

Bounds

$0 \leq x_1 \leq 3$

$1 \geq x_2$

$x_3 = 1$

$-2 \leq x_4 \leq \text{Inf}$

x_{yr} free

Integers

x0

Generals

x1 xyr

Binaries

x2

End

Notes:

- Keywords are: Min, Max, Minimize, Maximize, s.t., Subject To, Bounds, Integers, Generals, Binaries, End, Free, Inf.
- Keywords are not case sensitive and may be in plural or singular form. They should not be used as objective, row or column names.
- Bounds, Integers, Generals, Binaries sections are optional.
- Generals and Integers are synonymous.
- Bounds section (if any) must come before Integers, Generals, and Binaries sections.
- Row names must be followed by ':' without blank space. Row names are optional. If row names are present, they must be distinct (if the k-th constraint has no given name, its name is set automatically to "consk" for k=0,...). For valid row names, see the method [is_invalid_name\(\)](#).
- Column names must be followed by a blank space. They must be distinct. For valid column names, see the method [is_invalid_name\(\)](#).
- The objective function name must be followed by ':' without blank space. Objective function name is optional (if no objective function name is given, it is set to "obj" by default). For valid objective function names, see the method [is_invalid_name\(\)](#).
- Ranged constraints are written as two constraints. If a name is given for a ranged constraint, the upper bound constraint has that name and the lower bound constraint has that name with "_low" as suffix. This should be kept in mind when assigning names to ranged constraint, as the resulting name must be distinct from all the other names and be considered valid by the method [is_invalid_name\(\)](#).
- At most one term related to any single variable may appear in the objective function; if more than one term are present, only the last one is taken into account. At most one constant term may appear in the objective function; if present, it must appear last.
- Default bounds are 0 for lower bound and +infinity for upper bound.
- Free variables get default lower bound -infinity and default upper bound +infinity. Writing "x0 Free" in an LP file means "set lower bound on x0 to -infinity".
- If more than one upper (resp. lower) bound on a variable appears in the Bounds section, the last one is the one taken into account. The bounds for a binary variable are set to 0/1 only if this bound is stronger than the bound obtained from the Bounds section.
- Numbers larger than DBL_MAX (or larger than 1e+400) in the input file might crash the code.
- A comment must start with '\' or '/'. That symbol must either be the first character of a line or be preceded by a blank space. The comment ends at the end of the line. Comments are skipped while reading an Lp file and they may be inserted anywhere.

Definition at line 99 of file CoinLpIO.hpp.

9.39.2 Constructor & Destructor Documentation

9.39.2.1 CoinLpIO::CoinLpIO ()

Default Constructor.

9.39.2.2 CoinLpIO::CoinLpIO (const CoinLpIO &)

Copy constructor.

9.39.2.3 CoinLpIO::~~CoinLpIO ()

Destructor.

9.39.3 Member Function Documentation

9.39.3.1 void CoinLpIO::gutsOfDestructor ()

Does the heavy lifting for destruct and assignment.

9.39.3.2 void CoinLpIO::gutsOfCopy (const CoinLpIO &)

Does the heavy lifting for copy and assignment.

9.39.3.3 CoinLpIO& CoinLpIO::operator= (const CoinLpIO & rhs)

assignment operator

9.39.3.4 void CoinLpIO::freePreviousNames (const int section)

Free the vector previous_names_[section] and set card_previous_names_[section] to 0.

section = 0 for row names, section = 1 for column names.

9.39.3.5 void CoinLpIO::freeAll ()

Free all memory (except memory related to hash tables and objName_).

9.39.3.6 void CoinLpIO::convertBoundToSense (const double lower, const double upper, char & sense, double & right, double & range) const [inline]

A quick inlined function to convert from lb/ub style constraint definition to sense/rhs/range style.

9.39.3.7 const char* CoinLpIO::getProblemName () const

Get the problem name.

9.39.3.8 void CoinLpIO::setProblemName (const char * name)

Set problem name.

9.39.3.9 int CoinLpIO::getNumCols () const

Get number of columns.

9.39.3.10 int CoinLpIO::getNumRows () const

Get number of rows.

9.39.3.11 int CoinLpIO::getNumElements () const

Get number of nonzero elements.

9.39.3.12 const double* CoinLpIO::getColLower () const

Get pointer to array[getNumCols()] of column lower bounds.

9.39.3.13 const double* CoinLpIO::getColUpper () const

Get pointer to array[getNumCols()] of column upper bounds.

9.39.3.14 const double* CoinLpIO::getRowLower () const

Get pointer to array[getNumRows()] of row lower bounds.

9.39.3.15 const double* CoinLpIO::getRowUpper () const

Get pointer to array[getNumRows()] of row upper bounds.

9.39.3.16 const char* CoinLpIO::getRowSense () const

Get pointer to array[getNumRows()] of constraint senses.

- 'L': \leq constraint
- 'E': = constraint
- 'G': \geq constraint
- 'R': ranged constraint
- 'N': free constraint

9.39.3.17 const double* CoinLpIO::getRightHandSide () const

Get pointer to array[getNumRows()] of constraint right-hand sides.

Given constraints with upper (rowupper) and/or lower (rowlower) bounds, the constraint right-hand side (rhs) is set as

- if rowsense()[i] == 'L' then rhs()[i] == rowupper()[i]
- if rowsense()[i] == 'G' then rhs()[i] == rowlower()[i]
- if rowsense()[i] == 'R' then rhs()[i] == rowupper()[i]
- if rowsense()[i] == 'N' then rhs()[i] == 0.0

9.39.3.18 const double* CoinLpIO::getRowRange () const

Get pointer to array[getNumRows()] of row ranges.

Given constraints with upper (rowupper) and/or lower (rowlower) bounds, the constraint range (rowrange) is set as

- if rowsense()[i] == 'R' then rowrange()[i] == rowupper()[i] - rowlower()[i]

- if `rowsense()[i] != 'R'` then `rowrange()[i]` is 0.0

Put another way, only ranged constraints have a nontrivial value for `rowrange`.

9.39.3.19 `const double* CoinLpIO::getObjCoefficients () const`

Get pointer to array[`getNumCols()`] of objective function coefficients.

9.39.3.20 `const CoinPackedMatrix* CoinLpIO::getMatrixByRow () const`

Get pointer to row-wise copy of the coefficient matrix.

9.39.3.21 `const CoinPackedMatrix* CoinLpIO::getMatrixByCol () const`

Get pointer to column-wise copy of the coefficient matrix.

9.39.3.22 `const char* CoinLpIO::getObjName () const`

Get objective function name.

9.39.3.23 `void CoinLpIO::getPreviousRowNames (char const *const *prev, int *card_prev) const`

Get pointer to array[*`card_prev`] of previous row names.

The value of *`card_prev` might be different than `getNumRows()+1` if non distinct row names were present or if no previous names were saved or if the object was holding a different problem before.

9.39.3.24 `void CoinLpIO::getPreviousColNames (char const *const *prev, int *card_prev) const`

Get pointer to array[*`card_prev`] of previous column names.

The value of *`card_prev` might be different than `getNumCols()` if non distinct column names were present or if no previous names were saved, or if the object was holding a different problem before.

9.39.3.25 `char const* const* CoinLpIO::getRowNames () const`

Get pointer to array[`getNumRows()+1`] of row names, including objective function name as last entry.

9.39.3.26 `char const* const* CoinLpIO::getColNames () const`

Get pointer to array[`getNumCols()`] of column names.

9.39.3.27 `const char* CoinLpIO::rowName (int index) const`

Return the row name for the specified index.

Return the objective function name if index = `getNumRows()`. Return 0 if the index is out of range or if row names are not defined.

9.39.3.28 `const char* CoinLpIO::columnName (int index) const`

Return the column name for the specified index.

Return 0 if the index is out of range or if column names are not defined.

9.39.3.29 `int CoinLpIO::rowIndex (const char * name) const`

Return the index for the specified row name.

Return `getNumRows()` for the objective function name. Return -1 if the name is not found.

9.39.3.30 `int CoinLpIO::columnIndex (const char * name) const`

Return the index for the specified column name.

Return -1 if the name is not found.

9.39.3.31 `double CoinLpIO::objectiveOffset () const`

Returns the (constant) objective offset.

9.39.3.32 `void CoinLpIO::setObjectiveOffset (double value) [inline]`

Set objective offset.

Definition at line 265 of file CoinLpIO.hpp.

9.39.3.33 `bool CoinLpIO::isInteger (int columnNumber) const`

Return true if a column is an integer (binary or general integer) variable.

9.39.3.34 `const char* CoinLpIO::integerColumns () const`

Get characteristic vector of integer variables.

9.39.3.35 `double CoinLpIO::getInfinity () const`

Get infinity.

9.39.3.36 `void CoinLpIO::setInfinity (const double)`

Set infinity.

Any number larger is considered infinity. Default: DBL_MAX

9.39.3.37 `double CoinLpIO::getEpsilon () const`

Get epsilon.

9.39.3.38 `void CoinLpIO::setEpsilon (const double)`

Set epsilon.

Default: 1e-5.

9.39.3.39 `int CoinLpIO::getNumberAcross () const`

Get numberAcross, the number of monomials to be printed per line.

9.39.3.40 `void CoinLpIO::setNumberAcross (const int)`

Set numberAcross.

Default: 10.

9.39.3.41 `int CoinLpIO::getDecimals () const`

Get decimals, the number of digits to write after the decimal point.

9.39.3.42 void CoinLpIO::setDecimals (const int)

Set decimals.

Default: 5

9.39.3.43 void CoinLpIO::setLpDataWithoutRowAndColNames (const CoinPackedMatrix & m, const double * collb, const double * colub, const double * obj_coeff, const char * integrality, const double * rowlb, const double * rowub)

Set the data of the object.

Set it from the coefficient matrix m, the lower bounds collb, the upper bounds colub, objective function obj_coeff, integrality vector integrality, lower/upper bounds on the constraints. The sense of optimization of the objective function is assumed to be a minimization. Numbers larger than DBL_MAX (or larger than 1e+400) might crash the code.

9.39.3.44 int CoinLpIO::is_invalid_name (const char * buff, const bool ranged) const

Return 0 if buff is a valid name for a row, a column or objective function, return a positive number otherwise.

If parameter ranged = true, the name is intended for a ranged constraint.

Return 1 if the name has more than 100 characters (96 characters for a ranged constraint name, as "_low" will be added to the name).

Return 2 if the name starts with a number.

Return 3 if the name is not built with the letters a to z, A to Z, the numbers 0 to 9 or the characters " ! # \$ % & () . ; ? @ _ ' { } ~

Return 4 if the name is a keyword.

Return 5 if the name is empty or NULL.

9.39.3.45 int CoinLpIO::are_invalid_names (char const *const * vnames, const int card_vnames, const bool check_ranged) const

Return 0 if each of the card_vnames entries of vnames is a valid name, return a positive number otherwise.

The return value, if not 0, is the return value of [is_invalid_name\(\)](#) for the last invalid name in vnames. If check_ranged = true, the names are row names and names for ranged constraints must be checked for additional restrictions since "_low" will be added to the name if an Lp file is written. When check_ranged = true, card_vnames must have [getNumRows\(\)+1](#) entries, with entry vnames[[getNumRows\(\)](#)] being the name of the objective function. For a description of valid names and return values, see the method [is_invalid_name\(\)](#).

This method must not be called with check_ranged = true before [setLpDataWithoutRowAndColNames\(\)](#) has been called, since access to the indices of all the ranged constraints is required.

9.39.3.46 void CoinLpIO::setDefaultRowNames ()

Set objective function name to the default "obj" and row names to the default "cons0", "cons1", ...

9.39.3.47 void CoinLpIO::setDefaultColNames ()

Set column names to the default "x0", "x1", ...

9.39.3.48 void CoinLpIO::setLpDataRowAndColNames (char const *const *const rownames, char const *const *const colnames)

Set the row and column names.

The array rownames must either be NULL or have exactly [getNumRows\(\)+1](#) distinct entries, each of them being a valid name (see [is_invalid_name\(\)](#)) and the last entry being the intended name for the objective function. If rownames is NULL, existing row names and objective function name are not changed. If rownames is deemed invalid, default row

names and objective function name are used (see [setDefaultRowNames\(\)](#)). The memory location of array rownames (or its entries) should not be related to the memory location of the array (or entries) obtained from [getRowNames\(\)](#) or [getPreviousRowNames\(\)](#), as the call to [setLpDataRowAndColNames\(\)](#) modifies the corresponding arrays. Unpredictable results are obtained if this requirement is ignored.

Similar remarks apply to the array colnames, which must either be NULL or have exactly [getNumCols\(\)](#) entries.

9.39.3.49 `int CoinLpIO::writeLp (const char * filename, const double epsilon, const int numberAcross, const int decimals, const bool useRowNames = true)`

Write the data in Lp format in the file with name filename.

Coefficients with value less than epsilon away from an integer value are written as integers. Write at most numberAcross monomials on a line. Write non integer numbers with decimals digits after the decimal point. Write objective function name and row names if useRowNames = true.

Ranged constraints are written as two constraints. If row names are used, the upper bound constraint has the name of the original ranged constraint and the lower bound constraint has for name the original name with "_low" as suffix. If doing so creates two identical row names, default row names are used (see [setDefaultRowNames\(\)](#)).

9.39.3.50 `int CoinLpIO::writeLp (FILE * fp, const double epsilon, const int numberAcross, const int decimals, const bool useRowNames = true)`

Write the data in Lp format in the file pointed to by the parameter fp.

Coefficients with value less than epsilon away from an integer value are written as integers. Write at most numberAcross monomials on a line. Write non integer numbers with decimals digits after the decimal point. Write objective function name and row names if useRowNames = true.

Ranged constraints are written as two constraints. If row names are used, the upper bound constraint has the name of the original ranged constraint and the lower bound constraint has for name the original name with "_low" as suffix. If doing so creates two identical row names, default row names are used (see [setDefaultRowNames\(\)](#)).

9.39.3.51 `int CoinLpIO::writeLp (const char * filename, const bool useRowNames = true)`

Write the data in Lp format in the file with name filename.

Write objective function name and row names if useRowNames = true.

9.39.3.52 `int CoinLpIO::writeLp (FILE * fp, const bool useRowNames = true)`

Write the data in Lp format in the file pointed to by the parameter fp.

Write objective function name and row names if useRowNames = true.

9.39.3.53 `void CoinLpIO::readLp (const char * filename, const double epsilon)`

Read the data in Lp format from the file with name filename, using the given value for epsilon.

If the original problem is a maximization problem, the objective function is immediately flipped to get a minimization problem.

9.39.3.54 `void CoinLpIO::readLp (const char * filename)`

Read the data in Lp format from the file with name filename.

If the original problem is a maximization problem, the objective function is immediately flipped to get a minimization problem.

9.39.3.55 void CoinLpIO::readLp (FILE * *fp*, const double *epsilon*)

Read the data in Lp format from the file stream, using the given value for epsilon.

If the original problem is a maximization problem, the objective function is immediadtly flipped to get a minimization problem.

9.39.3.56 void CoinLpIO::readLp (FILE * *fp*)

Read the data in Lp format from the file stream.

If the original problem is a maximization problem, the objective function is immediadtly flipped to get a minimization problem.

9.39.3.57 void CoinLpIO::print () const

Dump the data. Low level method for debugging.

9.39.3.58 void CoinLpIO::passInMessageHandler (CoinMessageHandler * *handler*)

Pass in Message handler.

Supply a custom message handler. It will not be destroyed when the [CoinMpsIO](#) object is destroyed.

9.39.3.59 void CoinLpIO::newLanguage (CoinMessages::Language *language*)

Set the language for messages.

9.39.3.60 void CoinLpIO::setLanguage (CoinMessages::Language *language*) [inline]

Set the language for messages.

Definition at line 477 of file CoinLpIO.hpp.

9.39.3.61 CoinMessageHandler* CoinLpIO::messageHandler () const [inline]

Return the message handler.

Definition at line 480 of file CoinLpIO.hpp.

9.39.3.62 CoinMessages CoinLpIO::messages () [inline]

Return the messages.

Definition at line 483 of file CoinLpIO.hpp.

9.39.3.63 CoinMessages* CoinLpIO::messagesPointer () [inline]

Return the messages pointer.

Definition at line 485 of file CoinLpIO.hpp.

9.39.3.64 void CoinLpIO::startHash (char const *const *const *names*, const COINColumnIndex *number*, int *section*) [protected]

Build the hash table for the given names.

The parameter number is the cardinality of parameter names. Remove duplicate names.

section = 0 for row names, section = 1 for column names.

9.39.3.65 void CoinLpIO::stopHash (int *section*) [protected]

Delete hash storage.

If section = 0, it also frees objName_. section = 0 for row names, section = 1 for column names.

9.39.3.66 COINColumnIndex CoinLpIO::findHash (const char * *name*, int *section*) const [protected]

Return the index of the given name, return -1 if the name is not found.

Return [getNumRows\(\)](#) for the objective function name. section = 0 for row names (including objective function name), section = 1 for column names.

9.39.3.67 void CoinLpIO::insertHash (const char * *thisName*, int *section*) [protected]

Insert thisName in the hash table if not present yet; does nothing if the name is already in.

section = 0 for row names, section = 1 for column names.

9.39.3.68 void CoinLpIO::out_coeff (FILE * *fp*, double *v*, int *print_1*) const [protected]

Write a coefficient.

print_1 = 0 : do not print the value 1.

9.39.3.69 int CoinLpIO::find_obj (FILE * *fp*) const [protected]

Locate the objective function.

Return 1 if found the keyword "Minimize" or one of its variants, -1 if found keyword "Maximize" or one of its variants.

9.39.3.70 int CoinLpIO::is_subject_to (const char * *buff*) const [protected]

Return an integer indicating if the keyword "subject to" or one of its variants has been read.

Return 1 if buff is the keyword "s.t" or one of its variants. Return 2 if buff is the keyword "subject" or one of its variants. Return 0 otherwise.

9.39.3.71 int CoinLpIO::first_is_number (const char * *buff*) const [protected]

Return 1 if the first character of buff is a number.

Return 0 otherwise.

9.39.3.72 int CoinLpIO::is_comment (const char * *buff*) const [protected]

Return 1 if the first character of buff is '/' or '\'.

Return 0 otherwise.

9.39.3.73 void CoinLpIO::skip_comment (char * *buff*, FILE * *fp*) const [protected]

Read the file fp until buff contains an end of line.

9.39.3.74 void CoinLpIO::scan_next (char * *buff*, FILE * *fp*) const [protected]

Put in buff the next string that is not part of a comment.

9.39.3.75 int CoinLpIO::is_free (const char * *buff*) const [protected]

Return 1 if buff is the keyword "free" or one of its variants.

Return 0 otherwise.

9.39.3.76 `int CoinLpIO::is_inf (const char * buff) const` [protected]

Return 1 if buff is the keyword "inf" or one of its variants.

Return 0 otherwise.

9.39.3.77 `int CoinLpIO::is_sense (const char * buff) const` [protected]

Return an integer indicating the inequality sense read.

Return 0 if buff is ' \leq '. Return 1 if buff is ' $=$ '. Return 2 if buff is ' \geq '. Return -1 otherwise.

9.39.3.78 `int CoinLpIO::is_keyword (const char * buff) const` [protected]

Return an integer indicating if one of the keywords "Bounds", "Integers", "Generals", "Binaries", "End", or one of their variants has been read.

Return 1 if buff is the keyword "Bounds" or one of its variants. Return 2 if buff is the keyword "Integers" or "Generals" or one of their variants. Return 3 if buff is the keyword "Binaries" or one of its variants. Return 4 if buff is the keyword "End" or one of its variants. Return 0 otherwise.

9.39.3.79 `int CoinLpIO::read_monom_obj (FILE * fp, double * coeff, char ** name, int * cnt, char ** obj_name)`
[protected]

Read a monomial of the objective function.

Return 1 if "subject to" or one of its variants has been read.

9.39.3.80 `int CoinLpIO::read_monom_row (FILE * fp, char * start_str, double * coeff, char ** name, int cnt_coeff) const`
[protected]

Read a monomial of a constraint.

Return a positive number if the sense of the inequality has been read (see method [is_sense\(\)](#) for the return code). Return -1 otherwise.

9.39.3.81 `void CoinLpIO::realloc_coeff (double ** coeff, char *** colNames, int * maxcoeff) const` [protected]

Reallocate vectors related to number of coefficients.

9.39.3.82 `void CoinLpIO::realloc_row (char *** rowNames, int ** start, double ** rhs, double ** rowlow, double ** rowup, int * maxrow) const` [protected]

Reallocate vectors related to rows.

9.39.3.83 `void CoinLpIO::realloc_col (double ** colrow, double ** colup, char ** is_int, int * maxcol) const`
[protected]

Reallocate vectors related to columns.

9.39.3.84 `void CoinLpIO::read_row (FILE * fp, char * buff, double ** pcoeff, char *** pcolNames, int * cnt_coeff, int * maxcoeff, double * rhs, double * rowlow, double * rowup, int * cnt_row, double inf) const` [protected]

Read a constraint.

9.39.3.85 void CoinLpIO::checkRowNames () [protected]

Check that current objective name and all row names are distinct including row names obtained by adding "_low" for ranged constraints.

If there is a conflict in the names, they are replaced by default row names (see [setDefaultRowNames\(\)](#)).

This method must not be called before [setLpDataWithoutRowAndColNames\(\)](#) has been called, since access to the indices of all the ranged constraints is required.

This method must not be called before [setLpDataRowAndColNames\(\)](#) has been called, since access to all the row names is required.

9.39.3.86 void CoinLpIO::checkColNames () [protected]

Check that current column names are distinct.

If not, they are replaced by default column names (see [setDefaultColNames\(\)](#)).

This method must not be called before [setLpDataRowAndColNames\(\)](#) has been called, since access to all the column names is required.

9.39.4 Friends And Related Function Documentation

9.39.4.1 void CoinLpIOUnitTest (const std::string & lpDir) [friend]

9.39.5 Member Data Documentation

9.39.5.1 char* CoinLpIO::problemName_ [protected]

Problem name.

Definition at line 490 of file CoinLpIO.hpp.

9.39.5.2 CoinMessageHandler* CoinLpIO::handler_ [protected]

Message handler.

Definition at line 493 of file CoinLpIO.hpp.

9.39.5.3 bool CoinLpIO::defaultHandler_ [protected]

Flag to say if the message handler is the default handler.

If true, the handler will be destroyed when the [CoinMpsIO](#) object is destroyed; if false, it will not be destroyed.

Definition at line 499 of file CoinLpIO.hpp.

9.39.5.4 CoinMessages CoinLpIO::messages_ [protected]

Messages.

Definition at line 501 of file CoinLpIO.hpp.

9.39.5.5 int CoinLpIO::numberOfRows_ [protected]

Number of rows.

Definition at line 504 of file CoinLpIO.hpp.

9.39.5.6 `int CoinLpIO::numberColumns_` `[protected]`

Number of columns.

Definition at line 507 of file CoinLpIO.hpp.

9.39.5.7 `int CoinLpIO::numberElements_` `[protected]`

Number of elements.

Definition at line 510 of file CoinLpIO.hpp.

9.39.5.8 `CoinPackedMatrix* CoinLpIO::matrixByColumn_` `[mutable], [protected]`

Pointer to column-wise copy of problem matrix coefficients.

Definition at line 513 of file CoinLpIO.hpp.

9.39.5.9 `CoinPackedMatrix* CoinLpIO::matrixByRow_` `[protected]`

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 516 of file CoinLpIO.hpp.

9.39.5.10 `double* CoinLpIO::rowlower_` `[protected]`

Pointer to dense vector of row lower bounds.

Definition at line 519 of file CoinLpIO.hpp.

9.39.5.11 `double* CoinLpIO::rowupper_` `[protected]`

Pointer to dense vector of row upper bounds.

Definition at line 522 of file CoinLpIO.hpp.

9.39.5.12 `double* CoinLpIO::collower_` `[protected]`

Pointer to dense vector of column lower bounds.

Definition at line 525 of file CoinLpIO.hpp.

9.39.5.13 `double* CoinLpIO::colupper_` `[protected]`

Pointer to dense vector of column upper bounds.

Definition at line 528 of file CoinLpIO.hpp.

9.39.5.14 `double* CoinLpIO::rhs_` `[mutable], [protected]`

Pointer to dense vector of row rhs.

Definition at line 531 of file CoinLpIO.hpp.

9.39.5.15 `double* CoinLpIO::rowrange_` `[mutable], [protected]`

Pointer to dense vector of slack variable upper bounds for ranged constraints (undefined for non-ranged constraints)

Definition at line 536 of file CoinLpIO.hpp.

9.39.5.16 `char* CoinLpIO::rowsense_` [mutable], [protected]

Pointer to dense vector of row senses.

Definition at line 539 of file CoinLpIO.hpp.

9.39.5.17 `double* CoinLpIO::objective_` [protected]

Pointer to dense vector of objective coefficients.

Definition at line 542 of file CoinLpIO.hpp.

9.39.5.18 `double CoinLpIO::objectiveOffset_` [protected]

Constant offset for objective value.

Definition at line 545 of file CoinLpIO.hpp.

9.39.5.19 `char* CoinLpIO::integerType_` [protected]

Pointer to dense vector specifying if a variable is continuous (0) or integer (1).

Definition at line 549 of file CoinLpIO.hpp.

9.39.5.20 `char* CoinLpIO::fileName_` [protected]

Current file name.

Definition at line 552 of file CoinLpIO.hpp.

9.39.5.21 `double CoinLpIO::infinity_` [protected]

Value to use for infinity.

Definition at line 555 of file CoinLpIO.hpp.

9.39.5.22 `double CoinLpIO::epsilon_` [protected]

Value to use for epsilon.

Definition at line 558 of file CoinLpIO.hpp.

9.39.5.23 `int CoinLpIO::numberAcross_` [protected]

Number of monomials printed in a row.

Definition at line 561 of file CoinLpIO.hpp.

9.39.5.24 `int CoinLpIO::decimals_` [protected]

Number of decimals printed for coefficients.

Definition at line 564 of file CoinLpIO.hpp.

9.39.5.25 `char* CoinLpIO::objName_` [protected]

Objective function name.

Definition at line 567 of file CoinLpIO.hpp.

9.39.5.26 `char** CoinLpIO::previous_names_[2]` `[protected]`

Row names (including objective function name) and column names when `stopHash()` for the corresponding section was last called or for initial names (deemed invalid) read from a file.

section = 0 for row names, section = 1 for column names.

Definition at line 575 of file `CoinLpIO.hpp`.

9.39.5.27 `int CoinLpIO::card_previous_names_[2]` `[protected]`

`card_previous_names_[section]` holds the number of entries in the vector `previous_names_[section]`.

section = 0 for row names, section = 1 for column names.

Definition at line 581 of file `CoinLpIO.hpp`.

9.39.5.28 `char** CoinLpIO::names_[2]` `[protected]`

Row names (including objective function name) and column names (linked to Hash tables).

section = 0 for row names, section = 1 for column names.

Definition at line 587 of file `CoinLpIO.hpp`.

9.39.5.29 `int CoinLpIO::maxHash_[2]` `[protected]`

Maximum number of entries in a hash table section.

section = 0 for row names, section = 1 for column names.

Definition at line 596 of file `CoinLpIO.hpp`.

9.39.5.30 `int CoinLpIO::numberHash_[2]` `[protected]`

Number of entries in a hash table section.

section = 0 for row names, section = 1 for column names.

Definition at line 601 of file `CoinLpIO.hpp`.

9.39.5.31 `CoinHashLink* CoinLpIO::hash_[2]` `[mutable], [protected]`

Hash tables with two sections.

section = 0 for row names (including objective function name), section = 1 for column names.

Definition at line 606 of file `CoinLpIO.hpp`.

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

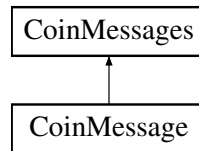
- `/home/ted/COIN/trunk/CoinUtils/src/CoinLpIO.hpp`

9.40 CoinMessage Class Reference

The standard set of `Coin` messages.

```
#include <CoinMessage.hpp>
```

Inheritance diagram for `CoinMessage`:



Public Member Functions

Constructors etc

- `CoinMessage (Language language=us_en)`
Constructor.

Additional Inherited Members

9.40.1 Detailed Description

The standard set of [Coin](#) messages.

This class provides convenient access to the standard set of [Coin](#) messages. In a nutshell, it's a [CoinMessages](#) object with a constructor that preloads the standard [Coin](#) messages.

Definition at line 79 of file `CoinMessage.hpp`.

9.40.2 Constructor & Destructor Documentation

9.40.2.1 `CoinMessage::CoinMessage (Language language = us_en)`

Constructor.

Build a [CoinMessages](#) object and load it with the standard set of [Coin](#) messages.

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

- `/home/ted/COIN/trunk/CoinUtils/src/CoinMessage.hpp`

9.41 CoinMessageHandler Class Reference

Base class for message handling.

```
#include <CoinMessageHandler.hpp>
```

Public Member Functions

Virtual methods that the derived classes may provide

- virtual int `print ()`
Print message, return 0 normally.
- virtual void `checkSeverity ()`
Check message severity - if too bad then abort.

Constructors etc

- [CoinMessageHandler](#) ()
Constructor.
- [CoinMessageHandler](#) (FILE *fp)
Constructor to put to file pointer (won't be closed)
- virtual [~CoinMessageHandler](#) ()
Destructor.
- [CoinMessageHandler](#) (const [CoinMessageHandler](#) &)
The copy constructor.
- [CoinMessageHandler](#) & [operator=](#) (const [CoinMessageHandler](#) &)
Assignment operator.
- virtual [CoinMessageHandler](#) * [clone](#) () const
Clone.

Get and set methods

- int [detail](#) (int messageNumber, const [CoinMessages](#) &normalMessage) const
Get detail level of a message.
- int [logLevel](#) () const
Get current log (detail) level.
- void [setLogLevel](#) (int value)
Set current log (detail) level.
- int [logLevel](#) (int which) const
Get alternative log level.
- void [setLogLevel](#) (int which, int value)
Set alternative log level value.
- void [setPrecision](#) (unsigned int new_precision)
Set the number of significant digits for printing floating point numbers.
- int [precision](#) ()
Current number of significant digits for printing floating point numbers.
- void [setPrefix](#) (bool yesNo)
Switch message prefix on or off.
- bool [prefix](#) () const
Current setting for printing message prefix.
- double [doubleValue](#) (int position) const
Values of double fields already processed.
- int [numberDoubleFields](#) () const
Number of double fields already processed.
- int [intValue](#) (int position) const
Values of integer fields already processed.
- int [numberIntFields](#) () const
Number of integer fields already processed.
- char [charValue](#) (int position) const
Values of char fields already processed.
- int [numberCharFields](#) () const
Number of char fields already processed.
- std::string [stringValue](#) (int position) const
Values of string fields already processed.
- int [numberStringFields](#) () const
Number of string fields already processed.
- [CoinOneMessage](#) [currentMessage](#) () const
Current message.
- std::string [currentSource](#) () const
Source of current message.
- const char * [messageBuffer](#) () const
Output buffer.

- `int highestNumber () const`
Highest message number (indicates any errors)
- `FILE * filePointer () const`
Get current file pointer.
- `void setFilePointer (FILE *fp)`
Set new file pointer.

Actions to create a message

- `CoinMessageHandler & message (int messageNumber, const CoinMessages &messages)`
Start a message.
- `CoinMessageHandler & message (int detail=-1)`
Start or continue a message.
- `CoinMessageHandler & message (int externalNumber, const char *source, const char *msg, char severity, int detail=-1)`
Print a complete message.
- `CoinMessageHandler & operator<< (int intvalue)`
Process an integer parameter value.
- `CoinMessageHandler & operator<< (double doublevalue)`
Process a double parameter value.
- `CoinMessageHandler & operator<< (const std::string &stringvalue)`
Process a STL string parameter value.
- `CoinMessageHandler & operator<< (char charvalue)`
Process a char parameter value.
- `CoinMessageHandler & operator<< (const char *stringvalue)`
Process a C-style string parameter value.
- `CoinMessageHandler & operator<< (CoinMessageMarker)`
Process a marker.
- `int finish ()`
Finish (and print) the message.
- `CoinMessageHandler & printing (bool onOff)`
Enable or disable printing of an optional portion of a message.

Protected Attributes

Protected member data

- `std::vector< double > doubleValue_`
values in message
- `std::vector< int > longValue_`
- `std::vector< char > charValue_`
- `std::vector< std::string > stringValue_`
- `int logLevel_`
Log level.
- `int logLevels_ [COIN_NUM_LOG]`
Log levels.
- `int prefix_`
Whether we want prefix (may get more subtle so is int)
- `CoinOneMessage currentMessage_`
Current message.
- `int internalNumber_`
Internal number for use with enums.
- `char * format_`
Format string for message (remainder)

- char [messageBuffer_](#) [COIN_MESSAGE_HANDLER_MAX_BUFFER_SIZE]
Output buffer.
- char * [messageOut_](#)
Position in output buffer.
- std::string [source_](#)
Current source of message.
- int [printStatus_](#)
0 - Normal.
- int [highestNumber_](#)
Highest message number (indicates any errors)
- FILE * [fp_](#)
File pointer.
- char [g_format_](#) [8]
Current format for floating point numbers.
- int [g_precision_](#)
Current number of significant digits for floating point numbers.

Friends

- bool [CoinMessageHandlerUnitTest](#) ()
A function that tests the methods in the [CoinMessageHandler](#) class.

9.41.1 Detailed Description

Base class for message handling.

The default behavior is described here: messages are printed, and (if the severity is sufficiently high) execution will be aborted. Inherit and redefine the methods [print](#) and [checkSeverity](#) to augment the behaviour.

Messages can be printed with or without a prefix; the prefix will consist of a source string, the external ID number, and a letter code, e.g., Clp6024W. A prefix makes the messages look less nimble but is very useful for "grep" etc.

Usage

The general approach to using the COIN messaging facility is as follows:

- Define your messages. For each message, you must supply an external ID number, a log (detail) level, and a format string. Typically, you define a convenience structure for this, something that's easy to use to create an array of initialised message definitions at compile time.
- Create a [CoinMessages](#) object, sized to accommodate the number of messages you've defined. (Incremental growth will happen if necessary as messages are loaded, but it's inefficient.)
- Load the messages into the [CoinMessages](#) object. Typically this entails creating a [CoinOneMessage](#) object for each message and passing it as a parameter to [CoinMessages::addMessage\(\)](#). You specify the message's internal ID as the other parameter to [addMessage](#).
- Create and use a [CoinMessageHandler](#) object to print messages.

See, for example, [CoinMessage.hpp](#) and [CoinMessage.cpp](#) for an example of the first three steps. 'Format codes' below has a simple example of printing a message.

External ID numbers and severity

[CoinMessageHandler](#) assumes the following relationship between the external ID number of a message and the severity of the message:

- <3000 are informational ('I')
- <6000 warnings ('W')
- <9000 non-fatal errors ('E')
- >=9000 aborts the program (after printing the message) ('S')

Log (detail) levels

The default behaviour is that a message will print if its detail level is less than or equal to the handler's log level. If all you want to do is set a single log level for the handler, use [setLogLevel\(int\)](#).

If you want to get fancy, here's how it really works: There's an array, [logLevels_](#), which you can manipulate with [setLogLevel\(int,int\)](#). Each entry [logLevels_\[i\]](#) specifies the log level for messages of class *i* (see [CoinMessages::class_](#)). If [logLevels_\[0\]](#) is set to the magic number -1000 you get the simple behaviour described above, whatever the class of the messages. If [logLevels_\[0\]](#) is set to a valid log level (≥ 0), then [logLevels_\[i\]](#) really is the log level for messages of class *i*.

Format codes

[CoinMessageHandler](#) can print integers (normal, long, and long long), doubles, characters, and strings. See the descriptions of the various `<<` operators.

When processing a standard message with a format string, the formatting codes specified in the format string will be passed to the `sprintf` function, along with the argument. When generating a message with no format string, each `<<` operator uses a simple format code appropriate for its argument. Consult the documentation for the standard `printf` facility for further information on format codes.

The special format code `'%?'` provides a hook to enable or disable printing. For each `'%?'` code, there must be a corresponding call to [printing\(bool\)](#). This provides a way to define optional parts in messages, delineated by the code `'%?'` in the format string. Printing can be suppressed for these optional parts, but any operands must still be supplied. For example, given the message string

```
"A message with%? an optional integer %d and%? a double %g."
```

installed in [CoinMessages](#) `exampleMsgs` with index 5, and [CoinMessageHandler](#) `hdl`, the code

```
hdl.message(5,exampleMsgs) ;
hdl.printing(true) << 42 ;
hdl.printing(true) << 53.5 << CoinMessageEol ;
```

will print

```
A message with an optional integer 42 and a double 53.5.
```

while

```
hdl.message(5,exampleMsgs) ;
hdl.printing(false) << 42 ;
hdl.printing(true) << 53.5 << CoinMessageEol ;
```

will print

```
A message with a double 53.5.
```

For additional examples of usage, see `CoinMessageHandlerUnitTest` in `CoinMessageHandlerTest.cpp`.

Definition at line 327 of file `CoinMessageHandler.hpp`.

9.41.2 Constructor & Destructor Documentation

9.41.2.1 CoinMessageHandler::CoinMessageHandler ()

Constructor.

9.41.2.2 CoinMessageHandler::CoinMessageHandler (FILE * *fp*)

Constructor to put to file pointer (won't be closed)

9.41.2.3 virtual CoinMessageHandler::~CoinMessageHandler () [virtual]

Destructor.

9.41.2.4 CoinMessageHandler::CoinMessageHandler (const CoinMessageHandler &)

The copy constructor.

9.41.3 Member Function Documentation

9.41.3.1 virtual int CoinMessageHandler::print () [virtual]

Print message, return 0 normally.

9.41.3.2 virtual void CoinMessageHandler::checkSeverity () [virtual]

Check message severity - if too bad then abort.

9.41.3.3 CoinMessageHandler& CoinMessageHandler::operator= (const CoinMessageHandler &)

Assignment operator.

9.41.3.4 virtual CoinMessageHandler* CoinMessageHandler::clone () const [virtual]

Clone.

9.41.3.5 int CoinMessageHandler::detail (int *messageNumber*, const CoinMessages & *normalMessage*) const [inline]

Get detail level of a message.

Definition at line 360 of file CoinMessageHandler.hpp.

9.41.3.6 int CoinMessageHandler::logLevel () const [inline]

Get current log (detail) level.

Definition at line 363 of file CoinMessageHandler.hpp.

9.41.3.7 void CoinMessageHandler::setLogLevel (int *value*)

Set current log (detail) level.

If the log level is equal or greater than the detail level of a message, the message will be printed. A rough convention for the amount of output expected is

- 0 - none
- 1 - minimal

- 2 - normal low
- 3 - normal high
- 4 - verbose

Please assign log levels to messages accordingly. Log levels of 8 and above (8,16,32, *etc.*) are intended for selective debugging. The logical AND of the log level specified in the message and the current log level is used to determine if the message is printed. (In other words, you're using individual bits to determine which messages are printed.)

9.41.3.8 `int CoinMessageHandler::logLevel (int which) const` `[inline]`

Get alternative log level.

Definition at line 384 of file CoinMessageHandler.hpp.

9.41.3.9 `void CoinMessageHandler::setLogLevel (int which, int value)`

Set alternative log level value.

Can be used to store alternative log level information within the handler.

9.41.3.10 `void CoinMessageHandler::setPrecision (unsigned int new_precision)`

Set the number of significant digits for printing floating point numbers.

9.41.3.11 `int CoinMessageHandler::precision ()` `[inline]`

Current number of significant digits for printing floating point numbers.

Definition at line 395 of file CoinMessageHandler.hpp.

9.41.3.12 `void CoinMessageHandler::setPrefix (bool yesNo)`

Switch message prefix on or off.

9.41.3.13 `bool CoinMessageHandler::prefix () const`

Current setting for printing message prefix.

9.41.3.14 `double CoinMessageHandler::doubleValue (int position) const` `[inline]`

Values of double fields already processed.

As the parameter for a double field is processed, the value is saved and can be retrieved using this function.

Definition at line 406 of file CoinMessageHandler.hpp.

9.41.3.15 `int CoinMessageHandler::numberDoubleFields () const` `[inline]`

Number of double fields already processed.

Incremented each time a field of type double is processed.

Definition at line 412 of file CoinMessageHandler.hpp.

9.41.3.16 `int CoinMessageHandler::intValue (int position) const` `[inline]`

Values of integer fields already processed.

As the parameter for a integer field is processed, the value is saved and can be retrieved using this function.

Definition at line 419 of file CoinMessageHandler.hpp.

9.41.3.17 `int CoinMessageHandler::numberIntFields () const [inline]`

Number of integer fields already processed.

Incremented each time a field of type integer is processed.

Definition at line 425 of file CoinMessageHandler.hpp.

9.41.3.18 `char CoinMessageHandler::charValue (int position) const [inline]`

Values of char fields already processed.

As the parameter for a char field is processed, the value is saved and can be retrieved using this function.

Definition at line 432 of file CoinMessageHandler.hpp.

9.41.3.19 `int CoinMessageHandler::numberCharFields () const [inline]`

Number of char fields already processed.

Incremented each time a field of type char is processed.

Definition at line 438 of file CoinMessageHandler.hpp.

9.41.3.20 `std::string CoinMessageHandler::stringValue (int position) const [inline]`

Values of string fields already processed.

As the parameter for a string field is processed, the value is saved and can be retrieved using this function.

Definition at line 445 of file CoinMessageHandler.hpp.

9.41.3.21 `int CoinMessageHandler::numberStringFields () const [inline]`

Number of string fields already processed.

Incremented each time a field of type string is processed.

Definition at line 451 of file CoinMessageHandler.hpp.

9.41.3.22 `CoinOneMessage CoinMessageHandler::currentMessage () const [inline]`

Current message.

Definition at line 455 of file CoinMessageHandler.hpp.

9.41.3.23 `std::string CoinMessageHandler::currentSource () const [inline]`

Source of current message.

Definition at line 458 of file CoinMessageHandler.hpp.

9.41.3.24 `const char* CoinMessageHandler::messageBuffer () const [inline]`

Output buffer.

Definition at line 461 of file CoinMessageHandler.hpp.

9.41.3.25 `int CoinMessageHandler::highestNumber () const [inline]`

Highest message number (indicates any errors)

Definition at line 464 of file CoinMessageHandler.hpp.

9.41.3.26 `FILE* CoinMessageHandler::filePointer () const [inline]`

Get current file pointer.

Definition at line 467 of file CoinMessageHandler.hpp.

9.41.3.27 `void CoinMessageHandler::setFilePointer (FILE * fp) [inline]`

Set new file pointer.

Definition at line 470 of file CoinMessageHandler.hpp.

9.41.3.28 `CoinMessageHandler& CoinMessageHandler::message (int messageNumber, const CoinMessages & messages)`

Start a message.

Look up the specified message. A prefix will be generated if enabled. The message will be printed if the current log level is equal or greater than the log level of the message.

9.41.3.29 `CoinMessageHandler& CoinMessageHandler::message (int detail = -1)`

Start or continue a message.

With `detail = -1` (default), does nothing except return a reference to the handler. (I.e., `msghandler.message() << "foo"` is precisely equivalent to `msghandler << "foo"`.) If `msgDetail` is `>= 0`, it will be used as the detail level to determine whether the message should print (assuming class 0).

This can be used with any of the `<<` operators. One use is to start a message which will be constructed entirely from scratch. Another use is continuation of a message after code that interrupts the usual sequence of `<<` operators.

9.41.3.30 `CoinMessageHandler& CoinMessageHandler::message (int externalNumber, const char * source, const char * msg, char severity, int detail = -1)`

Print a complete message.

Generate a standard prefix and append `msg` 'as is'. This is intended as a transition mechanism. The standard prefix is generated (if enabled), and `msg` is appended. The message must be ended with a `CoinMessageEol` marker. Attempts to add content with `<<` will have no effect.

The default value of `detail` will not change printing status. If `detail` is `>= 0`, it will be used as the detail level to determine whether the message should print (assuming class 0).

9.41.3.31 `CoinMessageHandler& CoinMessageHandler::operator<< (int intvalue)`

Process an integer parameter value.

The default format code is 'd'.

9.41.3.32 `CoinMessageHandler& CoinMessageHandler::operator<< (double doublevalue)`

Process a double parameter value.

The default format code is 'd'.

9.41.3.33 `CoinMessageHandler& CoinMessageHandler::operator<< (const std::string & stringvalue)`

Process a STL string parameter value.

The default format code is 'g'.

9.41.3.34 CoinMessageHandler& CoinMessageHandler::operator<< (char *charvalue*)

Process a char parameter value.

The default format code is 's'.

9.41.3.35 CoinMessageHandler& CoinMessageHandler::operator<< (const char * *stringvalue*)

Process a C-style string parameter value.

The default format code is 'c'.

9.41.3.36 CoinMessageHandler& CoinMessageHandler::operator<< (CoinMessageMarker)

Process a marker.

The default format code is 's'.

9.41.3.37 int CoinMessageHandler::finish ()

Finish (and print) the message.

Equivalent to using the CoinMessageEol marker.

9.41.3.38 CoinMessageHandler& CoinMessageHandler::printing (bool *onOff*)

Enable or disable printing of an optional portion of a message.

Optional portions of a message are delimited by '%?' markers, and printing processes one %? marker. If `onOff` is true, the subsequent portion of the message (to the next %? marker or the end of the format string) will be printed. If `onOff` is false, printing is suppressed. Parameters must still be supplied, whether printing is suppressed or not. See the class documentation for an example.

9.41.4 Friends And Related Function Documentation**9.41.4.1 bool CoinMessageHandlerUnitTest () [friend]**

A function that tests the methods in the [CoinMessageHandler](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

9.41.5 Member Data Documentation**9.41.5.1 std::vector<double> CoinMessageHandler::doubleValue_ [protected]**

values in message

Definition at line 593 of file CoinMessageHandler.hpp.

9.41.5.2 std::vector<int> CoinMessageHandler::longValue_ [protected]

Definition at line 594 of file CoinMessageHandler.hpp.

9.41.5.3 std::vector<char> CoinMessageHandler::charValue_ [protected]

Definition at line 595 of file CoinMessageHandler.hpp.

9.41.5.4 `std::vector<std::string> CoinMessageHandler::stringValue_` [protected]

Definition at line 596 of file CoinMessageHandler.hpp.

9.41.5.5 `int CoinMessageHandler::logLevel_` [protected]

Log level.

Definition at line 598 of file CoinMessageHandler.hpp.

9.41.5.6 `int CoinMessageHandler::logLevels_[COIN_NUM_LOG]` [protected]

Log levels.

Definition at line 600 of file CoinMessageHandler.hpp.

9.41.5.7 `int CoinMessageHandler::prefix_` [protected]

Whether we want prefix (may get more subtle so is int)

Definition at line 602 of file CoinMessageHandler.hpp.

9.41.5.8 `CoinOneMessage CoinMessageHandler::currentMessage_` [protected]

Current message.

Definition at line 604 of file CoinMessageHandler.hpp.

9.41.5.9 `int CoinMessageHandler::internalNumber_` [protected]

Internal number for use with enums.

Definition at line 606 of file CoinMessageHandler.hpp.

9.41.5.10 `char* CoinMessageHandler::format_` [protected]

Format string for message (remainder)

Definition at line 608 of file CoinMessageHandler.hpp.

9.41.5.11 `char CoinMessageHandler::messageBuffer_[COIN_MESSAGE_HANDLER_MAX_BUFFER_SIZE]`
[protected]

Output buffer.

Definition at line 610 of file CoinMessageHandler.hpp.

9.41.5.12 `char* CoinMessageHandler::messageOut_` [protected]

Position in output buffer.

Definition at line 612 of file CoinMessageHandler.hpp.

9.41.5.13 `std::string CoinMessageHandler::source_` [protected]

Current source of message.

Definition at line 614 of file CoinMessageHandler.hpp.

9.41.5.14 `int CoinMessageHandler::printStatus_` [protected]

0 - Normal.

1 - Put in values, move along format, but don't print. 2 - A complete message was provided; nothing more to do but print when CoinMessageEol is processed. Any << operators are treated as noops. 3 - do nothing except look for CoinMessageEol (i.e., the message detail level was not sufficient to cause it to print).

Definition at line 623 of file CoinMessageHandler.hpp.

9.41.5.15 `int CoinMessageHandler::highestNumber_` `[protected]`

Highest message number (indicates any errors)

Definition at line 625 of file CoinMessageHandler.hpp.

9.41.5.16 `FILE* CoinMessageHandler::fp_` `[protected]`

File pointer.

Definition at line 627 of file CoinMessageHandler.hpp.

9.41.5.17 `char CoinMessageHandler::g_format_[8]` `[protected]`

Current format for floating point numbers.

Definition at line 629 of file CoinMessageHandler.hpp.

9.41.5.18 `int CoinMessageHandler::g_precision_` `[protected]`

Current number of significant digits for floating point numbers.

Definition at line 631 of file CoinMessageHandler.hpp.

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

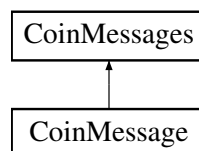
- [/home/ted/COIN/trunk/CoinUtils/src/CoinMessageHandler.hpp](#)

9.42 CoinMessages Class Reference

Class to hold and manipulate an array of massaged messages.

```
#include <CoinMessageHandler.hpp>
```

Inheritance diagram for CoinMessages:



Public Types

- enum `Language` { `us_en` = 0, `uk_en`, `it` }
- Supported languages.*

Public Member Functions

Constructors etc

- [CoinMessages](#) (int numberMessages=0)
Constructor with number of messages.
- [~CoinMessages](#) ()
Destructor.
- [CoinMessages](#) (const [CoinMessages](#) &)
The copy constructor.
- [CoinMessages](#) & [operator=](#) (const [CoinMessages](#) &)
assignment operator.

Useful stuff

- [void addMessage](#) (int messageNumber, const [CoinOneMessage](#) &message)
Installs a new message in the specified index position.
- [void replaceMessage](#) (int messageNumber, const char *message)
Replaces the text of the specified message.
- [Language language](#) () const
Language.
- [void setLanguage](#) ([Language](#) newlanguage)
Set language.
- [void setDetailMessage](#) (int newLevel, int messageNumber)
Change detail level for one message.
- [void setDetailMessages](#) (int newLevel, int numberMessages, int *messageNumbers)
Change detail level for several messages.
- [void setDetailMessages](#) (int newLevel, int low, int high)
Change detail level for all messages with low <= ID number < high.
- int [getClass](#) () const
Returns class.
- [void toCompact](#) ()
Moves to compact format.
- [void fromCompact](#) ()
Moves from compact format.

Public Attributes

member data

- int [numberMessages_](#)
Number of messages.
- [Language language_](#)
Language.
- char [source_](#) [5]
Source (null-terminated string, maximum 4 characters).
- int [class_](#)
Class - see later on before [CoinMessageHandler](#).
- int [lengthMessages_](#)
Length of fake [CoinOneMessage](#) array.
- [CoinOneMessage ** message_](#)
Messages.

9.42.1 Detailed Description

Class to hold and manipulate an array of massaged messages.

Note that the message index used to reference a message in the array of messages is completely distinct from the external ID number stored with the message.

Definition at line 128 of file [CoinMessageHandler.hpp](#).

9.42.2 Member Enumeration Documentation

9.42.2.1 enum CoinMessages::Language

Supported languages.

These are the languages that are supported. At present only `us_en` is serious and the rest are for testing.

Enumerator

us_en

uk_en

it

Definition at line 136 of file `CoinMessageHandler.hpp`.

9.42.3 Constructor & Destructor Documentation

9.42.3.1 CoinMessages::CoinMessages (int *numberOfMessages* = 0)

Constructor with number of messages.

9.42.3.2 CoinMessages::~CoinMessages ()

Destructor.

9.42.3.3 CoinMessages::CoinMessages (const CoinMessages &)

The copy constructor.

9.42.4 Member Function Documentation

9.42.4.1 CoinMessages& CoinMessages::operator= (const CoinMessages &)

assignment operator.

9.42.4.2 void CoinMessages::addMessage (int *messageNumber*, const CoinOneMessage & *message*)

Installs a new message in the specified index position.

Any existing message is replaced, and a copy of the specified message is installed.

9.42.4.3 void CoinMessages::replaceMessage (int *messageNumber*, const char * *message*)

Replaces the text of the specified message.

Any existing text is deleted and the specified text is copied into the specified message.

9.42.4.4 Language CoinMessages::language () const [inline]

Language.

Need to think about iso codes

Definition at line 169 of file `CoinMessageHandler.hpp`.

9.42.4.5 void CoinMessages::setLanguage (Language *newlanguage*) [inline]

Set language.

Definition at line 172 of file CoinMessageHandler.hpp.

9.42.4.6 void CoinMessages::setDetailMessage (int *newLevel*, int *messageNumber*)

Change detail level for one message.

9.42.4.7 void CoinMessages::setDetailMessages (int *newLevel*, int *numberMessages*, int * *messageNumbers*)

Change detail level for several messages.

messageNumbers is expected to contain the indices of the messages to be changed. If *numberMessages* \geq 10000 or *messageNumbers* is NULL, the detail level is changed on all messages.

9.42.4.8 void CoinMessages::setDetailMessages (int *newLevel*, int *low*, int *high*)

Change detail level for all messages with $low \leq$ ID number $<$ *high*.

9.42.4.9 int CoinMessages::getClass () const [inline]

Returns class.

Definition at line 189 of file CoinMessageHandler.hpp.

9.42.4.10 void CoinMessages::toCompact ()

Moves to compact format.

9.42.4.11 void CoinMessages::fromCompact ()

Moves from compact format.

9.42.5 Member Data Documentation

9.42.5.1 int CoinMessages::numberMessages_

Number of messages.

Definition at line 200 of file CoinMessageHandler.hpp.

9.42.5.2 Language CoinMessages::language_

Language.

Definition at line 202 of file CoinMessageHandler.hpp.

9.42.5.3 char CoinMessages::source_[5]

Source (null-terminated string, maximum 4 characters).

Definition at line 204 of file CoinMessageHandler.hpp.

9.42.5.4 int CoinMessages::class_

Class - see later on before [CoinMessageHandler](#).

Definition at line 206 of file CoinMessageHandler.hpp.

9.42.5.5 int CoinMessages::lengthMessages_

Length of fake [CoinOneMessage](#) array.

First you get numberMessages_ pointers which point to stuff

Definition at line 210 of file CoinMessageHandler.hpp.

9.42.5.6 CoinOneMessage** CoinMessages::message_

Messages.

Definition at line 212 of file CoinMessageHandler.hpp.

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

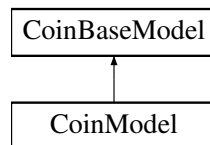
- [/home/ted/COIN/trunk/CoinUtils/src/CoinMessageHandler.hpp](#)

9.43 CoinModel Class Reference

This is a simple minded model which is stored in a format which makes it easier to construct and modify but not efficient for algorithms.

```
#include <CoinModel.hpp>
```

Inheritance diagram for CoinModel:



Public Member Functions

- int [computeAssociated](#) (double *associated)
Fills in all associated - returning number of errors.
- [CoinPackedMatrix](#) * [quadraticRow](#) (int rowNumber, double *linear, int &numberBad) const
Gets correct form for a quadratic row - user to delete If row is not quadratic then returns which other variables are involved with tiny (1.0e-100) elements and count of total number of variables which could not be put in quadratic form.
- void [replaceQuadraticRow](#) (int rowNumber, const double *linear, const [CoinPackedMatrix](#) *quadraticPart)
Replaces a quadratic row.
- [CoinModel](#) * [reorder](#) (const char *mark) const
If possible return a model where if all variables marked nonzero are fixed the problem will be linear.
- int [expandKnapsack](#) (int knapsackRow, int &numberOutput, double *buildObj, [CoinBigIndex](#) *buildStart, int *buildRow, double *buildElement, int reConstruct=-1) const
Expands out all possible combinations for a knapsack If buildObj NULL then just computes space needed - returns number elements On entry numberOutput is maximum allowed, on exit it is number needed or -1 (as will be number elements) if maximum exceeded.
- void [setCutMarker](#) (int size, const int *marker)
Sets cut marker array.
- void [setPriorities](#) (int size, const int *priorities)
Sets priority array.

- `const int * priorities () const`
priorities (given for all columns (-1 if not integer))
- `void setOriginalIndices (const int *row, const int *column)`
For decomposition set original row and column indices.

Useful methods for building model

- `void addRow (int numberInRow, const int *columns, const double *elements, double rowLower=-COIN_DBL_MAX, double rowUpper=COIN_DBL_MAX, const char *name=NULL)`
add a row - numberInRow may be zero
- `void addColumn (int numberInColumn, const int *rows, const double *elements, double columnLower=0.0, double columnUpper=COIN_DBL_MAX, double objectiveValue=0.0, const char *name=NULL, bool isInteger=false)`
*add a column - numberInColumn may be zero */*
- `void addCol (int numberInColumn, const int *rows, const double *elements, double columnLower=0.0, double columnUpper=COIN_DBL_MAX, double objectiveValue=0.0, const char *name=NULL, bool isInteger=false)`
*add a column - numberInColumn may be zero */*
- `void operator() (int i, int j, double value)`
Sets value for row i and column j.
- `void setElement (int i, int j, double value)`
Sets value for row i and column j.
- `int getRow (int whichRow, int *column, double *element)`
Gets sorted row - user must provide enough space (easiest is allocate number of columns).
- `int getColumn (int whichColumn, int *column, double *element)`
Gets sorted column - user must provide enough space (easiest is allocate number of rows).
- `void setQuadraticElement (int i, int j, double value)`
Sets quadratic value for column i and j.
- `void operator() (int i, int j, const char *value)`
Sets value for row i and column j as string.
- `void setElement (int i, int j, const char *value)`
Sets value for row i and column j as string.
- `int associateElement (const char *stringValue, double value)`
Associates a string with a value. Returns string id (or -1 if does not exist)
- `void setRowLower (int whichRow, double rowLower)`
Sets rowLower (if row does not exist then all rows up to this are defined with default values and no elements)
- `void setRowUpper (int whichRow, double rowUpper)`
Sets rowUpper (if row does not exist then all rows up to this are defined with default values and no elements)
- `void setRowBounds (int whichRow, double rowLower, double rowUpper)`
Sets rowLower and rowUpper (if row does not exist then all rows up to this are defined with default values and no elements)
- `void setRowName (int whichRow, const char *rowName)`
Sets name (if row does not exist then all rows up to this are defined with default values and no elements)
- `void setColumnLower (int whichColumn, double columnLower)`
Sets columnLower (if column does not exist then all columns up to this are defined with default values and no elements)
- `void setColumnUpper (int whichColumn, double columnUpper)`
Sets columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)
- `void setColumnBounds (int whichColumn, double columnLower, double columnUpper)`
Sets columnLower and columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)
- `void setColumnObjective (int whichColumn, double columnObjective)`
Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)
- `void setColumnName (int whichColumn, const char *columnName)`
Sets name (if column does not exist then all columns up to this are defined with default values and no elements)

- **void setColumnsInteger** (int whichColumn, bool columnsInteger)
Sets integer state (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setObjective** (int whichColumn, double columnObjective)
Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setIsInteger** (int whichColumn, bool columnsInteger)
Sets integer state (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setInteger** (int whichColumn)
Sets integer (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setContinuous** (int whichColumn)
Sets continuous (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColLower** (int whichColumn, double columnLower)
Sets columnLower (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColUpper** (int whichColumn, double columnUpper)
Sets columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColBounds** (int whichColumn, double columnLower, double columnUpper)
Sets columnLower and columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColObjective** (int whichColumn, double columnObjective)
Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColName** (int whichColumn, const char *columnName)
Sets name (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColsInteger** (int whichColumn, bool columnsInteger)
Sets integer (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setRowLower** (int whichRow, const char *rowLower)
Sets rowLower (if row does not exist then all rows up to this are defined with default values and no elements)
- **void setRowUpper** (int whichRow, const char *rowUpper)
Sets rowUpper (if row does not exist then all rows up to this are defined with default values and no elements)
- **void setColumnLower** (int whichColumn, const char *columnLower)
Sets columnLower (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColumnUpper** (int whichColumn, const char *columnUpper)
Sets columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColumnObjective** (int whichColumn, const char *columnObjective)
Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setColumnsInteger** (int whichColumn, const char *columnsInteger)
Sets integer (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setObjective** (int whichColumn, const char *columnObjective)
Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)
- **void setIsInteger** (int whichColumn, const char *columnsInteger)
Sets integer (if column does not exist then all columns up to this are defined with default values and no elements)
- **void deleteRow** (int whichRow)
Deletes all entries in row and bounds.
- **void deleteColumn** (int whichColumn)
Deletes all entries in column and bounds and objective.
- **void deleteCol** (int whichColumn)
Deletes all entries in column and bounds.
- **int deleteElement** (int row, int column)
Takes element out of matrix - returning position (<0 if not there);.
- **void deleteThisElement** (int row, int column, int position)
Takes element out of matrix when position known.
- **int packRows** ()
Packs down all rows i.e.

- int `packColumns` ()
Packs down all columns i.e.
- int `packCols` ()
Packs down all columns i.e.
- int `pack` ()
Packs down all rows and columns.
- void `setObjective` (int `numberColumns`, const double *`objective`)
Sets columnObjective array.
- void `setColumnLower` (int `numberColumns`, const double *`columnLower`)
Sets columnLower array.
- void `setColLower` (int `numberColumns`, const double *`columnLower`)
Sets columnLower array.
- void `setColumnUpper` (int `numberColumns`, const double *`columnUpper`)
Sets columnUpper array.
- void `setColUpper` (int `numberColumns`, const double *`columnUpper`)
Sets columnUpper array.
- void `setRowLower` (int `numberRows`, const double *`rowLower`)
Sets rowLower array.
- void `setRowUpper` (int `numberRows`, const double *`rowUpper`)
Sets rowUpper array.
- int `writeMps` (const char *`filename`, int `compression`=0, int `formatType`=0, int `numberAcross`=2, bool `keepStrings`=false)
Write the problem in MPS format to a file with the given filename.
- int `differentModel` (`CoinModel` &`other`, bool `ignoreNames`)
Check two models against each other.

For structured models

- void `passInMatrix` (const `CoinPackedMatrix` &`matrix`)
Pass in `CoinPackedMatrix` (and switch off element updates)
- int `convertMatrix` ()
Convert elements to `CoinPackedMatrix` (and switch off element updates).
- const `CoinPackedMatrix` * `packedMatrix` () const
Return a pointer to `CoinPackedMatrix` (or NULL)
- const int * `originalRows` () const
Return pointers to original rows (for decomposition)
- const int * `originalColumns` () const
Return pointers to original columns (for decomposition)

For getting information

- `CoinBigIndex` `numberElements` () const
Return number of elements.
- const `CoinModelTriple` * `elements` () const
Return elements as triples.
- double `operator()` (int `i`, int `j`) const
Returns value for row `i` and column `j`.
- double `getElement` (int `i`, int `j`) const
Returns value for row `i` and column `j`.
- double `operator()` (const char *`rowName`, const char *`columnName`) const
Returns value for row `rowName` and column `columnName`.
- double `getElement` (const char *`rowName`, const char *`columnName`) const
Returns value for row `rowName` and column `columnName`.
- double `getQuadraticElement` (int `i`, int `j`) const

- Returns quadratic value for columns i and j.*
- const char * [getElementAsString](#) (int i, int j) const
Returns value for row i and column j as string.
- double * [pointer](#) (int i, int j) const
Returns pointer to element for row i column j.
- int [position](#) (int i, int j) const
Returns position in elements for row i column j.
- [CoinModelLink firstInRow](#) (int whichRow) const
Returns first element in given row - index is -1 if none.
- [CoinModelLink lastInRow](#) (int whichRow) const
Returns last element in given row - index is -1 if none.
- [CoinModelLink firstInColumn](#) (int whichColumn) const
Returns first element in given column - index is -1 if none.
- [CoinModelLink lastInColumn](#) (int whichColumn) const
Returns last element in given column - index is -1 if none.
- [CoinModelLink next](#) ([CoinModelLink](#) ¤t) const
Returns next element in current row or column - index is -1 if none.
- [CoinModelLink previous](#) ([CoinModelLink](#) ¤t) const
Returns previous element in current row or column - index is -1 if none.
- [CoinModelLink firstInQuadraticColumn](#) (int whichColumn) const
Returns first element in given quadratic column - index is -1 if none.
- [CoinModelLink lastInQuadraticColumn](#) (int whichColumn) const
Returns last element in given quadratic column - index is -1 if none.
- double [getRowLower](#) (int whichRow) const
Gets rowLower (if row does not exist then -COIN_DBL_MAX)
- double [getRowUpper](#) (int whichRow) const
Gets rowUpper (if row does not exist then +COIN_DBL_MAX)
- const char * [getRowName](#) (int whichRow) const
Gets name (if row does not exist then NULL)
- double [rowLower](#) (int whichRow) const
- double [rowUpper](#) (int whichRow) const
Gets rowUpper (if row does not exist then COIN_DBL_MAX)
- const char * [rowName](#) (int whichRow) const
Gets name (if row does not exist then NULL)
- double [getColumnLower](#) (int whichColumn) const
Gets columnLower (if column does not exist then 0.0)
- double [getColumnUpper](#) (int whichColumn) const
Gets columnUpper (if column does not exist then COIN_DBL_MAX)
- double [getColumnObjective](#) (int whichColumn) const
Gets columnObjective (if column does not exist then 0.0)
- const char * [getColumnName](#) (int whichColumn) const
Gets name (if column does not exist then NULL)
- bool [getColumnIsInteger](#) (int whichColumn) const
Gets if integer (if column does not exist then false)
- double [columnLower](#) (int whichColumn) const
Gets columnLower (if column does not exist then 0.0)
- double [columnUpper](#) (int whichColumn) const
Gets columnUpper (if column does not exist then COIN_DBL_MAX)
- double [columnObjective](#) (int whichColumn) const
Gets columnObjective (if column does not exist then 0.0)
- double [objective](#) (int whichColumn) const
Gets columnObjective (if column does not exist then 0.0)
- const char * [columnName](#) (int whichColumn) const
Gets name (if column does not exist then NULL)
- bool [columnIsInteger](#) (int whichColumn) const

- Gets if integer (if column does not exist then false)*
- bool `isInteger` (int whichColumn) const
- Gets if integer (if column does not exist then false)*
- double `getColLower` (int whichColumn) const
- Gets columnLower (if column does not exist then 0.0)*
- double `getColUpper` (int whichColumn) const
- Gets columnUpper (if column does not exist then COIN_DBL_MAX)*
- double `getColObjective` (int whichColumn) const
- Gets columnObjective (if column does not exist then 0.0)*
- const char * `getColName` (int whichColumn) const
- Gets name (if column does not exist then NULL)*
- bool `getColsInteger` (int whichColumn) const
- Gets if integer (if column does not exist then false)*
- const char * `getRowLowerAsString` (int whichRow) const
- Gets rowLower (if row does not exist then -COIN_DBL_MAX)*
- const char * `getRowUpperAsString` (int whichRow) const
- Gets rowUpper (if row does not exist then +COIN_DBL_MAX)*
- const char * `rowLowerAsString` (int whichRow) const
- const char * `rowUpperAsString` (int whichRow) const
- Gets rowUpper (if row does not exist then COIN_DBL_MAX)*
- const char * `getColumnLowerAsString` (int whichColumn) const
- Gets columnLower (if column does not exist then 0.0)*
- const char * `getColumnUpperAsString` (int whichColumn) const
- Gets columnUpper (if column does not exist then COIN_DBL_MAX)*
- const char * `getColumnObjectiveAsString` (int whichColumn) const
- Gets columnObjective (if column does not exist then 0.0)*
- const char * `getColumnIsIntegerAsString` (int whichColumn) const
- Gets if integer (if column does not exist then false)*
- const char * `columnLowerAsString` (int whichColumn) const
- Gets columnLower (if column does not exist then 0.0)*
- const char * `columnUpperAsString` (int whichColumn) const
- Gets columnUpper (if column does not exist then COIN_DBL_MAX)*
- const char * `columnObjectiveAsString` (int whichColumn) const
- Gets columnObjective (if column does not exist then 0.0)*
- const char * `objectiveAsString` (int whichColumn) const
- Gets columnObjective (if column does not exist then 0.0)*
- const char * `columnIsIntegerAsString` (int whichColumn) const
- Gets if integer (if column does not exist then false)*
- const char * `isIntegerAsString` (int whichColumn) const
- Gets if integer (if column does not exist then false)*
- int `row` (const char *rowName) const
- Row index from row name (-1 if no names or no match)*
- int `column` (const char *columnName) const
- Column index from column name (-1 if no names or no match)*
- int `type` () const
- Returns type.*
- double `unsetValue` () const
- returns unset value*
- int `createPackedMatrix` (CoinPackedMatrix &matrix, const double *associated)
- Creates a packed matrix - return number of errors.*
- int `countPlusMinusOne` (CoinBigIndex *startPositive, CoinBigIndex *startNegative, const double *associated)
- Fills in startPositive and startNegative with counts for +-1 matrix.*
- void `createPlusMinusOne` (CoinBigIndex *startPositive, CoinBigIndex *startNegative, int *indices, const double *associated)
- Creates +-1 matrix given startPositive and startNegative counts for +-1 matrix.*

- int `createArrays` (double *&rowLower, double *&rowUpper, double *&columnLower, double *&columnUpper, double *&objective, int *&integerType, double *&associated)
Creates copies of various arrays - return number of errors.
- bool `stringsExist` () const
Says if strings exist.
- const `CoinModelHash` * `stringArray` () const
Return string array.
- double * `associatedArray` () const
Returns associated array.
- double * `rowLowerArray` () const
Return rowLower array.
- double * `rowUpperArray` () const
Return rowUpper array.
- double * `columnLowerArray` () const
Return columnLower array.
- double * `columnUpperArray` () const
Return columnUpper array.
- double * `objectiveArray` () const
Return objective array.
- int * `integerTypeArray` () const
Return integerType array.
- const `CoinModelHash` * `rowNames` () const
Return row names array.
- const `CoinModelHash` * `columnNames` () const
Return column names array.
- void `zapRowNames` ()
Reset row names.
- void `zapColumnNames` ()
Reset column names.
- const int * `cutMarker` () const
Returns array of 0 or nonzero if can be a cut (or returns NULL)
- double `optimizationDirection` () const
Direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.
- void `setOptimizationDirection` (double value)
Set direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.
- void * `moreInfo` () const
Return pointer to more information.
- void `setMoreInfo` (void *info)
Set pointer to more information.
- int `whatIsSet` () const
Returns which parts of model are set 1 - matrix 2 - rhs 4 - row names 8 - column bounds and/or objective 16 - column names 32 - integer types.

for block models - matrix will be `CoinPackedMatrix`

- void `loadBlock` (const `CoinPackedMatrix` &matrix, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub)
Load in a problem by copying the arguments.
- void `loadBlock` (const `CoinPackedMatrix` &matrix, const double *collb, const double *colub, const double *obj, const char *rowlen, const double *rowrhs, const double *rowrng)
Load in a problem by copying the arguments.
- void `loadBlock` (const int numcols, const int numRows, const `CoinBigIndex` *start, const int *index, const double *value, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub)
Load in a problem by copying the arguments.

- `void loadBlock` (const int numcols, const int numRows, const `CoinBigIndex` *start, const int *index, const double *value, const double *collb, const double *colub, const double *obj, const char *rowns, const double *rowrhs, const double *rowrng)

Load in a problem by copying the arguments.

Constructors, destructor

- `CoinModel` ()
Default constructor.
- `CoinModel` (int firstRows, int firstColumns, int firstElements, bool noNames=false)
Constructor with sizes.
- `CoinModel` (const char *fileName, int allowStrings=0)
Read a problem in MPS or GAMS format from the given filename.
- `CoinModel` (int nonLinear, const char *fileName, const `void` *info)
Read a problem from AMPL nl file NOTE - as I can't work out configure etc the source code is in Cbc_ampl.cpp!
- `CoinModel` (int `numberRows`, int `numberColumns`, const `CoinPackedMatrix` *matrix, const double *rowLower, const double *rowUpper, const double *columnLower, const double *columnUpper, const double *objective)
From arrays.
- virtual `CoinBaseModel` * `clone` () const
Clone.
- virtual `~CoinModel` ()
Destructor.

Copy method

- `CoinModel` (const `CoinModel` &)
The copy constructor.
- `CoinModel` & `operator=` (const `CoinModel` &)
=

For debug

- `void validateLinks` () const
Checks that links are consistent.

Additional Inherited Members

9.43.1 Detailed Description

This is a simple minded model which is stored in a format which makes it easier to construct and modify but not efficient for algorithms.

It has to be passed across to `ClpModel` or `OsiSolverInterface` by `addRows`, `addCol(umn)s` or `loadProblem`.

It may have up to four parts - 1) A matrix of doubles (or strings - see note A) 2) Column information including integer information and names 3) Row information including names 4) Quadratic objective (not implemented - but see A)

This class is meant to make it more efficient to build a model. It is at its most efficient when all additions are done as `addRow` or as `addCol` but not mixed. If only 1 and 2 exist then `solver.addColumns` may be used to pass to solver, if only 1 and 3 exist then `solver.addRows` may be used. Otherwise `solver.loadProblem` must be used.

If `addRows` and `addColumns` are mixed or if individual elements are set then the speed will drop to some extent and more memory will be used.

It is also possible to iterate over existing elements and to access columns and rows by name. Again each of these use memory and cpu time. However memory is unlikely to be critical as most algorithms will use much more.

Notes: A) Although this could be used to pass nonlinear information around the only use at present is to have named values e.g. value1 which can then be set to a value after model is created. I have no idea whether that could be useful but I thought it might be fun. Quadratic terms are allowed in strings! A solver could try and use this if so - the convention is that 0.5* quadratic is stored

B) This class could be useful for modeling.

Definition at line 152 of file CoinModel.hpp.

9.43.2 Constructor & Destructor Documentation

9.43.2.1 CoinModel::CoinModel ()

Default constructor.

9.43.2.2 CoinModel::CoinModel (int *firstRows*, int *firstColumns*, int *firstElements*, bool *noNames* = false)

Constructor with sizes.

9.43.2.3 CoinModel::CoinModel (const char * *fileName*, int *allowStrings* = 0)

Read a problem in MPS or GAMS format from the given filename.

9.43.2.4 CoinModel::CoinModel (int *nonLinear*, const char * *fileName*, const void * *info*)

Read a problem from AMPL nl file NOTE - as I can't work out configure etc the source code is in Cbc_ampl.cpp!

9.43.2.5 CoinModel::CoinModel (int *numberRows*, int *numberColumns*, const CoinPackedMatrix * *matrix*, const double * *rowLower*, const double * *rowUpper*, const double * *columnLower*, const double * *columnUpper*, const double * *objective*)

From arrays.

9.43.2.6 virtual CoinModel::~CoinModel () [virtual]

Destructor.

9.43.2.7 CoinModel::CoinModel (const CoinModel &)

The copy constructor.

9.43.3 Member Function Documentation

9.43.3.1 void CoinModel::addRow (int *numberInRow*, const int * *columns*, const double * *elements*, double *rowLower* = -COIN_DBL_MAX, double *rowUpper* = COIN_DBL_MAX, const char * *name* = NULL)

add a row - numberInRow may be zero

9.43.3.2 void CoinModel::addColumn (int *numberInColumn*, const int * *rows*, const double * *elements*, double *columnLower* = 0.0, double *columnUpper* = COIN_DBL_MAX, double *objectiveValue* = 0.0, const char * *name* = NULL, bool *isInteger* = false)

add a column - numberInColumn may be zero */

9.43.3.3 void CoinModel::addCol (int *numberInColumn*, const int * *rows*, const double * *elements*, double *columnLower* = 0.0, double *columnUpper* = COIN_DBL_MAX, double *objectiveValue* = 0.0, const char * *name* = NULL, bool *isInteger* = false) [inline]

add a column - numberInColumn may be zero */

Definition at line 168 of file CoinModel.hpp.

9.43.3.4 void CoinModel::operator() (int *i*, int *j*, double *value*) [inline]

Sets value for row *i* and column *j*.

Definition at line 176 of file CoinModel.hpp.

9.43.3.5 void CoinModel::setElement (int *i*, int *j*, double *value*)

Sets value for row *i* and column *j*.

9.43.3.6 int CoinModel::getRow (int *whichRow*, int * *column*, double * *element*)

Gets sorted row - user must provide enough space (easiest is allocate number of columns).

If column or element NULL then just returns number Returns number of elements

9.43.3.7 int CoinModel::getColumn (int *whichColumn*, int * *column*, double * *element*)

Gets sorted column - user must provide enough space (easiest is allocate number of rows).

If row or element NULL then just returns number Returns number of elements

9.43.3.8 void CoinModel::setQuadraticElement (int *i*, int *j*, double *value*)

Sets quadratic value for column *i* and *j*.

9.43.3.9 void CoinModel::operator() (int *i*, int *j*, const char * *value*) [inline]

Sets value for row *i* and column *j* as string.

Definition at line 195 of file CoinModel.hpp.

9.43.3.10 void CoinModel::setElement (int *i*, int *j*, const char * *value*)

Sets value for row *i* and column *j* as string.

9.43.3.11 int CoinModel::associateElement (const char * *stringValue*, double *value*)

Associates a string with a value. Returns string id (or -1 if does not exist)

9.43.3.12 void CoinModel::setRowLower (int *whichRow*, double *rowLower*)

Sets rowLower (if row does not exist then all rows up to this are defined with default values and no elements)

9.43.3.13 void CoinModel::setRowUpper (int *whichRow*, double *rowUpper*)

Sets rowUpper (if row does not exist then all rows up to this are defined with default values and no elements)

9.43.3.14 void CoinModel::setRowBounds (int *whichRow*, double *rowLower*, double *rowUpper*)

Sets rowLower and rowUpper (if row does not exist then all rows up to this are defined with default values and no elements)

9.43.3.15 `void CoinModel::setRowName (int whichRow, const char * rowName)`

Sets name (if row does not exist then all rows up to this are defined with default values and no elements)

9.43.3.16 `void CoinModel::setColumnLower (int whichColumn, double columnLower)`

Sets columnLower (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.17 `void CoinModel::setColumnUpper (int whichColumn, double columnUpper)`

Sets columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.18 `void CoinModel::setColumnBounds (int whichColumn, double columnLower, double columnUpper)`

Sets columnLower and columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.19 `void CoinModel::setColumnObjective (int whichColumn, double columnObjective)`

Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.20 `void CoinModel::setColumnName (int whichColumn, const char * columnName)`

Sets name (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.21 `void CoinModel::setColumnsInteger (int whichColumn, bool columnsInteger)`

Sets integer state (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.22 `void CoinModel::setObjective (int whichColumn, double columnObjective)` `[inline]`

Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 244 of file CoinModel.hpp.

9.43.3.23 `void CoinModel::setIsInteger (int whichColumn, bool columnsInteger)` `[inline]`

Sets integer state (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 249 of file CoinModel.hpp.

9.43.3.24 `void CoinModel::setInteger (int whichColumn)` `[inline]`

Sets integer (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 254 of file CoinModel.hpp.

9.43.3.25 `void CoinModel::setContinuous (int whichColumn)` `[inline]`

Sets continuous (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 259 of file CoinModel.hpp.

9.43.3.26 `void CoinModel::setColLower (int whichColumn, double columnLower)` `[inline]`

Sets columnLower (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 264 of file CoinModel.hpp.

9.43.3.27 void CoinModel::setColUpper (int *whichColumn*, double *columnUpper*) [inline]

Sets columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 269 of file CoinModel.hpp.

9.43.3.28 void CoinModel::setColBounds (int *whichColumn*, double *columnLower*, double *columnUpper*) [inline]

Sets columnLower and columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 274 of file CoinModel.hpp.

9.43.3.29 void CoinModel::setColObjective (int *whichColumn*, double *columnObjective*) [inline]

Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 279 of file CoinModel.hpp.

9.43.3.30 void CoinModel::setColName (int *whichColumn*, const char * *columnName*) [inline]

Sets name (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 284 of file CoinModel.hpp.

9.43.3.31 void CoinModel::setColsInteger (int *whichColumn*, bool *columnsInteger*) [inline]

Sets integer (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 289 of file CoinModel.hpp.

9.43.3.32 void CoinModel::setRowLower (int *whichRow*, const char * *rowLower*)

Sets rowLower (if row does not exist then all rows up to this are defined with default values and no elements)

9.43.3.33 void CoinModel::setRowUpper (int *whichRow*, const char * *rowUpper*)

Sets rowUpper (if row does not exist then all rows up to this are defined with default values and no elements)

9.43.3.34 void CoinModel::setColumnLower (int *whichColumn*, const char * *columnLower*)

Sets columnLower (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.35 void CoinModel::setColumnUpper (int *whichColumn*, const char * *columnUpper*)

Sets columnUpper (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.36 void CoinModel::setColumnObjective (int *whichColumn*, const char * *columnObjective*)

Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.37 void CoinModel::setColumnsInteger (int *whichColumn*, const char * *columnsInteger*)

Sets integer (if column does not exist then all columns up to this are defined with default values and no elements)

9.43.3.38 `void CoinModel::setObjective (int whichColumn, const char * columnObjective)` `[inline]`

Sets columnObjective (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 318 of file CoinModel.hpp.

9.43.3.39 `void CoinModel::setIsInteger (int whichColumn, const char * columnIsInteger)` `[inline]`

Sets integer (if column does not exist then all columns up to this are defined with default values and no elements)

Definition at line 323 of file CoinModel.hpp.

9.43.3.40 `void CoinModel::deleteRow (int whichRow)`

Deletes all entries in row and bounds.

Will be ignored by writeMps etc and will be packed down if asked for.

9.43.3.41 `void CoinModel::deleteColumn (int whichColumn)`

Deletes all entries in column and bounds and objective.

Will be ignored by writeMps etc and will be packed down if asked for.

9.43.3.42 `void CoinModel::deleteCol (int whichColumn)` `[inline]`

Deletes all entries in column and bounds.

If last column the number of columns will be decremented and true returned.

Definition at line 333 of file CoinModel.hpp.

9.43.3.43 `int CoinModel::deleteElement (int row, int column)`

Takes element out of matrix - returning position (<0 if not there);.

9.43.3.44 `void CoinModel::deleteThisElement (int row, int column, int position)`

Takes element out of matrix when position known.

9.43.3.45 `int CoinModel::packRows ()`

Packs down all rows i.e.

removes empty rows permanently. Empty rows have no elements and feasible bounds. returns number of rows deleted.

9.43.3.46 `int CoinModel::packColumns ()`

Packs down all columns i.e.

removes empty columns permanently. Empty columns have no elements and no objective. returns number of columns deleted.

9.43.3.47 `int CoinModel::packCols ()` `[inline]`

Packs down all columns i.e.

removes empty columns permanently. Empty columns have no elements and no objective. returns number of columns deleted.

Definition at line 347 of file CoinModel.hpp.

9.43.3.48 `int CoinModel::pack ()`

Packs down all rows and columns.

i.e. removes empty rows and columns permanently. Empty rows have no elements and feasible bounds. Empty columns have no elements and no objective. returns number of rows+columns deleted.

9.43.3.49 `void CoinModel::setObjective (int numberColumns, const double * objective)`

Sets columnObjective array.

9.43.3.50 `void CoinModel::setColumnLower (int numberColumns, const double * columnLower)`

Sets columnLower array.

9.43.3.51 `void CoinModel::setColLower (int numberColumns, const double * columnLower)` `[inline]`

Sets columnLower array.

Definition at line 363 of file CoinModel.hpp.

9.43.3.52 `void CoinModel::setColumnUpper (int numberColumns, const double * columnUpper)`

Sets columnUpper array.

9.43.3.53 `void CoinModel::setColUpper (int numberColumns, const double * columnUpper)` `[inline]`

Sets columnUpper array.

Definition at line 370 of file CoinModel.hpp.

9.43.3.54 `void CoinModel::setRowLower (int numberRows, const double * rowLower)`

Sets rowLower array.

9.43.3.55 `void CoinModel::setRowUpper (int numberRows, const double * rowUpper)`

Sets rowUpper array.

9.43.3.56 `int CoinModel::writeMps (const char * filename, int compression = 0, int formatType = 0, int numberAcross = 2, bool keepStrings = false)`

Write the problem in MPS format to a file with the given filename.

Parameters

<i>compression</i>	<p>can be set to three values to indicate what kind of file should be written</p> <ul style="list-style-type: none"> • 0: plain text (default) • 1: gzip compressed (.gz is appended to <code>filename</code>) • 2: bzip2 compressed (.bz2 is appended to <code>filename</code>) (TODO) <p>If the library was not compiled with the requested compression then <code>writeMps</code> falls back to writing a plain text file.</p>
<i>formatType</i>	<p>specifies the precision to used for values in the MPS file</p> <ul style="list-style-type: none"> • 0: normal precision (default) • 1: extra accuracy • 2: IEEE hex
<i>numberAcross</i>	<p>specifies whether 1 or 2 (default) values should be specified on every data line in the MPS file.</p>

not const as may change model e.g. fill in default bounds

9.43.3.57 `int CoinModel::differentModel (CoinModel & other, bool ignoreNames)`

Check two models against each other.

Return nonzero if different. Ignore names if that set. May modify both models by cleaning up

9.43.3.58 `void CoinModel::passInMatrix (const CoinPackedMatrix & matrix)`

Pass in [CoinPackedMatrix](#) (and switch off element updates)

9.43.3.59 `int CoinModel::convertMatrix ()`

Convert elements to [CoinPackedMatrix](#) (and switch off element updates).

Returns number of errors

9.43.3.60 `const CoinPackedMatrix* CoinModel::packedMatrix () const [inline]`

Return a pointer to [CoinPackedMatrix](#) (or NULL)

Definition at line 423 of file `CoinModel.hpp`.

9.43.3.61 `const int* CoinModel::originalRows () const [inline]`

Return pointers to original rows (for decomposition)

Definition at line 426 of file `CoinModel.hpp`.

9.43.3.62 `const int* CoinModel::originalColumns () const [inline]`

Return pointers to original columns (for decomposition)

Definition at line 429 of file `CoinModel.hpp`.

9.43.3.63 `CoinBigIndex CoinModel::numberElements () const [inline],[virtual]`

Return number of elements.

Implements [CoinBaseModel](#).

Definition at line 437 of file CoinModel.hpp.

9.43.3.64 `const CoinModelTriple* CoinModel::elements () const` `[inline]`

Return elements as triples.

Definition at line 440 of file CoinModel.hpp.

9.43.3.65 `double CoinModel::operator() (int i, int j) const` `[inline]`

Returns value for row *i* and column *j*.

Definition at line 443 of file CoinModel.hpp.

9.43.3.66 `double CoinModel::getElement (int i, int j) const`

Returns value for row *i* and column *j*.

9.43.3.67 `double CoinModel::operator() (const char * rowName, const char * columnName) const` `[inline]`

Returns value for row *rowName* and column *columnName*.

Definition at line 448 of file CoinModel.hpp.

9.43.3.68 `double CoinModel::getElement (const char * rowName, const char * columnName) const`

Returns value for row *rowName* and column *columnName*.

9.43.3.69 `double CoinModel::getQuadraticElement (int i, int j) const`

Returns quadratic value for columns *i* and *j*.

9.43.3.70 `const char* CoinModel::getElementAsString (int i, int j) const`

Returns value for row *i* and column *j* as string.

Returns NULL if does not exist. Returns "Numeric" if not a string

9.43.3.71 `double* CoinModel::pointer (int i, int j) const`

Returns pointer to element for row *i* column *j*.

Only valid until next modification. NULL if element does not exist

9.43.3.72 `int CoinModel::position (int i, int j) const`

Returns position in elements for row *i* column *j*.

Only valid until next modification. -1 if element does not exist

9.43.3.73 `CoinModelLink CoinModel::firstInRow (int whichRow) const`

Returns first element in given row - index is -1 if none.

Index is given by `.index` and value by `.value`

9.43.3.74 `CoinModelLink CoinModel::lastInRow (int whichRow) const`

Returns last element in given row - index is -1 if none.

Index is given by `.index` and value by `.value`

9.43.3.75 CoinModelLink CoinModel::firstInColumn (int *whichColumn*) const

Returns first element in given column - index is -1 if none.

Index is given by .index and value by .value

9.43.3.76 CoinModelLink CoinModel::lastInColumn (int *whichColumn*) const

Returns last element in given column - index is -1 if none.

Index is given by .index and value by .value

9.43.3.77 CoinModelLink CoinModel::next (CoinModelLink & *current*) const

Returns next element in current row or column - index is -1 if none.

Index is given by .index and value by .value. User could also tell because input.next would be NULL

9.43.3.78 CoinModelLink CoinModel::previous (CoinModelLink & *current*) const

Returns previous element in current row or column - index is -1 if none.

Index is given by .index and value by .value. User could also tell because input.previous would be NULL May not be correct if matrix updated.

9.43.3.79 CoinModelLink CoinModel::firstInQuadraticColumn (int *whichColumn*) const

Returns first element in given quadratic column - index is -1 if none.

Index is given by .index and value by .value May not be correct if matrix updated.

9.43.3.80 CoinModelLink CoinModel::lastInQuadraticColumn (int *whichColumn*) const

Returns last element in given quadratic column - index is -1 if none.

Index is given by .index and value by .value

9.43.3.81 double CoinModel::getRowLower (int *whichRow*) const

Gets rowLower (if row does not exist then -COIN_DBL_MAX)

9.43.3.82 double CoinModel::getRowUpper (int *whichRow*) const

Gets rowUpper (if row does not exist then +COIN_DBL_MAX)

9.43.3.83 const char* CoinModel::getRowName (int *whichRow*) const

Gets name (if row does not exist then NULL)

9.43.3.84 double CoinModel::rowLower (int *whichRow*) const [inline]

Definition at line 514 of file CoinModel.hpp.

9.43.3.85 double CoinModel::rowUpper (int *whichRow*) const [inline]

Gets rowUpper (if row does not exist then COIN_DBL_MAX)

Definition at line 518 of file CoinModel.hpp.

9.43.3.86 `const char* CoinModel::rowName (int whichRow) const` `[inline]`

Gets name (if row does not exist then NULL)

Definition at line 522 of file CoinModel.hpp.

9.43.3.87 `double CoinModel::getColumnLower (int whichColumn) const`

Gets columnLower (if column does not exist then 0.0)

9.43.3.88 `double CoinModel::getColumnUpper (int whichColumn) const`

Gets columnUpper (if column does not exist then COIN_DBL_MAX)

9.43.3.89 `double CoinModel::getColumnObjective (int whichColumn) const`

Gets columnObjective (if column does not exist then 0.0)

9.43.3.90 `const char* CoinModel::getColumnName (int whichColumn) const`

Gets name (if column does not exist then NULL)

9.43.3.91 `bool CoinModel::getColumnIsInteger (int whichColumn) const`

Gets if integer (if column does not exist then false)

9.43.3.92 `double CoinModel::columnLower (int whichColumn) const` `[inline]`

Gets columnLower (if column does not exist then 0.0)

Definition at line 541 of file CoinModel.hpp.

9.43.3.93 `double CoinModel::columnUpper (int whichColumn) const` `[inline]`

Gets columnUpper (if column does not exist then COIN_DBL_MAX)

Definition at line 545 of file CoinModel.hpp.

9.43.3.94 `double CoinModel::columnObjective (int whichColumn) const` `[inline]`

Gets columnObjective (if column does not exist then 0.0)

Definition at line 549 of file CoinModel.hpp.

9.43.3.95 `double CoinModel::objective (int whichColumn) const` `[inline]`

Gets columnObjective (if column does not exist then 0.0)

Definition at line 553 of file CoinModel.hpp.

9.43.3.96 `const char* CoinModel::columnName (int whichColumn) const` `[inline]`

Gets name (if column does not exist then NULL)

Definition at line 557 of file CoinModel.hpp.

9.43.3.97 `bool CoinModel::columnIsInteger (int whichColumn) const` `[inline]`

Gets if integer (if column does not exist then false)

Definition at line 561 of file CoinModel.hpp.

9.43.3.98 `bool CoinModel::isInteger (int whichColumn) const` `[inline]`

Gets if integer (if column does not exist then false)

Definition at line 565 of file CoinModel.hpp.

9.43.3.99 `double CoinModel::getColLower (int whichColumn) const` `[inline]`

Gets columnLower (if column does not exist then 0.0)

Definition at line 569 of file CoinModel.hpp.

9.43.3.100 `double CoinModel::getColUpper (int whichColumn) const` `[inline]`

Gets columnUpper (if column does not exist then COIN_DBL_MAX)

Definition at line 573 of file CoinModel.hpp.

9.43.3.101 `double CoinModel::getColObjective (int whichColumn) const` `[inline]`

Gets columnObjective (if column does not exist then 0.0)

Definition at line 577 of file CoinModel.hpp.

9.43.3.102 `const char* CoinModel::getColName (int whichColumn) const` `[inline]`

Gets name (if column does not exist then NULL)

Definition at line 581 of file CoinModel.hpp.

9.43.3.103 `bool CoinModel::getCollsInteger (int whichColumn) const` `[inline]`

Gets if integer (if column does not exist then false)

Definition at line 585 of file CoinModel.hpp.

9.43.3.104 `const char* CoinModel::getRowLowerAsString (int whichRow) const`

Gets rowLower (if row does not exist then -COIN_DBL_MAX)

9.43.3.105 `const char* CoinModel::getRowUpperAsString (int whichRow) const`

Gets rowUpper (if row does not exist then +COIN_DBL_MAX)

9.43.3.106 `const char* CoinModel::rowLowerAsString (int whichRow) const` `[inline]`

Definition at line 593 of file CoinModel.hpp.

9.43.3.107 `const char* CoinModel::rowUpperAsString (int whichRow) const` `[inline]`

Gets rowUpper (if row does not exist then COIN_DBL_MAX)

Definition at line 597 of file CoinModel.hpp.

9.43.3.108 `const char* CoinModel::getColumnLowerAsString (int whichColumn) const`

Gets columnLower (if column does not exist then 0.0)

9.43.3.109 `const char* CoinModel::getColumnUpperAsString (int whichColumn) const`

Gets columnUpper (if column does not exist then COIN_DBL_MAX)

9.43.3.110 `const char* CoinModel::getColumnObjectiveAsString (int whichColumn) const`

Gets columnObjective (if column does not exist then 0.0)

9.43.3.111 `const char* CoinModel::getColumnIsIntegerAsString (int whichColumn) const`

Gets if integer (if column does not exist then false)

9.43.3.112 `const char* CoinModel::columnLowerAsString (int whichColumn) const` `[inline]`

Gets columnLower (if column does not exist then 0.0)

Definition at line 613 of file CoinModel.hpp.

9.43.3.113 `const char* CoinModel::columnUpperAsString (int whichColumn) const` `[inline]`

Gets columnUpper (if column does not exist then COIN_DBL_MAX)

Definition at line 617 of file CoinModel.hpp.

9.43.3.114 `const char* CoinModel::columnObjectiveAsString (int whichColumn) const` `[inline]`

Gets columnObjective (if column does not exist then 0.0)

Definition at line 621 of file CoinModel.hpp.

9.43.3.115 `const char* CoinModel::objectiveAsString (int whichColumn) const` `[inline]`

Gets columnObjective (if column does not exist then 0.0)

Definition at line 625 of file CoinModel.hpp.

9.43.3.116 `const char* CoinModel::columnIsIntegerAsString (int whichColumn) const` `[inline]`

Gets if integer (if column does not exist then false)

Definition at line 629 of file CoinModel.hpp.

9.43.3.117 `const char* CoinModel::isIntegerAsString (int whichColumn) const` `[inline]`

Gets if integer (if column does not exist then false)

Definition at line 633 of file CoinModel.hpp.

9.43.3.118 `int CoinModel::row (const char * rowName) const`

Row index from row name (-1 if no names or no match)

9.43.3.119 `int CoinModel::column (const char * columnName) const`

Column index from column name (-1 if no names or no match)

9.43.3.120 `int CoinModel::type () const` `[inline]`

Returns type.

Definition at line 640 of file CoinModel.hpp.

9.43.3.121 `double CoinModel::unsetValue () const` `[inline]`

returns unset value

Definition at line 643 of file CoinModel.hpp.

9.43.3.122 `int CoinModel::createPackedMatrix (CoinPackedMatrix & matrix, const double * associated)`

Creates a packed matrix - return number of errors.

9.43.3.123 `int CoinModel::countPlusMinusOne (CoinBigIndex * startPositive, CoinBigIndex * startNegative, const double * associated)`

Fills in startPositive and startNegative with counts for +-1 matrix.

If not +-1 then startPositive[0]==-1 otherwise counts and startPositive[numberColumns]== size

- return number of errors

9.43.3.124 `void CoinModel::createPlusMinusOne (CoinBigIndex * startPositive, CoinBigIndex * startNegative, int * indices, const double * associated)`

Creates +-1 matrix given startPositive and startNegative counts for +-1 matrix.

9.43.3.125 `int CoinModel::createArrays (double *& rowLower, double *& rowUpper, double *& columnLower, double *& columnUpper, double *& objective, int *& integerType, double *& associated)`

Creates copies of various arrays - return number of errors.

9.43.3.126 `bool CoinModel::stringsExist () const [inline]`

Says if strings exist.

Definition at line 666 of file CoinModel.hpp.

9.43.3.127 `const CoinModelHash* CoinModel::stringArray () const [inline]`

Return string array.

Definition at line 669 of file CoinModel.hpp.

9.43.3.128 `double* CoinModel::associatedArray () const [inline]`

Returns associated array.

Definition at line 672 of file CoinModel.hpp.

9.43.3.129 `double* CoinModel::rowLowerArray () const [inline]`

Return rowLower array.

Definition at line 675 of file CoinModel.hpp.

9.43.3.130 `double* CoinModel::rowUpperArray () const [inline]`

Return rowUpper array.

Definition at line 678 of file CoinModel.hpp.

9.43.3.131 `double* CoinModel::columnLowerArray () const [inline]`

Return columnLower array.

Definition at line 681 of file CoinModel.hpp.

9.43.3.132 `double* CoinModel::columnUpperArray () const [inline]`

Return columnUpper array.

Definition at line 684 of file CoinModel.hpp.

9.43.3.133 `double* CoinModel::objectiveArray () const [inline]`

Return objective array.

Definition at line 687 of file CoinModel.hpp.

9.43.3.134 `int* CoinModel::integerTypeArray () const [inline]`

Return integerType array.

Definition at line 690 of file CoinModel.hpp.

9.43.3.135 `const CoinModelHash* CoinModel::rowNames () const [inline]`

Return row names array.

Definition at line 693 of file CoinModel.hpp.

9.43.3.136 `const CoinModelHash* CoinModel::columnNames () const [inline]`

Return column names array.

Definition at line 696 of file CoinModel.hpp.

9.43.3.137 `void CoinModel::zapRowNames () [inline]`

Reset row names.

Definition at line 699 of file CoinModel.hpp.

9.43.3.138 `void CoinModel::zapColumnNames () [inline]`

Reset column names.

Definition at line 702 of file CoinModel.hpp.

9.43.3.139 `const int* CoinModel::cutMarker () const [inline]`

Returns array of 0 or nonzero if can be a cut (or returns NULL)

Definition at line 705 of file CoinModel.hpp.

9.43.3.140 `double CoinModel::optimizationDirection () const [inline]`

Direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.

Definition at line 708 of file CoinModel.hpp.

9.43.3.141 `void CoinModel::setOptimizationDirection (double value) [inline]`

Set direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.

Definition at line 712 of file CoinModel.hpp.

9.43.3.142 **void*** CoinModel::moreInfo () const [inline]

Return pointer to more information.

Definition at line 715 of file CoinModel.hpp.

9.43.3.143 **void** CoinModel::setMoreInfo (**void** * *info*) [inline]

Set pointer to more information.

Definition at line 718 of file CoinModel.hpp.

9.43.3.144 **int** CoinModel::whatIsSet () const

Returns which parts of model are set 1 - matrix 2 - rhs 4 - row names 8 - column bounds and/or objective 16 - column names 32 - integer types.

9.43.3.145 **void** CoinModel::loadBlock (const CoinPackedMatrix & *matrix*, const double * *collb*, const double * *colub*, const double * *obj*, const double * *rowlb*, const double * *rowub*)

Load in a problem by copying the arguments.

The constraints on the rows are given by lower and upper bounds.

If a pointer is 0 then the following values are the default:

- *colub*: all columns have upper bound infinity
- *collb*: all columns have lower bound 0
- *rowub*: all rows have upper bound infinity
- *rowlb*: all rows have lower bound -infinity
- *obj*: all variables have 0 objective coefficient

Note that the default values for *rowub* and *rowlb* produce the constraint $-inf \leq ax \leq inf$. This is probably not what you want.

9.43.3.146 **void** CoinModel::loadBlock (const CoinPackedMatrix & *matrix*, const double * *collb*, const double * *colub*, const double * *obj*, const char * *rowsen*, const double * *rowrhs*, const double * *rowrng*)

Load in a problem by copying the arguments.

The constraints on the rows are given by sense/rhs/range triplets.

If a pointer is 0 then the following values are the default:

- *colub*: all columns have upper bound infinity
- *collb*: all columns have lower bound 0
- *obj*: all variables have 0 objective coefficient
- *rowsen*: all rows are \geq
- *rowrhs*: all right hand sides are 0
- *rowrng*: 0 for the ranged rows

Note that the default values for *rowsen*, *rowrhs*, and *rowrng* produce the constraint $ax \geq 0$.

9.43.3.147 `void CoinModel::loadBlock (const int numcols, const int numrows, const CoinBigIndex * start, const int * index, const double * value, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub)`

Load in a problem by copying the arguments.

The constraint matrix is is specified with standard column-major column starts / row indices / coefficients vectors. The constraints on the rows are given by lower and upper bounds.

The matrix vectors must be gap-free. Note that *start* must have *numcols*+1 entries so that the length of the last column can be calculated as *start*[*numcols*]-*start*[*numcols*-1].

See the previous *loadBlock* method using *rowlb* and *rowub* for default argument values.

9.43.3.148 `void CoinModel::loadBlock (const int numcols, const int numrows, const CoinBigIndex * start, const int * index, const double * value, const double * collb, const double * colub, const double * obj, const char * rowsen, const double * rowrhs, const double * rowrng)`

Load in a problem by copying the arguments.

The constraint matrix is is specified with standard column-major column starts / row indices / coefficients vectors. The constraints on the rows are given by sense/rhs/range triplets.

The matrix vectors must be gap-free. Note that *start* must have *numcols*+1 entries so that the length of the last column can be calculated as *start*[*numcols*]-*start*[*numcols*-1].

See the previous *loadBlock* method using *sense/rhs/range* for default argument values.

9.43.3.149 `virtual CoinBaseModel* CoinModel::clone () const [virtual]`

Clone.

Implements [CoinBaseModel](#).

9.43.3.150 `CoinModel& CoinModel::operator= (const CoinModel &)`

=

9.43.3.151 `void CoinModel::validateLinks () const`

Checks that links are consistent.

9.43.3.152 `int CoinModel::computeAssociated (double * associated)`

Fills in all associated - returning number of errors.

9.43.3.153 `CoinPackedMatrix* CoinModel::quadraticRow (int rowNumber, double * linear, int & numberBad) const`

Gets correct form for a quadratic row - user to delete If row is not quadratic then returns which other variables are involved with tiny (1.0e-100) elements and count of total number of variables which could not be put in quadratic form.

9.43.3.154 `void CoinModel::replaceQuadraticRow (int rowNumber, const double * linear, const CoinPackedMatrix * quadraticPart)`

Replaces a quadratic row.

9.43.3.155 `CoinModel* CoinModel::reorder (const char * mark) const`

If possible return a model where if all variables marked nonzero are fixed the problem will be linear.

At present may only work if quadratic. Returns NULL if not possible

9.43.3.156 `int CoinModel::expandKnapsack (int knapsackRow, int & numberOutput, double * buildObj, CoinBigIndex * buildStart, int * buildRow, double * buildElement, int reConstruct = -1) const`

Expands out all possible combinations for a knapsack If buildObj NULL then just computes space needed - returns number elements On entry numberOutput is maximum allowed, on exit it is number needed or -1 (as will be number elements) if maximum exceeded.

numberOutput will have at least space to return values which reconstruct input. Rows returned will be original rows but no entries will be returned for any rows all of whose entries are in knapsack. So up to user to allow for this. If reConstruct >=0 then returns number of entrie which make up item "reConstruct" in expanded knapsack. Values in buildRow and buildElement;

9.43.3.157 `void CoinModel::setCutMarker (int size, const int * marker)`

Sets cut marker array.

9.43.3.158 `void CoinModel::setPriorities (int size, const int * priorities)`

Sets priority array.

9.43.3.159 `const int* CoinModel::priorities () const` `[inline]`

priorities (given for all columns (-1 if not integer)

Definition at line 910 of file CoinModel.hpp.

9.43.3.160 `void CoinModel::setOriginalIndices (const int * row, const int * column)`

For decomposition set original row and column indices.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinModel.hpp](#)

9.44 CoinModelHash Class Reference

```
#include <CoinModelUseful.hpp>
```

Public Member Functions

Constructors, destructor

- [CoinModelHash](#) ()
Default constructor.
- [~CoinModelHash](#) ()
Destructor.

Copy method

- [CoinModelHash](#) (const [CoinModelHash](#) &)
The copy constructor.
- [CoinModelHash](#) & [operator=](#) (const [CoinModelHash](#) &)
=

sizing (just increases)

- `void resize` (int maxItems, bool forceReHash=false)
Resize hash (also re-hashes)
- int `numberItems` () const
Number of items i.e. rows if just row names.
- `void setNumberItems` (int number)
Set number of items.
- int `maximumItems` () const
Maximum number of items.
- const char *const * `names` () const
Names.

hashing

- int `hash` (const char *`name`) const
Returns index or -1.
- `void addHash` (int index, const char *`name`)
Adds to hash.
- `void deleteHash` (int index)
Deletes from hash.
- const char * `name` (int which) const
Returns name at position (or NULL)
- char * `getName` (int which) const
Returns non const name at position (or NULL)
- `void setName` (int which, char *`name`)
Sets name at position (does not create)
- `void validateHash` () const
Validates.

9.44.1 Detailed Description

Definition at line 180 of file CoinModelUseful.hpp.

9.44.2 Constructor & Destructor Documentation

9.44.2.1 CoinModelHash::CoinModelHash ()

Default constructor.

9.44.2.2 CoinModelHash::~CoinModelHash ()

Destructor.

9.44.2.3 CoinModelHash::CoinModelHash (const CoinModelHash &)

The copy constructor.

9.44.3 Member Function Documentation

9.44.3.1 CoinModelHash& CoinModelHash::operator= (const CoinModelHash &)

=

9.44.3.2 `void CoinModelHash::resize (int maxItems, bool forceReHash = false)`

Resize hash (also re-hashes)

9.44.3.3 `int CoinModelHash::numberItems () const [inline]`

Number of items i.e. rows if just row names.

Definition at line 204 of file CoinModelUseful.hpp.

9.44.3.4 `void CoinModelHash::setNumberItems (int number)`

Set number of items.

9.44.3.5 `int CoinModelHash::maximumItems () const [inline]`

Maximum number of items.

Definition at line 209 of file CoinModelUseful.hpp.

9.44.3.6 `const char* const* CoinModelHash::names () const [inline]`

Names.

Definition at line 212 of file CoinModelUseful.hpp.

9.44.3.7 `int CoinModelHash::hash (const char * name) const`

Returns index or -1.

9.44.3.8 `void CoinModelHash::addHash (int index, const char * name)`

Adds to hash.

9.44.3.9 `void CoinModelHash::deleteHash (int index)`

Deletes from hash.

9.44.3.10 `const char* CoinModelHash::name (int which) const`

Returns name at position (or NULL)

9.44.3.11 `char* CoinModelHash::getName (int which) const`

Returns non const name at position (or NULL)

9.44.3.12 `void CoinModelHash::setName (int which, char * name)`

Sets name at position (does not create)

9.44.3.13 `void CoinModelHash::validateHash () const`

Validates.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinModelUseful.hpp](#)

9.45 CoinModelHash2 Class Reference

For int,int hashing.

```
#include <CoinModelUseful.hpp>
```

Public Member Functions

Constructors, destructor

- [CoinModelHash2](#) ()
Default constructor.
- [~CoinModelHash2](#) ()
Destructor.

Copy method

- [CoinModelHash2](#) (const [CoinModelHash2](#) &)
The copy constructor.
- [CoinModelHash2](#) & [operator=](#) (const [CoinModelHash2](#) &)
=

sizing (just increases)

- [void](#) [resize](#) (int maxItems, const [CoinModelTriple](#) *triples, bool forceReHash=false)
Resize hash (also re-hashes)
- int [numberItems](#) () const
Number of items.
- [void](#) [setNumberItems](#) (int number)
Set number of items.
- int [maximumItems](#) () const
Maximum number of items.

hashing

- int [hash](#) (int row, int column, const [CoinModelTriple](#) *triples) const
Returns index or -1.
- [void](#) [addHash](#) (int index, int row, int column, const [CoinModelTriple](#) *triples)
Adds to hash.
- [void](#) [deleteHash](#) (int index, int row, int column)
Deletes from hash.

9.45.1 Detailed Description

For int,int hashing.

Definition at line 253 of file CoinModelUseful.hpp.

9.45.2 Constructor & Destructor Documentation

9.45.2.1 CoinModelHash2::CoinModelHash2 ()

Default constructor.

9.45.2.2 CoinModelHash2::~~CoinModelHash2 ()

Destructor.

9.45.2.3 CoinModelHash2::CoinModelHash2 (const CoinModelHash2 &)

The copy constructor.

9.45.3 Member Function Documentation

9.45.3.1 CoinModelHash2& CoinModelHash2::operator= (const CoinModelHash2 &)

=

9.45.3.2 void CoinModelHash2::resize (int *maxItems*, const CoinModelTriple * *triples*, bool *forceReHash* = false)

Resize hash (also re-hashes)

9.45.3.3 int CoinModelHash2::numberItems () const [inline]

Number of items.

Definition at line 277 of file CoinModelUseful.hpp.

9.45.3.4 void CoinModelHash2::setNumberItems (int *number*)

Set number of items.

9.45.3.5 int CoinModelHash2::maximumItems () const [inline]

Maximum number of items.

Definition at line 282 of file CoinModelUseful.hpp.

9.45.3.6 int CoinModelHash2::hash (int *row*, int *column*, const CoinModelTriple * *triples*) const

Returns index or -1.

9.45.3.7 void CoinModelHash2::addHash (int *index*, int *row*, int *column*, const CoinModelTriple * *triples*)

Adds to hash.

9.45.3.8 void CoinModelHash2::deleteHash (int *index*, int *row*, int *column*)

Deletes from hash.

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinModelUseful.hpp](#)

9.46 CoinModelHashLink Struct Reference

for names and hashing

```
#include <CoinModelUseful.hpp>
```

Public Attributes

- int [index](#)
- int [next](#)

9.46.1 Detailed Description

for names and hashing

Definition at line 128 of file CoinModelUseful.hpp.

9.46.2 Member Data Documentation

9.46.2.1 int CoinModelHashLink::index

Definition at line 129 of file CoinModelUseful.hpp.

9.46.2.2 int CoinModelHashLink::next

Definition at line 129 of file CoinModelUseful.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinModelUseful.hpp](#)

9.47 CoinModelInfo2 Struct Reference

This is a model which is made up of Coin(Structured)Model blocks.

```
#include <CoinStructuredModel.hpp>
```

Public Member Functions

- [CoinModelInfo2](#) ()

Public Attributes

- int [rowBlock](#)
- int [columnBlock](#)
- char [matrix](#)
- char [rhs](#)
- char [rowName](#)
- char [integer](#)
- char [bounds](#)
- char [columnName](#)

9.47.1 Detailed Description

This is a model which is made up of Coin(Structured)Model blocks.

Definition at line 15 of file CoinStructuredModel.hpp.

9.47.2 Constructor & Destructor Documentation

9.47.2.1 CoinModelInfo2::CoinModelInfo2 () [inline]

Definition at line 24 of file CoinStructuredModel.hpp.

9.47.3 Member Data Documentation

9.47.3.1 int CoinModelInfo2::rowBlock

Definition at line 16 of file CoinStructuredModel.hpp.

9.47.3.2 int CoinModelInfo2::columnBlock

Definition at line 17 of file CoinStructuredModel.hpp.

9.47.3.3 char CoinModelInfo2::matrix

Definition at line 18 of file CoinStructuredModel.hpp.

9.47.3.4 char CoinModelInfo2::rhs

Definition at line 19 of file CoinStructuredModel.hpp.

9.47.3.5 char CoinModelInfo2::rowName

Definition at line 20 of file CoinStructuredModel.hpp.

9.47.3.6 char CoinModelInfo2::integer

Definition at line 21 of file CoinStructuredModel.hpp.

9.47.3.7 char CoinModelInfo2::bounds

Definition at line 22 of file CoinStructuredModel.hpp.

9.47.3.8 char CoinModelInfo2::columnName

Definition at line 23 of file CoinStructuredModel.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinStructuredModel.hpp](#)

9.48 CoinModelLink Class Reference

This is for various structures/classes needed by [CoinModel](#).

```
#include <CoinModelUseful.hpp>
```

Public Member Functions

Constructors, destructor

- [CoinModelLink](#) ()

- *Default constructor.*
• [~CoinModelLink](#) ()
Destructor.

Copy method

- [CoinModelLink](#) (const [CoinModelLink](#) &)
The copy constructor.
- [CoinModelLink](#) & [operator=](#) (const [CoinModelLink](#) &)
=

Sets and gets method

- int [row](#) () const
Get row.
- int [column](#) () const
Get column.
- double [value](#) () const
Get value.
- double [element](#) () const
Get value.
- int [position](#) () const
Get position.
- bool [onRow](#) () const
Get onRow.
- void [setRow](#) (int [row](#))
Set row.
- void [setColumn](#) (int [column](#))
Set column.
- void [setValue](#) (double [value](#))
Set value.
- void [setElement](#) (double [value](#))
Set value.
- void [setPosition](#) (int [position](#))
Set position.
- void [setOnRow](#) (bool [onRow](#))
Set onRow.

9.48.1 Detailed Description

This is for various structures/classes needed by [CoinModel](#).

[CoinModelLink](#) [CoinModelLinkedList](#) [CoinModelHashfor](#) for going through row or column

Definition at line 30 of file [CoinModelUseful.hpp](#).

9.48.2 Constructor & Destructor Documentation

9.48.2.1 [CoinModelLink::CoinModelLink](#) ()

Default constructor.

9.48.2.2 [CoinModelLink::~~CoinModelLink](#) ()

Destructor.

9.48.2.3 CoinModelLink::CoinModelLink (const CoinModelLink &)

The copy constructor.

9.48.3 Member Function Documentation

9.48.3.1 CoinModelLink& CoinModelLink::operator= (const CoinModelLink &)

=

9.48.3.2 int CoinModelLink::row () const [inline]

Get row.

Definition at line 52 of file CoinModelUseful.hpp.

9.48.3.3 int CoinModelLink::column () const [inline]

Get column.

Definition at line 55 of file CoinModelUseful.hpp.

9.48.3.4 double CoinModelLink::value () const [inline]

Get value.

Definition at line 58 of file CoinModelUseful.hpp.

9.48.3.5 double CoinModelLink::element () const [inline]

Get value.

Definition at line 61 of file CoinModelUseful.hpp.

9.48.3.6 int CoinModelLink::position () const [inline]

Get position.

Definition at line 64 of file CoinModelUseful.hpp.

9.48.3.7 bool CoinModelLink::onRow () const [inline]

Get onRow.

Definition at line 67 of file CoinModelUseful.hpp.

9.48.3.8 void CoinModelLink::setRow (int row) [inline]

Set row.

Definition at line 70 of file CoinModelUseful.hpp.

9.48.3.9 void CoinModelLink::setColumn (int column) [inline]

Set column.

Definition at line 73 of file CoinModelUseful.hpp.

9.48.3.10 **void CoinModelLink::setValue (double *value*)** [inline]

Set value.

Definition at line 76 of file CoinModelUseful.hpp.

9.48.3.11 **void CoinModelLink::setElement (double *value*)** [inline]

Set value.

Definition at line 79 of file CoinModelUseful.hpp.

9.48.3.12 **void CoinModelLink::setPosition (int *position*)** [inline]

Set position.

Definition at line 82 of file CoinModelUseful.hpp.

9.48.3.13 **void CoinModelLink::setOnRow (bool *onRow*)** [inline]

Set onRow.

Definition at line 85 of file CoinModelUseful.hpp.

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinModelUseful.hpp](#)

9.49 CoinModelLinkedList Class Reference

```
#include <CoinModelUseful.hpp>
```

Public Member Functions

Constructors, destructor

- [CoinModelLinkedList](#) ()
Default constructor.
- [~CoinModelLinkedList](#) ()
Destructor.

Copy method

- [CoinModelLinkedList](#) (const [CoinModelLinkedList](#) &)
The copy constructor.
- [CoinModelLinkedList](#) & **operator=** (const [CoinModelLinkedList](#) &)
=

sizing (just increases)

- [void](#) **resize** (int maxMajor, int maxElements)
Resize list - for row list maxMajor is maximum rows.
- [void](#) **create** (int maxMajor, int maxElements, int [numberMajor](#), int numberMinor, int type, int [numberElements](#), const [CoinModelTriple](#) *triples)
Create list - for row list maxMajor is maximum rows.
- int [numberMajor](#) () const

- *Number of major items i.e. rows if just row links.*
int [maximumMajor](#) () const
- *Maximum number of major items i.e. rows if just row links.*
int [numberElements](#) () const
- *Number of elements.*
int [maximumElements](#) () const
- *Maximum number of elements.*
int [firstFree](#) () const
- *First on free chain.*
int [lastFree](#) () const
- *Last on free chain.*
int [first](#) (int which) const
- *First on chain.*
int [last](#) (int which) const
- *Last on chain.*
const int * [next](#) () const
- *Next array.*
const int * [previous](#) () const
- *Previous array.*

does work

- int [addEasy](#) (int majorIndex, int numberOfElements, const int *indices, const double *elements, [CoinModelTriple](#) *triples, [CoinModelHash2](#) &hash)
Adds to list - easy case i.e.
- void [addHard](#) (int minorIndex, int numberOfElements, const int *indices, const double *elements, [CoinModelTriple](#) *triples, [CoinModelHash2](#) &hash)
Adds to list - hard case i.e.
- void [addHard](#) (int [first](#), const [CoinModelTriple](#) *triples, int [firstFree](#), int [lastFree](#), const int *nextOther)
Adds to list - hard case i.e.
- void [deleteSame](#) (int which, [CoinModelTriple](#) *triples, [CoinModelHash2](#) &hash, bool zapTriples)
Deletes from list - same case i.e.
- void [updateDeleted](#) (int which, [CoinModelTriple](#) *triples, [CoinModelLinkedList](#) &otherList)
Deletes from list - other case i.e.
- void [deleteRowOne](#) (int position, [CoinModelTriple](#) *triples, [CoinModelHash2](#) &hash)
Deletes one element from Row list.
- void [updateDeletedOne](#) (int position, const [CoinModelTriple](#) *triples)
Update column list for one element when one element deleted from row copy.
- void [fill](#) (int [first](#), int [last](#))
Fills first,last with -1.
- void [synchronize](#) ([CoinModelLinkedList](#) &other)
Puts in free list from other list.
- void [validateLinks](#) (const [CoinModelTriple](#) *triples) const
Checks that links are consistent.

9.49.1 Detailed Description

Definition at line 312 of file [CoinModelUseful.hpp](#).

9.49.2 Constructor & Destructor Documentation

9.49.2.1 CoinModelLinkedList::CoinModelLinkedList ()

Default constructor.

9.49.2.2 CoinModelLinkedList::~~CoinModelLinkedList ()

Destructor.

9.49.2.3 CoinModelLinkedList::CoinModelLinkedList (const CoinModelLinkedList &)

The copy constructor.

9.49.3 Member Function Documentation

9.49.3.1 CoinModelLinkedList& CoinModelLinkedList::operator= (const CoinModelLinkedList &)

=

9.49.3.2 void CoinModelLinkedList::resize (int maxMajor, int maxElements)

Resize list - for row list maxMajor is maximum rows.

9.49.3.3 void CoinModelLinkedList::create (int maxMajor, int maxElements, int numberMajor, int numberMinor, int type, int numberElements, const CoinModelTriple * triples)

Create list - for row list maxMajor is maximum rows.

type 0 row list, 1 column list

9.49.3.4 int CoinModelLinkedList::numberMajor () const [inline]

Number of major items i.e. rows if just row links.

Definition at line 344 of file CoinModelUseful.hpp.

9.49.3.5 int CoinModelLinkedList::maximumMajor () const [inline]

Maximum number of major items i.e. rows if just row links.

Definition at line 347 of file CoinModelUseful.hpp.

9.49.3.6 int CoinModelLinkedList::numberElements () const [inline]

Number of elements.

Definition at line 350 of file CoinModelUseful.hpp.

9.49.3.7 int CoinModelLinkedList::maximumElements () const [inline]

Maximum number of elements.

Definition at line 353 of file CoinModelUseful.hpp.

9.49.3.8 int CoinModelLinkedList::firstFree () const [inline]

First on free chain.

Definition at line 356 of file CoinModelUseful.hpp.

9.49.3.9 int CoinModelLinkedList::lastFree () const [inline]

Last on free chain.

Definition at line 359 of file CoinModelUseful.hpp.

9.49.3.10 `int CoinModelLinkedList::first (int which) const` `[inline]`

First on chain.

Definition at line 362 of file CoinModelUseful.hpp.

9.49.3.11 `int CoinModelLinkedList::last (int which) const` `[inline]`

Last on chain.

Definition at line 365 of file CoinModelUseful.hpp.

9.49.3.12 `const int* CoinModelLinkedList::next () const` `[inline]`

Next array.

Definition at line 368 of file CoinModelUseful.hpp.

9.49.3.13 `const int* CoinModelLinkedList::previous () const` `[inline]`

Previous array.

Definition at line 371 of file CoinModelUseful.hpp.

9.49.3.14 `int CoinModelLinkedList::addEasy (int majorIndex, int numberOfElements, const int * indices, const double * elements, CoinModelTriple * triples, CoinModelHash2 & hash)`

Adds to list - easy case i.e.

add row to row list Returns where chain starts

9.49.3.15 `void CoinModelLinkedList::addHard (int minorIndex, int numberOfElements, const int * indices, const double * elements, CoinModelTriple * triples, CoinModelHash2 & hash)`

Adds to list - hard case i.e.

add row to column list

9.49.3.16 `void CoinModelLinkedList::addHard (int first, const CoinModelTriple * triples, int firstFree, int lastFree, const int * nextOther)`

Adds to list - hard case i.e.

add row to column list This is when elements have been added to other copy

9.49.3.17 `void CoinModelLinkedList::deleteSame (int which, CoinModelTriple * triples, CoinModelHash2 & hash, bool zapTriples)`

Deletes from list - same case i.e.

delete row from row list

9.49.3.18 `void CoinModelLinkedList::updateDeleted (int which, CoinModelTriple * triples, CoinModelLinkedList & otherList)`

Deletes from list - other case i.e.

delete row from column list This is when elements have been deleted from other copy

9.49.3.19 **void** CoinModelLinkedList::deleteRowOne (int *position*, CoinModelTriple * *triples*, CoinModelHash2 & *hash*)

Deletes one element from Row list.

9.49.3.20 **void** CoinModelLinkedList::updateDeletedOne (int *position*, const CoinModelTriple * *triples*)

Update column list for one element when one element deleted from row copy.

9.49.3.21 **void** CoinModelLinkedList::fill (int *first*, int *last*)

Fills first,last with -1.

9.49.3.22 **void** CoinModelLinkedList::synchronize (CoinModelLinkedList & *other*)

Puts in free list from other list.

9.49.3.23 **void** CoinModelLinkedList::validateLinks (const CoinModelTriple * *triples*) const

Checks that links are consistent.

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinModelUseful.hpp](#)

9.50 CoinModelTriple Struct Reference

for linked lists

```
#include <CoinModelUseful.hpp>
```

Public Attributes

- unsigned int [row](#)
- int [column](#)
- double [value](#)

9.50.1 Detailed Description

for linked lists

Definition at line 107 of file CoinModelUseful.hpp.

9.50.2 Member Data Documentation

9.50.2.1 unsigned int CoinModelTriple::row

Definition at line 110 of file CoinModelUseful.hpp.

9.50.2.2 int CoinModelTriple::column

Definition at line 112 of file CoinModelUseful.hpp.

9.50.2.3 double CoinModelTriple::value

Definition at line 113 of file CoinModelUseful.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinModelUseful.hpp](#)

9.51 CoinMpsCardReader Class Reference

Very simple code for reading MPS data.

```
#include <CoinMpsIO.hpp>
```

Public Member Functions

Constructor and destructor

- [CoinMpsCardReader](#) ([CoinFileInput](#) *input, [CoinMpsIO](#) *reader)
Constructor expects file to be open This one takes gzFile if fp null.
- [~CoinMpsCardReader](#) ()
Destructor.

card stuff

- [COINSectionType readToNextSection](#) ()
Read to next section.
- [COINSectionType nextField](#) ()
Gets next field and returns section type e.g. COIN_COLUMN_SECTION.
- [int nextGmsField](#) (int expectedType)
Gets next field for .gms file and returns type.
- [COINSectionType whichSection](#) () const
Returns current section type.
- [void setWhichSection](#) ([COINSectionType](#) section)
Sets current section type.
- [bool freeFormat](#) () const
Sees if free format.
- [void setFreeFormat](#) (bool yesNo)
Sets whether free format. Mainly for blank RHS etc.
- [COINMpsType mpsType](#) () const
Only for first field on card otherwise BLANK_COLUMN e.g.
- [int cleanCard](#) ()
Reads and cleans card - taking out trailing blanks - return 1 if EOF.
- [const char * rowName](#) () const
Returns row name of current field.
- [const char * columnName](#) () const
Returns column name of current field.
- [double value](#) () const
Returns value in current field.
- [const char * valueString](#) () const
Returns value as string in current field.
- [const char * card](#) () const
Whole card (for printing)
- [char * mutableCard](#) ()

- *Whole card - so we look at it (not const so nextBlankOr will work for gms reader)*
- `void setPosition (char *position)`
set position (again so gms reader will work)
- `char * getPosition () const`
get position (again so gms reader will work)
- `CoinBigIndex cardNumber () const`
Returns card number.
- `CoinFileInput * fileInput () const`
Returns file input.
- `void setStringsAllowed ()`
Sets whether strings allowed.

Protected Attributes

data

- `double value_`
Current value.
- `char card_ [MAX_CARD_LENGTH]`
Current card image.
- `char * position_`
Current position within card image.
- `char * eol_`
End of card.
- `COINMpsType mpsType_`
Current COINMpsType.
- `char rowName_ [COIN_MAX_FIELD_LENGTH]`
Current row name.
- `char columnName_ [COIN_MAX_FIELD_LENGTH]`
Current column name.
- `CoinFileInput * input_`
File input.
- `COINSectionType section_`
Which section we think we are in.
- `CoinBigIndex cardNumber_`
Card number.
- `bool freeFormat_`
Whether free format. Just for blank RHS etc.
- `int ieeeFormat_`
Whether IEEE - 0 no, 1 INTEL, 2 not INTEL.
- `bool eightChar_`
If all names <= 8 characters then allow embedded blanks.
- `CoinMpsIO * reader_`
MpsIO.
- `CoinMessageHandler * handler_`
Message handler.
- `CoinMessages messages_`
Messages.
- `char valueString_ [COIN_MAX_FIELD_LENGTH]`
Current element as characters (only if strings allowed)
- `bool stringsAllowed_`
Whether strings allowed.

methods

- double [osi_strtod](#) (char *ptr, char **output, int type)
type - 0 normal, 1 INTEL IEEE, 2 other IEEE
- double [osi_strtod](#) (char *ptr, char **output)
For strings.
- static void [strcpyAndCompress](#) (char *to, const char *from)
remove blanks
- static char * [nextBlankOr](#) (char *image)

9.51.1 Detailed Description

Very simple code for reading MPS data.

Definition at line 59 of file CoinMpsIO.hpp.

9.51.2 Constructor & Destructor Documentation

9.51.2.1 CoinMpsCardReader::CoinMpsCardReader (CoinFileInput * input, CoinMpsIO * reader)

Constructor expects file to be open This one takes gzFile if fp null.

9.51.2.2 CoinMpsCardReader::~CoinMpsCardReader ()

Destructor.

9.51.3 Member Function Documentation

9.51.3.1 COINSectionType CoinMpsCardReader::readToNextSection ()

Read to next section.

9.51.3.2 COINSectionType CoinMpsCardReader::nextField ()

Gets next field and returns section type e.g. COIN_COLUMN_SECTION.

9.51.3.3 int CoinMpsCardReader::nextGmsField (int expectedType)

Gets next field for .gms file and returns type.

-1 - EOF 0 - what we expected (and processed so pointer moves past) 1 - not what we expected leading blanks always ignored input types 0 - anything - stops on non blank card 1 - name (in columnname) 2 - value 3 - value name pair 4 - equation type 5 - ;

9.51.3.4 COINSectionType CoinMpsCardReader::whichSection () const [inline]

Returns current section type.

Definition at line 95 of file CoinMpsIO.hpp.

9.51.3.5 void CoinMpsCardReader::setWhichSection (COINSectionType section) [inline]

Sets current section type.

Definition at line 99 of file CoinMpsIO.hpp.

9.51.3.6 `bool CoinMpsCardReader::freeFormat () const [inline]`

Sees if free format.

Definition at line 103 of file CoinMpsIO.hpp.

9.51.3.7 `void CoinMpsCardReader::setFreeFormat (bool yesNo) [inline]`

Sets whether free format. Mainly for blank RHS etc.

Definition at line 106 of file CoinMpsIO.hpp.

9.51.3.8 `COINMpsType CoinMpsCardReader::mpsType () const [inline]`

Only for first field on card otherwise BLANK_COLUMN e.g.

COIN_E_ROW

Definition at line 110 of file CoinMpsIO.hpp.

9.51.3.9 `int CoinMpsCardReader::cleanCard ()`

Reads and cleans card - taking out trailing blanks - return 1 if EOF.

9.51.3.10 `const char* CoinMpsCardReader::rowName () const [inline]`

Returns row name of current field.

Definition at line 116 of file CoinMpsIO.hpp.

9.51.3.11 `const char* CoinMpsCardReader::columnName () const [inline]`

Returns column name of current field.

Definition at line 120 of file CoinMpsIO.hpp.

9.51.3.12 `double CoinMpsCardReader::value () const [inline]`

Returns value in current field.

Definition at line 124 of file CoinMpsIO.hpp.

9.51.3.13 `const char* CoinMpsCardReader::valueString () const [inline]`

Returns value as string in current field.

Definition at line 128 of file CoinMpsIO.hpp.

9.51.3.14 `const char* CoinMpsCardReader::card () const [inline]`

Whole card (for printing)

Definition at line 132 of file CoinMpsIO.hpp.

9.51.3.15 `char* CoinMpsCardReader::mutableCard () [inline]`

Whole card - so we look at it (not const so nextBlankOr will work for gms reader)

Definition at line 136 of file CoinMpsIO.hpp.

9.51.3.16 **void** CoinMpsCardReader::setPosition (*char * position*) [inline]

set position (again so gms reader will work)

Definition at line 140 of file CoinMpsIO.hpp.

9.51.3.17 **char*** CoinMpsCardReader::getPosition () **const** [inline]

get position (again so gms reader will work)

Definition at line 143 of file CoinMpsIO.hpp.

9.51.3.18 **CoinBigIndex** CoinMpsCardReader::cardNumber () **const** [inline]

Returns card number.

Definition at line 146 of file CoinMpsIO.hpp.

9.51.3.19 **CoinFileInput*** CoinMpsCardReader::fileInput () **const** [inline]

Returns file input.

Definition at line 150 of file CoinMpsIO.hpp.

9.51.3.20 **void** CoinMpsCardReader::setStringsAllowed () [inline]

Sets whether strings allowed.

Definition at line 154 of file CoinMpsIO.hpp.

9.51.3.21 **double** CoinMpsCardReader::osi_strtod (*char * ptr*, *char ** output*, *int type*)

type - 0 normal, 1 INTEL IEEE, 2 other IEEE

9.51.3.22 **static void** CoinMpsCardReader::strcpyAndCompress (*char * to*, *const char * from*) [static]

remove blanks

9.51.3.23 **static char*** CoinMpsCardReader::nextBlankOr (*char * image*) [static]

9.51.3.24 **double** CoinMpsCardReader::osi_strtod (*char * ptr*, *char ** output*)

For strings.

9.51.4 Member Data Documentation

9.51.4.1 **double** CoinMpsCardReader::value_ [protected]

Current value.

Definition at line 164 of file CoinMpsIO.hpp.

9.51.4.2 **char** CoinMpsCardReader::card_[MAX_CARD_LENGTH] [protected]

Current card image.

Definition at line 166 of file CoinMpsIO.hpp.

9.51.4.3 `char* CoinMpsCardReader::position_` [protected]

Current position within card image.

Definition at line 168 of file CoinMpsIO.hpp.

9.51.4.4 `char* CoinMpsCardReader::eol_` [protected]

End of card.

Definition at line 170 of file CoinMpsIO.hpp.

9.51.4.5 `COINMpsType CoinMpsCardReader::mpsType_` [protected]

Current COINMpsType.

Definition at line 172 of file CoinMpsIO.hpp.

9.51.4.6 `char CoinMpsCardReader::rowName_[COIN_MAX_FIELD_LENGTH]` [protected]

Current row name.

Definition at line 174 of file CoinMpsIO.hpp.

9.51.4.7 `char CoinMpsCardReader::columnName_[COIN_MAX_FIELD_LENGTH]` [protected]

Current column name.

Definition at line 176 of file CoinMpsIO.hpp.

9.51.4.8 `CoinFileInput* CoinMpsCardReader::input_` [protected]

File input.

Definition at line 178 of file CoinMpsIO.hpp.

9.51.4.9 `COINSectionType CoinMpsCardReader::section_` [protected]

Which section we think we are in.

Definition at line 180 of file CoinMpsIO.hpp.

9.51.4.10 `CoinBigIndex CoinMpsCardReader::cardNumber_` [protected]

Card number.

Definition at line 182 of file CoinMpsIO.hpp.

9.51.4.11 `bool CoinMpsCardReader::freeFormat_` [protected]

Whether free format. Just for blank RHS etc.

Definition at line 184 of file CoinMpsIO.hpp.

9.51.4.12 `int CoinMpsCardReader::ieeeFormat_` [protected]

Whether IEEE - 0 no, 1 INTEL, 2 not INTEL.

Definition at line 186 of file CoinMpsIO.hpp.

9.51.4.13 `bool CoinMpsCardReader::eightChar_` [protected]

If all names ≤ 8 characters then allow embedded blanks.

Definition at line 188 of file CoinMpsIO.hpp.

9.51.4.14 `CoinMpsIO* CoinMpsCardReader::reader_` [protected]

MpsIO.

Definition at line 190 of file CoinMpsIO.hpp.

9.51.4.15 `CoinMessageHandler* CoinMpsCardReader::handler_` [protected]

Message handler.

Definition at line 192 of file CoinMpsIO.hpp.

9.51.4.16 `CoinMessages CoinMpsCardReader::messages_` [protected]

Messages.

Definition at line 194 of file CoinMpsIO.hpp.

9.51.4.17 `char CoinMpsCardReader::valueString_[COIN_MAX_FIELD_LENGTH]` [protected]

Current element as characters (only if strings allowed)

Definition at line 196 of file CoinMpsIO.hpp.

9.51.4.18 `bool CoinMpsCardReader::stringsAllowed_` [protected]

Whether strings allowed.

Definition at line 198 of file CoinMpsIO.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinMpsIO.hpp](#)

9.52 CoinMpsIO Class Reference

MPS IO Interface.

```
#include <CoinMpsIO.hpp>
```

Classes

- struct [CoinHashLink](#)

Public Member Functions

Methods to retrieve problem information

These methods return information about the problem held by the [CoinMpsIO](#) object.

Querying an object that has no data associated with it result in zeros for the number of rows and columns, and NULL pointers from the methods that return vectors. Const pointers returned from any data-query method are always valid

- int `getNumCols` () const
Get number of columns.
- int `getNumRows` () const
Get number of rows.
- int `getNumElements` () const
Get number of nonzero elements.
- const double * `getColLower` () const
Get pointer to array[getNumCols()] of column lower bounds.
- const double * `getColUpper` () const
Get pointer to array[getNumCols()] of column upper bounds.
- const char * `getRowSense` () const
Get pointer to array[getNumRows()] of constraint senses.
- const double * `getRightHandSide` () const
Get pointer to array[getNumRows()] of constraint right-hand sides.
- const double * `getRowRange` () const
Get pointer to array[getNumRows()] of row ranges.
- const double * `getRowLower` () const
Get pointer to array[getNumRows()] of row lower bounds.
- const double * `getRowUpper` () const
Get pointer to array[getNumRows()] of row upper bounds.
- const double * `getObjCoefficients` () const
Get pointer to array[getNumCols()] of objective function coefficients.
- const `CoinPackedMatrix` * `getMatrixByRow` () const
Get pointer to row-wise copy of the coefficient matrix.
- const `CoinPackedMatrix` * `getMatrixByCol` () const
Get pointer to column-wise copy of the coefficient matrix.
- bool `isContinuous` (int colNumber) const
Return true if column is a continuous variable.
- bool `isInteger` (int columnNumber) const
Return true if a column is an integer variable.
- const char * `integerColumns` () const
Returns array[getNumCols()] specifying if a variable is integer.
- const char * `rowName` (int index) const
Returns the row name for the specified index.
- const char * `columnName` (int index) const
Returns the column name for the specified index.
- int `rowIndex` (const char *name) const
Returns the index for the specified row name.
- int `columnIndex` (const char *name) const
Returns the index for the specified column name.
- double `objectiveOffset` () const
Returns the (constant) objective offset.
- void `setObjectiveOffset` (double value)
Set objective offset.
- const char * `getProblemName` () const
Return the problem name.
- const char * `getObjectiveName` () const
Return the objective name.
- const char * `getRhsName` () const
Return the RHS vector name.
- const char * `getRangeName` () const
Return the range vector name.
- const char * `getBoundName` () const
Return the bound vector name.
- int `numberStringElements` () const

Number of string elements.

- const char * [stringElement](#) (int i) const
String element.

Methods to set problem information

Methods to load a problem into the [CoinMpsIO](#) object.

- [void setMpsData](#) (const [CoinPackedMatrix](#) &m, const double infinity, const double *collb, const double *colub, const double *obj, const char *integrality, const double *rowlb, const double *rowub, char const *const *const colnames, char const *const *const rownames)
Set the problem data.
- [void setMpsData](#) (const [CoinPackedMatrix](#) &m, const double infinity, const double *collb, const double *colub, const double *obj, const char *integrality, const double *rowlb, const double *rowub, const std::vector< std::string > &colnames, const std::vector< std::string > &rownames)
- [void setMpsData](#) (const [CoinPackedMatrix](#) &m, const double infinity, const double *collb, const double *colub, const double *obj, const char *integrality, const char *rowSEN, const double *rowRHS, const double *rowRNG, char const *const *const colnames, char const *const *const rownames)
- [void setMpsData](#) (const [CoinPackedMatrix](#) &m, const double infinity, const double *collb, const double *colub, const double *obj, const char *integrality, const char *rowSEN, const double *rowRHS, const double *rowRNG, const std::vector< std::string > &colnames, const std::vector< std::string > &rownames)
- [void copyInIntegerInformation](#) (const char *integerInformation)
Pass in an array[[getNumCols\(\)](#)] specifying if a variable is integer.
- [void setProblemName](#) (const char *name)
Set problem name.
- [void setObjectiveName](#) (const char *name)
Set objective name.

Parameter set/get methods

Methods to set and retrieve MPS IO parameters.

- [void setInfinity](#) (double value)
Set infinity.
- double [getInfinity](#) () const
Get infinity.
- [void setDefaultBound](#) (int value)
Set default upper bound for integer variables.
- int [getDefaultBound](#) () const
Get default upper bound for integer variables.
- int [allowStringElements](#) () const
Whether to allow string elements.
- [void setAllowStringElements](#) (int yesNo)
Whether to allow string elements (0 no, 1 yes, 2 yes and try flip)
- double [getSmallElementValue](#) () const
Small element value - elements less than this set to zero on input default is 1.0e-14.
- [void setSmallElementValue](#) (double value)

Methods for problem input and output

Methods to read and write MPS format problem files.

The read and write methods return the number of errors that occurred during the IO operation, or -1 if no file is opened.

Note

If the [CoinMpsIO](#) class was compiled with support for libz then readMps will automatically try to append .gz to the file name and open it as a compressed file if the specified file name cannot be opened. (Automatic append of the .bz2 suffix when libbz is used is on the TODO list.)

Todo Allow for file pointers and positioning

- [void setFileName](#) (const char *name)
Set the current file name for the [CoinMpsIO](#) object.
- const char * [getFileName](#) () const
Get the current file name for the [CoinMpsIO](#) object.
- int [readMps](#) (const char *filename, const char *extension="mps")
Read a problem in MPS format from the given filename.
- int [readMps](#) (const char *filename, const char *extension, int &numberSets, [CoinSet](#) **&sets)
Read a problem in MPS format from the given filename.
- int [readMps](#) ()
Read a problem in MPS format from a previously opened file.
- int [readMps](#) (int &numberSets, [CoinSet](#) **&sets)
and
- int [readBasis](#) (const char *filename, const char *extension, double *solution, unsigned char *rowStatus, unsigned char *columnStatus, const std::vector< std::string > &colnames, int numberColumns, const std::vector< std::string > &rownames, int numberOfRows)
Read a basis in MPS format from the given filename.
- int [readGms](#) (const char *filename, const char *extension="gms", bool convertObjective=false)
Read a problem in GAMS format from the given filename.
- int [readGms](#) (const char *filename, const char *extension, int &numberSets, [CoinSet](#) **&sets)
Read a problem in GAMS format from the given filename.
- int [readGms](#) (int &numberSets, [CoinSet](#) **&sets)
Read a problem in GAMS format from a previously opened file.
- int [readGMPL](#) (const char *modelName, const char *dataName=NULL, bool keepNames=false)
Read a problem in GMPL (subset of AMPL) format from the given filenames.
- int [writeMps](#) (const char *filename, int compression=0, int formatType=0, int numberAcross=2, [CoinPackedMatrix](#) *quadratic=NULL, int numberSOS=0, const [CoinSet](#) *setInfo=NULL) const
Write the problem in MPS format to a file with the given filename.
- const [CoinMpsCardReader](#) * [reader](#) () const
Return card reader object so can see what last card was e.g. QUADOBJ.
- int [readQuadraticMps](#) (const char *filename, int *&columnStart, int *&column, double *&elements, int checkSymmetry)
Read in a quadratic objective from the given filename.
- int [readConicMps](#) (const char *filename, int *&columnStart, int *&column, int *&coneType, int &numberCones)
Read in a list of cones from the given filename.
- [void setConvertObjective](#) (bool trueFalse)
Set whether to move objective from matrix.
- int [copyStringElements](#) (const [CoinModel](#) *model)
copies in strings from a [CoinModel](#) - returns number

Constructors and destructors

- [CoinMpsIO](#) ()
Default Constructor.
- [CoinMpsIO](#) (const [CoinMpsIO](#) &)
Copy constructor.
- [CoinMpsIO](#) & [operator=](#) (const [CoinMpsIO](#) &rhs)
Assignment operator.
- [~CoinMpsIO](#) ()

Destructor.

Message handling

- `void passInMessageHandler (CoinMessageHandler *handler)`
Pass in Message handler.
- `void newLanguage (CoinMessages::Language language)`
Set the language for messages.
- `void setLanguage (CoinMessages::Language language)`
Set the language for messages.
- `CoinMessageHandler * messageHandler () const`
Return the message handler.
- `CoinMessages messages ()`
Return the messages.
- `CoinMessages * messagesPointer ()`
Return the messages pointer.

Methods to release storage

These methods allow the client to reduce the storage used by the `CoinMpsIO` object by selectively releasing un-needed problem information.

- `void releaseRedundantInformation ()`
Release all information which can be re-calculated.
- `void releaseRowInformation ()`
Release all row information (lower, upper)
- `void releaseColumnInformation ()`
Release all column information (lower, upper, objective)
- `void releaseIntegerInformation ()`
Release integer information.
- `void releaseRowNames ()`
Release row names.
- `void releaseColumnNames ()`
Release column names.
- `void releaseMatrixInformation ()`
Release matrix information.

Protected Member Functions

Miscellaneous helper functions

- `void setMpsDataWithoutRowAndColNames (const CoinPackedMatrix &m, const double infinity, const double *collb, const double *colub, const double *obj, const char *integrality, const double *rowlb, const double *rowub)`
Utility method used several times to implement public methods.
- `void setMpsDataColAndRowNames (const std::vector< std::string > &colnames, const std::vector< std::string > &rownames)`
- `void setMpsDataColAndRowNames (char const *const *const colnames, char const *const *const rownames)`
- `void gutsOfDestructor ()`
Does the heavy lifting for destruct and assignment.
- `void gutsOfCopy (const CoinMpsIO &)`
Does the heavy lifting for copy and assignment.
- `void freeAll ()`
Clears problem data from the `CoinMpsIO` object.

- [void convertBoundToSense](#) (const double lower, const double upper, char &sense, double &right, double &range) const
A quick inlined function to convert from lb/ub style constraint definition to sense/rhs/range style.
- [void convertSenseToBound](#) (const char sense, const double right, const double range, double &lower, double &upper) const
A quick inlined function to convert from sense/rhs/range style constraint definition to lb/ub style.
- [int dealWithFileName](#) (const char *filename, const char *extension, [CoinFileInput](#) *&input)
Deal with a filename.
- [void addString](#) (int iRow, int iColumn, const char *value)
Add string to list iRow==numberRows is objective, nr+1 is lo, nr+2 is up iColumn==nc is rhs (can't cope with ranges at present)
- [void decodeString](#) (int iString, int &iRow, int &iColumn, const char *&value) const
Decode string.

Hash table methods

- [void startHash](#) (char **names, const int number, int section)
Creates hash list for names (section = 0 for rows, 1 columns)
- [void startHash](#) (int section) const
This one does it when names are already in.
- [void stopHash](#) (int section)
Deletes hash storage.
- [int findHash](#) (const char *name, int section) const
Finds match using hash, -1 not found.

Protected Attributes

Cached problem information

- char * [problemName_](#)
Problem name.
- char * [objectiveName_](#)
Objective row name.
- char * [rhsName_](#)
Right-hand side vector name.
- char * [rangeName_](#)
Range vector name.
- char * [boundName_](#)
Bounds vector name.
- int [numberRows_](#)
Number of rows.
- int [numberColumns_](#)
Number of columns.
- [CoinBigIndex](#) [numberElements_](#)
Number of coefficients.
- char * [rowsense_](#)
Pointer to dense vector of row sense indicators.
- double * [rhs_](#)
Pointer to dense vector of row right-hand side values.
- double * [rowrange_](#)
Pointer to dense vector of slack variable upper bounds for range constraints (undefined for non-range rows)
- [CoinPackedMatrix](#) * [matrixByRow_](#)
Pointer to row-wise copy of problem matrix coefficients.
- [CoinPackedMatrix](#) * [matrixByColumn_](#)

- *Pointer to column-wise copy of problem matrix coefficients.*
- double * [rowlower_](#)
Pointer to dense vector of row lower bounds.
- double * [rowupper_](#)
Pointer to dense vector of row upper bounds.
- double * [collower_](#)
Pointer to dense vector of column lower bounds.
- double * [colupper_](#)
Pointer to dense vector of column upper bounds.
- double * [objective_](#)
Pointer to dense vector of objective coefficients.
- double [objectiveOffset_](#)
Constant offset for objective value (i.e., RHS value for OBJ row)
- char * [integerType_](#)
Pointer to dense vector specifying if a variable is continuous (0) or integer (1).
- char ** [names_](#) [2]
Row and column names Linked to hash table sections (0 - row names, 1 column names)

Hash tables

- char * [fileName_](#)
Current file name.
- int [numberHash_](#) [2]
Number of entries in a hash table section.
- [CoinHashLink](#) * [hash_](#) [2]
Hash tables (two sections, 0 - row names, 1 - column names)

CoinMpsIO object parameters

- int [defaultBound_](#)
Upper bound when no bounds for integers.
- double [infinity_](#)
Value to use for infinity.
- double [smallElement_](#)
Small element value.
- [CoinMessageHandler](#) * [handler_](#)
Message handler.
- bool [defaultHandler_](#)
Flag to say if the message handler is the default handler.
- [CoinMessages](#) [messages_](#)
Messages.
- [CoinMpsCardReader](#) * [cardReader_](#)
Card reader.
- bool [convertObjective_](#)
If .gms file should it be massaged to move objective.
- int [allowStringElements_](#)
Whether to allow string elements.
- int [maximumStringElements_](#)
Maximum number of string elements.
- int [numberStringElements_](#)
Number of string elements.
- char ** [stringElements_](#)
String elements.

Friends

- `void CoinMpsIOUnitTest (const std::string &mpsDir)`
A function that tests the methods in the `CoinMpsIO` class.

9.52.1 Detailed Description

MPS IO Interface.

This class can be used to read in mps files without a solver. After reading the file, the `CoinMpsIO` object contains all relevant data, which may be more than a particular `OsiSolverInterface` allows for. Items may be deleted to allow for flexibility of data storage.

The implementation makes the `CoinMpsIO` object look very like a dummy solver, as the same conventions are used.

Definition at line 329 of file `CoinMpsIO.hpp`.

9.52.2 Constructor & Destructor Documentation

9.52.2.1 `CoinMpsIO::CoinMpsIO ()`

Default Constructor.

9.52.2.2 `CoinMpsIO::CoinMpsIO (const CoinMpsIO &)`

Copy constructor.

9.52.2.3 `CoinMpsIO::~CoinMpsIO ()`

Destructor.

9.52.3 Member Function Documentation

9.52.3.1 `int CoinMpsIO::getNumCols () const`

Get number of columns.

9.52.3.2 `int CoinMpsIO::getNumRows () const`

Get number of rows.

9.52.3.3 `int CoinMpsIO::getNumElements () const`

Get number of nonzero elements.

9.52.3.4 `const double* CoinMpsIO::getColLower () const`

Get pointer to array[`getNumCols()`] of column lower bounds.

9.52.3.5 `const double* CoinMpsIO::getColUpper () const`

Get pointer to array[`getNumCols()`] of column upper bounds.

9.52.3.6 `const char* CoinMpsIO::getRowSense () const`

Get pointer to array[`getNumRows()`] of constraint senses.

- 'L': \leq constraint
- 'E': = constraint
- 'G': \geq constraint
- 'R': ranged constraint
- 'N': free constraint

9.52.3.7 `const double* CoinMpsIO::getRightHandSide () const`

Get pointer to array[[getNumRows\(\)](#)] of constraint right-hand sides.

Given constraints with upper (rowupper) and/or lower (rowlower) bounds, the constraint right-hand side (rhs) is set as

- if `rowsense()[i] == 'L'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'G'` then `rhs()[i] == rowlower()[i]`
- if `rowsense()[i] == 'R'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'N'` then `rhs()[i] == 0.0`

9.52.3.8 `const double* CoinMpsIO::getRowRange () const`

Get pointer to array[[getNumRows\(\)](#)] of row ranges.

Given constraints with upper (rowupper) and/or lower (rowlower) bounds, the constraint range (rowrange) is set as

- if `rowsense()[i] == 'R'` then `rowrange()[i] == rowupper()[i] - rowlower()[i]`
- if `rowsense()[i] != 'R'` then `rowrange()[i]` is 0.0

Put another way, only range constraints have a nontrivial value for rowrange.

9.52.3.9 `const double* CoinMpsIO::getRowLower () const`

Get pointer to array[[getNumRows\(\)](#)] of row lower bounds.

9.52.3.10 `const double* CoinMpsIO::getRowUpper () const`

Get pointer to array[[getNumRows\(\)](#)] of row upper bounds.

9.52.3.11 `const double* CoinMpsIO::getObjCoefficients () const`

Get pointer to array[[getNumCols\(\)](#)] of objective function coefficients.

9.52.3.12 `const CoinPackedMatrix* CoinMpsIO::getMatrixByRow () const`

Get pointer to row-wise copy of the coefficient matrix.

9.52.3.13 `const CoinPackedMatrix* CoinMpsIO::getMatrixByCol () const`

Get pointer to column-wise copy of the coefficient matrix.

9.52.3.14 `bool CoinMpsIO::isContinuous (int colNumber) const`

Return true if column is a continuous variable.

9.52.3.15 bool CoinMpsIO::isInteger (int *columnNumber*) const

Return true if a column is an integer variable.

Note: This function returns true if the the column is a binary or general integer variable.

9.52.3.16 const char* CoinMpsIO::integerColumns () const

Returns array[getNumCols()] specifying if a variable is integer.

At present, simply coded as zero (continuous) and non-zero (integer) May be extended at a later date.

9.52.3.17 const char* CoinMpsIO::rowName (int *index*) const

Returns the row name for the specified index.

Returns 0 if the index is out of range.

9.52.3.18 const char* CoinMpsIO::columnName (int *index*) const

Returns the column name for the specified index.

Returns 0 if the index is out of range.

9.52.3.19 int CoinMpsIO::rowIndex (const char * *name*) const

Returns the index for the specified row name.

Returns -1 if the name is not found. Returns numberOfRows for the objective row and > numberOfRows for dropped free rows.

9.52.3.20 int CoinMpsIO::columnIndex (const char * *name*) const

Returns the index for the specified column name.

Returns -1 if the name is not found.

9.52.3.21 double CoinMpsIO::objectiveOffset () const

Returns the (constant) objective offset.

This is the RHS entry for the objective row

9.52.3.22 void CoinMpsIO::setObjectiveOffset (double *value*) [inline]

Set objective offset.

Definition at line 463 of file CoinMpsIO.hpp.

9.52.3.23 const char* CoinMpsIO::getProblemName () const

Return the problem name.

9.52.3.24 const char* CoinMpsIO::getObjectiveName () const

Return the objective name.

9.52.3.25 const char* CoinMpsIO::getRhsName () const

Return the RHS vector name.

9.52.3.26 `const char* CoinMpsIO::getRangeName () const`

Return the range vector name.

9.52.3.27 `const char* CoinMpsIO::getBoundName () const`

Return the bound vector name.

9.52.3.28 `int CoinMpsIO::numberStringElements () const [inline]`

Number of string elements.

Definition at line 481 of file CoinMpsIO.hpp.

9.52.3.29 `const char* CoinMpsIO::stringElement (int i) const [inline]`

String element.

Definition at line 484 of file CoinMpsIO.hpp.

9.52.3.30 `void CoinMpsIO::setMpsData (const CoinPackedMatrix & m, const double infinity, const double * collb, const double * colub, const double * obj, const char * integrality, const double * rowlb, const double * rowub, char const *const *const colnames, char const *const *const rownames)`

Set the problem data.

9.52.3.31 `void CoinMpsIO::setMpsData (const CoinPackedMatrix & m, const double infinity, const double * collb, const double * colub, const double * obj, const char * integrality, const double * rowlb, const double * rowub, const std::vector< std::string > & colnames, const std::vector< std::string > & rownames)`

9.52.3.32 `void CoinMpsIO::setMpsData (const CoinPackedMatrix & m, const double infinity, const double * collb, const double * colub, const double * obj, const char * integrality, const char * rowsen, const double * rowrhs, const double * rowrng, char const *const *const colnames, char const *const *const rownames)`

9.52.3.33 `void CoinMpsIO::setMpsData (const CoinPackedMatrix & m, const double infinity, const double * collb, const double * colub, const double * obj, const char * integrality, const char * rowsen, const double * rowrhs, const double * rowrng, const std::vector< std::string > & colnames, const std::vector< std::string > & rownames)`

9.52.3.34 `void CoinMpsIO::copyInIntegerInformation (const char * integerInformation)`

Pass in an array[getNumCols()] specifying if a variable is integer.

At present, simply coded as zero (continuous) and non-zero (integer) May be extended at a later date.

9.52.3.35 `void CoinMpsIO::setProblemName (const char * name)`

Set problem name.

9.52.3.36 `void CoinMpsIO::setObjectiveName (const char * name)`

Set objective name.

9.52.3.37 `void CoinMpsIO::setInfinity (double value)`

Set infinity.

9.52.3.38 `double CoinMpsIO::getInfinity () const`

Get infinity.

9.52.3.39 void CoinMpsIO::setDefaultBound (int *value*)

Set default upper bound for integer variables.

9.52.3.40 int CoinMpsIO::getDefaultBound () const

Get default upper bound for integer variables.

9.52.3.41 int CoinMpsIO::allowStringElements () const [inline]

Whether to allow string elements.

Definition at line 556 of file CoinMpsIO.hpp.

9.52.3.42 void CoinMpsIO::setAllowStringElements (int *yesNo*) [inline]

Whether to allow string elements (0 no, 1 yes, 2 yes and try flip)

Definition at line 559 of file CoinMpsIO.hpp.

9.52.3.43 double CoinMpsIO::getSmallElementValue () const [inline]

Small element value - elements less than this set to zero on input default is 1.0e-14.

Definition at line 563 of file CoinMpsIO.hpp.

9.52.3.44 void CoinMpsIO::setSmallElementValue (double *value*) [inline]

Definition at line 565 of file CoinMpsIO.hpp.

9.52.3.45 void CoinMpsIO::setFileName (const char * *name*)

Set the current file name for the [CoinMpsIO](#) object.

9.52.3.46 const char* CoinMpsIO::getFileName () const

Get the current file name for the [CoinMpsIO](#) object.

9.52.3.47 int CoinMpsIO::readMps (const char * *filename*, const char * *extension* = "mps")

Read a problem in MPS format from the given filename.

Use "stdin" or "-" to read from stdin.

9.52.3.48 int CoinMpsIO::readMps (const char * *filename*, const char * *extension*, int & *numberSets*, CoinSet **& *sets*)

Read a problem in MPS format from the given filename.

Use "stdin" or "-" to read from stdin. But do sets as well

9.52.3.49 int CoinMpsIO::readMps ()

Read a problem in MPS format from a previously opened file.

More precisely, read a problem using a [CoinMpsCardReader](#) object already associated with this [CoinMpsIO](#) object.

Todo Provide an interface that will allow a client to associate a [CoinMpsCardReader](#) object with a [CoinMpsIO](#) object by setting the `cardReader_` field.

9.52.3.50 `int CoinMpsIO::readMps (int & numberSets, CoinSet **& sets)`

and

9.52.3.51 `int CoinMpsIO::readBasis (const char * filename, const char * extension, double * solution, unsigned char * rowStatus, unsigned char * columnStatus, const std::vector< std::string > & colnames, int numberColumns, const std::vector< std::string > & rownames, int numberRows)`

Read a basis in MPS format from the given filename.

If VALUES on NAME card and solution not NULL fills in solution status values as for [CoinWarmStartBasis](#) (but one per char) -1 file error, 0 normal, 1 has solution values

Use "stdin" or "-" to read from stdin.

If sizes of names incorrect - read without names

9.52.3.52 `int CoinMpsIO::readGms (const char * filename, const char * extension = "gms", bool convertObjective = false)`

Read a problem in GAMS format from the given filename.

Use "stdin" or "-" to read from stdin. if convertObjective then massages objective column

9.52.3.53 `int CoinMpsIO::readGms (const char * filename, const char * extension, int & numberSets, CoinSet **& sets)`

Read a problem in GAMS format from the given filename.

Use "stdin" or "-" to read from stdin. But do sets as well

9.52.3.54 `int CoinMpsIO::readGms (int & numberSets, CoinSet **& sets)`

Read a problem in GAMS format from a previously opened file.

More precisely, read a problem using a [CoinMpsCardReader](#) object already associated with this [CoinMpsIO](#) object and

9.52.3.55 `int CoinMpsIO::readGMPL (const char * modelName, const char * dataName = NULL, bool keepNames = false)`

Read a problem in GMPL (subset of AMPL) format from the given filenames.

9.52.3.56 `int CoinMpsIO::writeMps (const char * filename, int compression = 0, int formatType = 0, int numberAcross = 2, CoinPackedMatrix * quadratic = NULL, int numberSOS = 0, const CoinSet * setInfo = NULL) const`

Write the problem in MPS format to a file with the given filename.

Parameters

<i>compression</i>	<p>can be set to three values to indicate what kind of file should be written</p> <ul style="list-style-type: none"> • 0: plain text (default) • 1: gzip compressed (.gz is appended to <i>filename</i>) • 2: bzip2 compressed (.bz2 is appended to <i>filename</i>) (TODO) <p>If the library was not compiled with the requested compression then writeMps falls back to writing a plain text file.</p>
--------------------	---

<i>formatType</i>	specifies the precision to used for values in the MPS file <ul style="list-style-type: none"> • 0: normal precision (default) • 1: extra accuracy • 2: IEEE hex
<i>numberAcross</i>	specifies whether 1 or 2 (default) values should be specified on every data line in the MPS file.
<i>quadratic</i>	specifies quadratic objective to be output

9.52.3.57 `const CoinMpsCardReader* CoinMpsIO::reader () const [inline]`

Return card reader object so can see what last card was e.g. QUADOBJ.

Definition at line 694 of file CoinMpsIO.hpp.

9.52.3.58 `int CoinMpsIO::readQuadraticMps (const char * filename, int *& columnStart, int *& column, double *& elements, int checkSymmetry)`

Read in a quadratic objective from the given filename.

If filename is NULL (or the same as the currently open file) then reading continues from the current file. If not, the file is closed and the specified file is opened.

Code should be added to general MPS reader to read this if QSECTION Data is assumed to be Q and objective is $c + 1/2 \times T \times Q \times N$ no assumption is made for symmetry, positive definite, etc. No check is made for duplicates or non-triangular if `checkSymmetry==0`. If 1 checks lower triangular (so off diagonal should be $2 \times Q$) if 2 makes lower triangular and assumes full Q (but adds off diagonals)

Arrays should be deleted by `delete []`

Returns number of errors:

- -1: bad file
- -2: no Quadratic section
- -3: an empty section
- +n: then matching errors etc (symmetry forced)
- -4: no matching errors but fails triangular test (triangularity forced)

`columnStart` is `numberColumns+1` long, others `numberNonZeros`

9.52.3.59 `int CoinMpsIO::readConicMps (const char * filename, int *& columnStart, int *& column, int *& coneType, int & numberCones)`

Read in a list of cones from the given filename.

If filename is NULL (or the same as the currently open file) then reading continues from the current file. If not, the file is closed and the specified file is opened.

Code should be added to general MPS reader to read this if CSECTION No checking is done that in unique cone

Arrays should be deleted by `delete []`

Returns number of errors, -1 bad file, -2 no conic section, -3 empty section

`columnStart` is `numberCones+1` long, other number of columns in matrix

`coneType` is 1 for QUAD, 2 for RQUAD (`numberCones` long)

9.52.3.60 `void CoinMpsIO::setConvertObjective (bool trueFalse) [inline]`

Set whether to move objective from matrix.

Definition at line 750 of file CoinMpsIO.hpp.

9.52.3.61 `int CoinMpsIO::copyStringElements (const CoinModel * model)`

copies in strings from a [CoinModel](#) - returns number

9.52.3.62 `CoinMpsIO& CoinMpsIO::operator= (const CoinMpsIO & rhs)`

Assignment operator.

9.52.3.63 `void CoinMpsIO::passInMessageHandler (CoinMessageHandler * handler)`

Pass in Message handler.

Supply a custom message handler. It will not be destroyed when the [CoinMpsIO](#) object is destroyed.

9.52.3.64 `void CoinMpsIO::newLanguage (CoinMessages::Language language)`

Set the language for messages.

9.52.3.65 `void CoinMpsIO::setLanguage (CoinMessages::Language language) [inline]`

Set the language for messages.

Definition at line 785 of file CoinMpsIO.hpp.

9.52.3.66 `CoinMessageHandler* CoinMpsIO::messageHandler () const [inline]`

Return the message handler.

Definition at line 788 of file CoinMpsIO.hpp.

9.52.3.67 `CoinMessages CoinMpsIO::messages () [inline]`

Return the messages.

Definition at line 791 of file CoinMpsIO.hpp.

9.52.3.68 `CoinMessages* CoinMpsIO::messagesPointer () [inline]`

Return the messages pointer.

Definition at line 793 of file CoinMpsIO.hpp.

9.52.3.69 `void CoinMpsIO::releaseRedundantInformation ()`

Release all information which can be re-calculated.

E.g., row sense, copies of rows, hash tables for names.

9.52.3.70 `void CoinMpsIO::releaseRowInformation ()`

Release all row information (lower, upper)

9.52.3.71 `void CoinMpsIO::releaseColumnInformation ()`

Release all column information (lower, upper, objective)

9.52.3.72 **void** CoinMpsIO::releaseIntegerInformation ()

Release integer information.

9.52.3.73 **void** CoinMpsIO::releaseRowNames ()

Release row names.

9.52.3.74 **void** CoinMpsIO::releaseColumnNames ()

Release column names.

9.52.3.75 **void** CoinMpsIO::releaseMatrixInformation ()

Release matrix information.

9.52.3.76 **void** CoinMpsIO::setMpsDataWithoutRowAndColNames (const CoinPackedMatrix & *m*, const double *infinity*, const double * *collb*, const double * *colub*, const double * *obj*, const char * *integrality*, const double * *rowlb*, const double * *rowub*) [protected]

Utility method used several times to implement public methods.

9.52.3.77 **void** CoinMpsIO::setMpsDataColAndRowNames (const std::vector< std::string > & *colnames*, const std::vector< std::string > & *rownames*) [protected]

9.52.3.78 **void** CoinMpsIO::setMpsDataColAndRowNames (char const *const *const *colnames*, char const *const *const *rownames*) [protected]

9.52.3.79 **void** CoinMpsIO::gutsOfDestructor () [protected]

Does the heavy lifting for destruct and assignment.

9.52.3.80 **void** CoinMpsIO::gutsOfCopy (const CoinMpsIO &) [protected]

Does the heavy lifting for copy and assignment.

9.52.3.81 **void** CoinMpsIO::freeAll () [protected]

Clears problem data from the [CoinMpsIO](#) object.

9.52.3.82 **void** CoinMpsIO::convertBoundToSense (const double *lower*, const double *upper*, char & *sense*, double & *right*, double & *range*) const [inline], [protected]

A quick inlined function to convert from lb/ub style constraint definition to sense/rhs/range style.

9.52.3.83 **void** CoinMpsIO::convertSenseToBound (const char *sense*, const double *right*, const double *range*, double & *lower*, double & *upper*) const [inline], [protected]

A quick inlined function to convert from sense/rhs/range style constraint definition to lb/ub style.

9.52.3.84 **int** CoinMpsIO::dealWithFileName (const char * *filename*, const char * *extension*, CoinFileInput *& *input*) [protected]

Deal with a filename.

As the name says. Returns +1 if the file name is new, 0 if it's the same as before (i.e., matches *fileName_*), and -1 if there's an error and the file can't be opened. Handles automatic append of .gz suffix when compiled with libz.

Todo Add automatic append of .bz2 suffix when compiled with libbz.

9.52.3.85 `void CoinMpsIO::addString (int iRow, int iColumn, const char * value)` [protected]

Add string to list $iRow == \text{numberRows}$ is objective, $nr+1$ is lo, $nr+2$ is up $iColumn == nc$ is rhs (can't cope with ranges at present)

9.52.3.86 `void CoinMpsIO::decodeString (int iString, int & iRow, int & iColumn, const char *& value) const` [protected]

Decode string.

9.52.3.87 `void CoinMpsIO::startHash (char ** names, const int number, int section)` [protected]

Creates hash list for names (section = 0 for rows, 1 columns)

9.52.3.88 `void CoinMpsIO::startHash (int section) const` [protected]

This one does it when names are already in.

9.52.3.89 `void CoinMpsIO::stopHash (int section)` [protected]

Deletes hash storage.

9.52.3.90 `int CoinMpsIO::findHash (const char * name, int section) const` [protected]

Finds match using hash, -1 not found.

9.52.4 Friends And Related Function Documentation

9.52.4.1 `void CoinMpsIOUnitTest (const std::string & mpsDir)` [friend]

A function that tests the methods in the [CoinMpsIO](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging. Also, if this method is compiled with optimization, the compilation takes 10-15 minutes and the machine pages (has 256M core memory!)

9.52.5 Member Data Documentation

9.52.5.1 `char* CoinMpsIO::problemName_` [protected]

Problem name.

Definition at line 916 of file `CoinMpsIO.hpp`.

9.52.5.2 `char* CoinMpsIO::objectiveName_` [protected]

Objective row name.

Definition at line 919 of file `CoinMpsIO.hpp`.

9.52.5.3 `char* CoinMpsIO::rhsName_` [protected]

Right-hand side vector name.

Definition at line 922 of file `CoinMpsIO.hpp`.

9.52.5.4 `char* CoinMpslO::rangeName_` `[protected]`

Range vector name.

Definition at line 925 of file CoinMpslO.hpp.

9.52.5.5 `char* CoinMpslO::boundName_` `[protected]`

Bounds vector name.

Definition at line 928 of file CoinMpslO.hpp.

9.52.5.6 `int CoinMpslO::numberOfRows_` `[protected]`

Number of rows.

Definition at line 931 of file CoinMpslO.hpp.

9.52.5.7 `int CoinMpslO::numberOfColumns_` `[protected]`

Number of columns.

Definition at line 934 of file CoinMpslO.hpp.

9.52.5.8 `CoinBigIndex CoinMpslO::numberOfElements_` `[protected]`

Number of coefficients.

Definition at line 937 of file CoinMpslO.hpp.

9.52.5.9 `char* CoinMpslO::rowSense_` `[mutable], [protected]`

Pointer to dense vector of row sense indicators.

Definition at line 940 of file CoinMpslO.hpp.

9.52.5.10 `double* CoinMpslO::rhs_` `[mutable], [protected]`

Pointer to dense vector of row right-hand side values.

Definition at line 943 of file CoinMpslO.hpp.

9.52.5.11 `double* CoinMpslO::rowRange_` `[mutable], [protected]`

Pointer to dense vector of slack variable upper bounds for range constraints (undefined for non-range rows)

Definition at line 948 of file CoinMpslO.hpp.

9.52.5.12 `CoinPackedMatrix* CoinMpslO::matrixByRow_` `[mutable], [protected]`

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 951 of file CoinMpslO.hpp.

9.52.5.13 `CoinPackedMatrix* CoinMpslO::matrixByColumn_` `[protected]`

Pointer to column-wise copy of problem matrix coefficients.

Definition at line 954 of file CoinMpslO.hpp.

9.52.5.14 `double* CoinMpsIO::rowlower_` `[protected]`

Pointer to dense vector of row lower bounds.

Definition at line 957 of file CoinMpsIO.hpp.

9.52.5.15 `double* CoinMpsIO::rowupper_` `[protected]`

Pointer to dense vector of row upper bounds.

Definition at line 960 of file CoinMpsIO.hpp.

9.52.5.16 `double* CoinMpsIO::collower_` `[protected]`

Pointer to dense vector of column lower bounds.

Definition at line 963 of file CoinMpsIO.hpp.

9.52.5.17 `double* CoinMpsIO::colupper_` `[protected]`

Pointer to dense vector of column upper bounds.

Definition at line 966 of file CoinMpsIO.hpp.

9.52.5.18 `double* CoinMpsIO::objective_` `[protected]`

Pointer to dense vector of objective coefficients.

Definition at line 969 of file CoinMpsIO.hpp.

9.52.5.19 `double CoinMpsIO::objectiveOffset_` `[protected]`

Constant offset for objective value (i.e., RHS value for OBJ row)

Definition at line 972 of file CoinMpsIO.hpp.

9.52.5.20 `char* CoinMpsIO::integerType_` `[protected]`

Pointer to dense vector specifying if a variable is continuous (0) or integer (1).

Definition at line 978 of file CoinMpsIO.hpp.

9.52.5.21 `char** CoinMpsIO::names_[2]` `[protected]`

Row and column names Linked to hash table sections (0 - row names, 1 column names)

Definition at line 983 of file CoinMpsIO.hpp.

9.52.5.22 `char* CoinMpsIO::fileName_` `[protected]`

Current file name.

Definition at line 989 of file CoinMpsIO.hpp.

9.52.5.23 `int CoinMpsIO::numberHash_[2]` `[protected]`

Number of entries in a hash table section.

Definition at line 992 of file CoinMpsIO.hpp.

9.52.5.24 CoinHashLink* CoinMpsIO::hash_[2] [mutable], [protected]

Hash tables (two sections, 0 - row names, 1 - column names)

Definition at line 995 of file CoinMpsIO.hpp.

9.52.5.25 int CoinMpsIO::defaultBound_ [protected]

Upper bound when no bounds for integers.

Definition at line 1001 of file CoinMpsIO.hpp.

9.52.5.26 double CoinMpsIO::infinity_ [protected]

Value to use for infinity.

Definition at line 1004 of file CoinMpsIO.hpp.

9.52.5.27 double CoinMpsIO::smallElement_ [protected]

Small element value.

Definition at line 1006 of file CoinMpsIO.hpp.

9.52.5.28 CoinMessageHandler* CoinMpsIO::handler_ [protected]

Message handler.

Definition at line 1009 of file CoinMpsIO.hpp.

9.52.5.29 bool CoinMpsIO::defaultHandler_ [protected]

Flag to say if the message handler is the default handler.

If true, the handler will be destroyed when the [CoinMpsIO](#) object is destroyed; if false, it will not be destroyed.

Definition at line 1015 of file CoinMpsIO.hpp.

9.52.5.30 CoinMessages CoinMpsIO::messages_ [protected]

Messages.

Definition at line 1017 of file CoinMpsIO.hpp.

9.52.5.31 CoinMpsCardReader* CoinMpsIO::cardReader_ [protected]

Card reader.

Definition at line 1019 of file CoinMpsIO.hpp.

9.52.5.32 bool CoinMpsIO::convertObjective_ [protected]

If .gms file should it be massaged to move objective.

Definition at line 1021 of file CoinMpsIO.hpp.

9.52.5.33 int CoinMpsIO::allowStringElements_ [protected]

Whether to allow string elements.

Definition at line 1023 of file CoinMpsIO.hpp.

9.52.5.34 int CoinMpsIO::maximumStringElements_ [protected]

Maximum number of string elements.

Definition at line 1025 of file CoinMpsIO.hpp.

9.52.5.35 int CoinMpsIO::numberStringElements_ [protected]

Number of string elements.

Definition at line 1027 of file CoinMpsIO.hpp.

9.52.5.36 char** CoinMpsIO::stringElements_ [protected]

String elements.

Definition at line 1029 of file CoinMpsIO.hpp.

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinMpsIO.hpp](#)

9.53 CoinOneMessage Class Reference

Class for one massaged message.

```
#include <CoinMessageHandler.hpp>
```

Public Member Functions

Constructors etc

- [CoinOneMessage](#) ()
Default constructor.
- [CoinOneMessage](#) (int [externalNumber](#), char [detail](#), const char *[message](#))
Normal constructor.
- [~CoinOneMessage](#) ()
Destructor.
- [CoinOneMessage](#) (const [CoinOneMessage](#) &)
The copy constructor.
- [CoinOneMessage](#) & [operator=](#) (const [CoinOneMessage](#) &)
assignment operator.

Useful stuff

- [void replaceMessage](#) (const char *[message](#))
Replace message text (e.g., text in a different language)

Get and set methods

- int [externalNumber](#) () const
Get message ID number.
- [void setExternalNumber](#) (int number)
Set message ID number.
- char [severity](#) () const
Severity.

- `void setDetail (int level)`
Set detail level.
- `int detail () const`
Get detail level.
- `char * message () const`
Return the message text.

Public Attributes

member data

- `int externalNumber_`
number to print out (also determines severity)
- `char detail_`
Will only print if detail matches.
- `char severity_`
Severity.
- `char message_ [400]`
Messages (in correct language) (not all 400 may exist)

9.53.1 Detailed Description

Class for one massaged message.

A message consists of a text string with formatting codes (`message_`), an integer identifier (`externalNumber_`) which also determines the severity level (`severity_`) of the message, and a detail (logging) level (`detail_`).

`CoinOneMessage` is just a container to hold this information. The interpretation is set by `CoinMessageHandler`, which see.

Definition at line 58 of file `CoinMessageHandler.hpp`.

9.53.2 Constructor & Destructor Documentation

9.53.2.1 `CoinOneMessage::CoinOneMessage ()`

Default constructor.

9.53.2.2 `CoinOneMessage::CoinOneMessage (int externalNumber, char detail, const char * message)`

Normal constructor.

9.53.2.3 `CoinOneMessage::~~CoinOneMessage ()`

Destructor.

9.53.2.4 `CoinOneMessage::CoinOneMessage (const CoinOneMessage &)`

The copy constructor.

9.53.3 Member Function Documentation

9.53.3.1 `CoinOneMessage& CoinOneMessage::operator= (const CoinOneMessage &)`

assignment operator.

9.53.3.2 void CoinOneMessage::replaceMessage (const char * *message*)

Replace message text (e.g., text in a different language)

9.53.3.3 int CoinOneMessage::externalNumber () const [inline]

Get message ID number.

Definition at line 85 of file CoinMessageHandler.hpp.

9.53.3.4 void CoinOneMessage::setExternalNumber (int *number*) [inline]

Set message ID number.

In the default [CoinMessageHandler](#), this number is printed in the message prefix and is used to determine the message severity level.

Definition at line 92 of file CoinMessageHandler.hpp.

9.53.3.5 char CoinOneMessage::severity () const [inline]

Severity.

Definition at line 95 of file CoinMessageHandler.hpp.

9.53.3.6 void CoinOneMessage::setDetail (int *level*) [inline]

Set detail level.

Definition at line 98 of file CoinMessageHandler.hpp.

9.53.3.7 int CoinOneMessage::detail () const [inline]

Get detail level.

Definition at line 101 of file CoinMessageHandler.hpp.

9.53.3.8 char* CoinOneMessage::message () const [inline]

Return the message text.

Definition at line 104 of file CoinMessageHandler.hpp.

9.53.4 Member Data Documentation

9.53.4.1 int CoinOneMessage::externalNumber_

number to print out (also determines severity)

Definition at line 111 of file CoinMessageHandler.hpp.

9.53.4.2 char CoinOneMessage::detail_

Will only print if detail matches.

Definition at line 113 of file CoinMessageHandler.hpp.

9.53.4.3 char CoinOneMessage::severity_

Severity.

Definition at line 115 of file CoinMessageHandler.hpp.

9.53.4.4 `char CoinOneMessage::message_[400] [mutable]`

Messages (in correct language) (not all 400 may exist)

Definition at line 117 of file CoinMessageHandler.hpp.

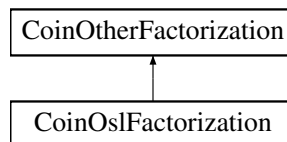
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/CoinUtils/src/CoinMessageHandler.hpp

9.54 CoinOslFactorization Class Reference

```
#include <CoinOslFactorization.hpp>
```

Inheritance diagram for CoinOslFactorization:



Public Member Functions

- `void gutsOfDestructor` (bool clearFact=true)
The real work of desstructor.
- `void gutsOfInitialize` (bool zapFact=true)
The real work of constructor.
- `void gutsOfCopy` (const `CoinOslFactorization` &other)
The real work of copy.

Constructors and destructor and copy

- `CoinOslFactorization` ()
Default constructor.
- `CoinOslFactorization` (const `CoinOslFactorization` &other)
Copy constructor.
- `virtual ~CoinOslFactorization` ()
Destructor.
- `CoinOslFactorization & operator=` (const `CoinOslFactorization` &other)
= copy
- `virtual CoinOtherFactorization * clone` () const
Clone.

Do factorization - public

- `virtual void getAreas` (int `numberRows`, int `numberColumns`, `CoinBigIndex` `maximumL`, `CoinBigIndex` `maximumU`)
Gets space for a factorization.
- `virtual void preProcess` ()
PreProcesses column ordered copy of basis.

- virtual int `factor` ()
Does most of factorization returning status 0 - OK -99 - needs more memory -1 - singular - use numberGoodColumns and redo.
- virtual void `postProcess` (const int *sequence, int *pivotVariable)
Does post processing on valid factorization - putting variables on correct rows.
- virtual void `makeNonSingular` (int *sequence, int numberColumns)
Makes a non-singular basis by replacing variables.
- int `factorize` (const CoinPackedMatrix &matrix, int rowsBasic[], int columnsBasic[], double areaFactor=0.0)
When part of LP - given by basic variables.

general stuff such as number of elements

- virtual int `numberElements` () const
Total number of elements in factorization.
- virtual CoinFactorizationDouble * `elements` () const
Returns array to put basis elements in.
- virtual int * `pivotRow` () const
Returns pivot row.
- virtual CoinFactorizationDouble * `workArea` () const
Returns work area.
- virtual int * `intWorkArea` () const
Returns int work area.
- virtual int * `numberInRow` () const
Number of entries in each row.
- virtual int * `numberInColumn` () const
Number of entries in each column.
- virtual CoinBigIndex * `starts` () const
Returns array to put basis starts in.
- virtual int * `permuteBack` () const
Returns permute back.
- virtual bool `wantsTableauColumn` () const
Returns true if wants tableauColumn in replaceColumn.
- virtual void `setUsefullInformation` (const int *info, int whereFrom)
Useful information for factorization 0 - iteration number whereFrom is 0 for factorize and 1 for replaceColumn.
- virtual void `maximumPivots` (int value)
Set maximum pivots.
- double `maximumCoefficient` () const
Returns maximum absolute value in factorization.
- double `conditionNumber` () const
Condition number - product of pivots after factorization.
- virtual void `clearArrays` ()
Get rid of all memory.

rank one updates which do exist

- virtual int `replaceColumn` (CoinIndexedVector *regionSparse, int pivotRow, double pivotCheck, bool checkBeforeModifying=false, double acceptablePivot=1.0e-8)
Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If checkBeforeModifying is true will do all accuracy checks before modifying factorization.

various uses of factorization (return code number elements)

which user may want to know about

- virtual int `updateColumnFT` (CoinIndexedVector *regionSparse, CoinIndexedVector *regionSparse2, bool noPermute=false)

Updates one column (FTRAN) from regionSparse2 Tries to do FT update number returned is negative if no room regionSparse starts as zero and is zero at end.

- virtual int [updateColumn](#) ([CoinIndexedVector](#) *regionSparse, [CoinIndexedVector](#) *regionSparse2, bool noPermute=false) const

This version has same effect as above with FTUpdate==false so number returned is always ≥ 0 .

- virtual int [updateTwoColumnsFT](#) ([CoinIndexedVector](#) *regionSparse1, [CoinIndexedVector](#) *regionSparse2, [CoinIndexedVector](#) *regionSparse3, bool noPermute=false)

does FTRAN on two columns

- virtual int [updateColumnTranspose](#) ([CoinIndexedVector](#) *regionSparse, [CoinIndexedVector](#) *regionSparse2) const

Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if regionSparse2 packed on input - will be packed on output.

various uses of factorization

*** Below this user may not want to know about

which user may not want to know about (left over from my LP code)

- virtual int * [indices](#) () const
Get rid of all memory.
- virtual int * [permute](#) () const
Returns permute in.

Protected Member Functions

- int [checkPivot](#) (double saveFromU, double oldPivot) const
Returns accuracy status of replaceColumn returns 0=OK, 1=Probably OK, 2=singular.

Protected Attributes

data

- [EKKfactinfo factInfo_](#)
Osl factorization data.

Friends

- void [CoinOslFactorizationUnitTest](#) (const std::string &mpsDir)

9.54.1 Detailed Description

Definition at line 106 of file CoinOslFactorization.hpp.

9.54.2 Constructor & Destructor Documentation

9.54.2.1 CoinOslFactorization::CoinOslFactorization ()

Default constructor.

9.54.2.2 CoinOslFactorization::CoinOslFactorization (const CoinOslFactorization & other)

Copy constructor.

9.54.2.3 `virtual CoinOslFactorization::~~CoinOslFactorization () [virtual]`

Destructor.

9.54.3 Member Function Documentation

9.54.3.1 `CoinOslFactorization& CoinOslFactorization::operator= (const CoinOslFactorization & other)`

= copy

9.54.3.2 `virtual CoinOtherFactorization* CoinOslFactorization::clone () const [virtual]`

Clone.

Implements [CoinOtherFactorization](#).

9.54.3.3 `virtual void CoinOslFactorization::getAreas (int numberOfRows, int numberOfColumns, CoinBigIndex maximumL, CoinBigIndex maximumU) [virtual]`

Gets space for a factorization.

Implements [CoinOtherFactorization](#).

9.54.3.4 `virtual void CoinOslFactorization::preProcess () [virtual]`

PreProcesses column ordered copy of basis.

Implements [CoinOtherFactorization](#).

9.54.3.5 `virtual int CoinOslFactorization::factor () [virtual]`

Does most of factorization returning status 0 - OK -99 - needs more memory -1 - singular - use numberGoodColumns and redo.

Implements [CoinOtherFactorization](#).

9.54.3.6 `virtual void CoinOslFactorization::postProcess (const int * sequence, int * pivotVariable) [virtual]`

Does post processing on valid factorization - putting variables on correct rows.

Implements [CoinOtherFactorization](#).

9.54.3.7 `virtual void CoinOslFactorization::makeNonSingular (int * sequence, int numberOfColumns) [virtual]`

Makes a non-singular basis by replacing variables.

Implements [CoinOtherFactorization](#).

9.54.3.8 `int CoinOslFactorization::factorize (const CoinPackedMatrix & matrix, int rowsBasic[], int columnsBasic[], double areaFactor = 0.0)`

When part of LP - given by basic variables.

Actually does factorization. Arrays passed in have non negative value to say basic. If status is okay, basic variables have pivot row - this is only needed If status is singular, then basic variables have pivot row and ones thrown out have -1 returns 0 -okay, -1 singular, -2 too many in basis, -99 memory

9.54.3.9 `virtual int CoinOslFactorization::numberElements () const [inline],[virtual]`

Total number of elements in factorization.

Implements [CoinOtherFactorization](#).

Definition at line 161 of file `CoinOslFactorization.hpp`.

9.54.3.10 `virtual CoinFactorizationDouble* CoinOslFactorization::elements () const [virtual]`

Returns array to put basis elements in.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.11 `virtual int* CoinOslFactorization::pivotRow () const [virtual]`

Returns pivot row.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.12 `virtual CoinFactorizationDouble* CoinOslFactorization::workArea () const [virtual]`

Returns work area.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.13 `virtual int* CoinOslFactorization::intWorkArea () const [virtual]`

Returns int work area.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.14 `virtual int* CoinOslFactorization::numberInRow () const [virtual]`

Number of entries in each row.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.15 `virtual int* CoinOslFactorization::numberInColumn () const [virtual]`

Number of entries in each column.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.16 `virtual CoinBigIndex* CoinOslFactorization::starts () const [virtual]`

Returns array to put basis starts in.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.17 `virtual int* CoinOslFactorization::permuteBack () const [virtual]`

Returns permute back.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.18 `virtual bool CoinOslFactorization::wantsTableauColumn () const [virtual]`

Returns true if wants tableauColumn in replaceColumn.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.19 `virtual void CoinOslFactorization::setUsefullInformation (const int * info, int whereFrom)` [virtual]

Useful information for factorization 0 - iteration number whereFrom is 0 for factorize and 1 for replaceColumn.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.20 `virtual void CoinOslFactorization::maximumPivots (int value)` [virtual]

Set maximum pivots.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.21 `double CoinOslFactorization::maximumCoefficient () const`

Returns maximum absolute value in factorization.

9.54.3.22 `double CoinOslFactorization::conditionNumber () const`

Condition number - product of pivots after factorization.

9.54.3.23 `virtual void CoinOslFactorization::clearArrays ()` [virtual]

Get rid of all memory.

Reimplemented from [CoinOtherFactorization](#).

9.54.3.24 `virtual int CoinOslFactorization::replaceColumn (CoinIndexedVector * regionSparse, int pivotRow, double pivotCheck, bool checkBeforeModifying = false, double acceptablePivot = 1.0e-8)` [virtual]

Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If checkBeforeModifying is true will do all accuracy checks before modifying factorization.

Whether to set this depends on speed considerations. You could just do this on first iteration after factorization and thereafter re-factorize partial update already in U

Implements [CoinOtherFactorization](#).

9.54.3.25 `virtual int CoinOslFactorization::updateColumnFT (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool noPermute = false)` [virtual]

Updates one column (FTRAN) from regionSparse2 Tries to do FT update number returned is negative if no room region-Sparse starts as zero and is zero at end.

Note - if regionSparse2 packed on input - will be packed on output

Implements [CoinOtherFactorization](#).

9.54.3.26 `virtual int CoinOslFactorization::updateColumn (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool noPermute = false) const` [virtual]

This version has same effect as above with FTUpdate==false so number returned is always >=0.

Implements [CoinOtherFactorization](#).

9.54.3.27 `virtual int CoinOslFactorization::updateTwoColumnsFT (CoinIndexedVector * regionSparse1, CoinIndexedVector * regionSparse2, CoinIndexedVector * regionSparse3, bool noPermute = false)` [virtual]

does FTRAN on two columns

Implements [CoinOtherFactorization](#).

9.54.3.28 `virtual int CoinOslFactorization::updateColumnTranspose (CoinIndexedVector * regionSparse,
CoinIndexedVector * regionSparse2) const` [virtual]

Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if region-Sparse2 packed on input - will be packed on output.

Implements [CoinOtherFactorization](#).

9.54.3.29 `virtual int* CoinOslFactorization::indices () const` [virtual]

Get rid of all memory.

Returns array to put basis indices in

Implements [CoinOtherFactorization](#).

9.54.3.30 `virtual int* CoinOslFactorization::permute () const` [inline],[virtual]

Returns permute in.

Implements [CoinOtherFactorization](#).

Definition at line 255 of file CoinOslFactorization.hpp.

9.54.3.31 `void CoinOslFactorization::gutsOfDestructor (bool clearFact = true)`

The real work of desstructor.

9.54.3.32 `void CoinOslFactorization::gutsOfInitialize (bool zapFact = true)`

The real work of constructor.

9.54.3.33 `void CoinOslFactorization::gutsOfCopy (const CoinOslFactorization & other)`

The real work of copy.

9.54.3.34 `int CoinOslFactorization::checkPivot (double saveFromU, double oldPivot) const` [protected]

Returns accuracy status of replaceColumn returns 0=OK, 1=Probably OK, 2=singular.

9.54.4 Friends And Related Function Documentation

9.54.4.1 `void CoinOslFactorizationUnitTest (const std::string & mpsDir)` [friend]

9.54.5 Member Data Documentation

9.54.5.1 `EKKfactinfo CoinOslFactorization::factInfo_` [protected]

Osl factorization data.

Definition at line 277 of file CoinOslFactorization.hpp.

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

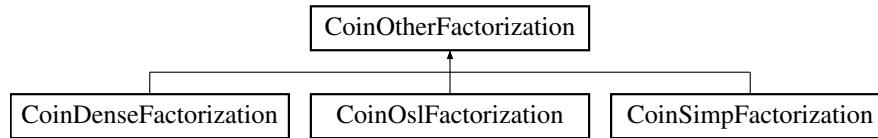
- [/home/ted/COIN/trunk/CoinUtils/src/CoinOslFactorization.hpp](#)

9.55 CoinOtherFactorization Class Reference

Abstract base class which also has some scalars so can be used from Dense or Simp.

```
#include <CoinDenseFactorization.hpp>
```

Inheritance diagram for CoinOtherFactorization:



Public Member Functions

Constructors and destructor and copy

- [CoinOtherFactorization](#) ()
Default constructor.
- [CoinOtherFactorization](#) (const [CoinOtherFactorization](#) &other)
Copy constructor.
- virtual [~CoinOtherFactorization](#) ()
Destructor.
- [CoinOtherFactorization](#) & [operator=](#) (const [CoinOtherFactorization](#) &other)
= copy
- virtual [CoinOtherFactorization](#) * [clone](#) () const =0
Clone.

general stuff such as status

- int [status](#) () const
Returns status.
- void [setStatus](#) (int value)
Sets status.
- int [pivots](#) () const
Returns number of pivots since factorization.
- void [setPivots](#) (int value)
Sets number of pivots since factorization.
- void [setNumberRows](#) (int value)
Set number of Rows after factorization.
- int [numberRows](#) () const
Number of Rows after factorization.
- int [numberColumns](#) () const
Total number of columns in factorization.
- int [numberGoodColumns](#) () const
Number of good columns in factorization.
- void [relaxAccuracyCheck](#) (double value)
Allows change of pivot accuracy check 1.0 == none > 1.0 relaxed.
- double [getAccuracyCheck](#) () const
- int [maximumPivots](#) () const
Maximum number of pivots between factorizations.
- virtual void [maximumPivots](#) (int value)
Set maximum pivots.

- double `pivotTolerance` () const
Pivot tolerance.
- void `pivotTolerance` (double value)
- double `zeroTolerance` () const
Zero tolerance.
- void `zeroTolerance` (double value)
- double `slackValue` () const
Whether slack value is +1 or -1.
- void `slackValue` (double value)
- virtual `CoinFactorizationDouble * elements` () const
Returns array to put basis elements in.
- virtual int * `pivotRow` () const
Returns pivot row.
- virtual `CoinFactorizationDouble * workArea` () const
Returns work area.
- virtual int * `intWorkArea` () const
Returns int work area.
- virtual int * `numberInRow` () const
Number of entries in each row.
- virtual int * `numberInColumn` () const
Number of entries in each column.
- virtual `CoinBigIndex * starts` () const
Returns array to put basis starts in.
- virtual int * `permuteBack` () const
Returns permute back.
- int `solveMode` () const
Get solve mode e.g.
- void `setSolveMode` (int value)
Set solve mode e.g.
- virtual bool `wantsTableauColumn` () const
Returns true if wants tableauColumn in replaceColumn.
- virtual void `setUsefullInformation` (const int *info, int whereFrom)
Useful information for factorization 0 - iteration number whereFrom is 0 for factorize and 1 for replaceColumn.
- virtual void `clearArrays` ()
Get rid of all memory.

virtual general stuff such as permutation

- virtual int * `indices` () const =0
Returns array to put basis indices in.
- virtual int * `permute` () const =0
Returns permute in.
- virtual int `numberElements` () const =0
Total number of elements in factorization.

Do factorization - public

- virtual void `getAreas` (int `numberRows`, int `numberColumns`, `CoinBigIndex` `maximumL`, `CoinBigIndex` `maximumU`)=0
Gets space for a factorization.
- virtual void `preProcess` ()=0
PreProcesses column ordered copy of basis.
- virtual int `factor` ()=0
Does most of factorization returning status 0 - OK -99 - needs more memory -1 - singular - use numberGoodColumns and redo.

- virtual `void postProcess` (const int *sequence, int *pivotVariable)=0
Does post processing on valid factorization - putting variables on correct rows.
- virtual `void makeNonSingular` (int *sequence, int numberColumns)=0
Makes a non-singular basis by replacing variables.

rank one updates which do exist

- virtual int `replaceColumn` (CoinIndexedVector *regionSparse, int pivotRow, double pivotCheck, bool checkBeforeModifying=false, double acceptablePivot=1.0e-8)=0
Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If checkBeforeModifying is true will do all accuracy checks before modifying factorization.

various uses of factorization (return code number elements)

which user may want to know about

- virtual int `updateColumnFT` (CoinIndexedVector *regionSparse, CoinIndexedVector *regionSparse2, bool noPermute=false)=0
Updates one column (FTRAN) from regionSparse2 Tries to do FT update number returned is negative if no room regionSparse starts as zero and is zero at end.
- virtual int `updateColumn` (CoinIndexedVector *regionSparse, CoinIndexedVector *regionSparse2, bool noPermute=false) const =0
This version has same effect as above with FTUpdate==false so number returned is always >=0.
- virtual int `updateTwoColumnsFT` (CoinIndexedVector *regionSparse1, CoinIndexedVector *regionSparse2, CoinIndexedVector *regionSparse3, bool noPermute=false)=0
does FTRAN on two columns
- virtual int `updateColumnTranspose` (CoinIndexedVector *regionSparse, CoinIndexedVector *regionSparse2) const =0
Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if regionSparse2 packed on input - will be packed on output.

Protected Attributes

data

- double `pivotTolerance_`
Pivot tolerance.
- double `zeroTolerance_`
Zero tolerance.
- double `slackValue_`
Whether slack value is +1 or -1.
- double `relaxCheck_`
Relax check on accuracy in replaceColumn.
- `CoinBigIndex` `factorElements_`
Number of elements after factorization.
- int `numberRows_`
Number of Rows in factorization.
- int `numberColumns_`
Number of Columns in factorization.
- int `numberGoodU_`
Number factorized in U (not row singletons)
- int `maximumPivots_`
Maximum number of pivots before factorization.
- int `numberPivots_`
Number pivots since last factorization.

- int `status_`
Status of factorization.
- int `maximumRows_`
Maximum rows ever (i.e. use to copy arrays etc)
- `CoinBigIndex` `maximumSpace_`
Maximum length of iterating area.
- int * `pivotRow_`
Pivot row.
- `CoinFactorizationDouble` * `elements_`
*Elements of factorization and updates length is $\max R * \max R + \max \text{Space}$ will always be long enough so can have $nR * nR$ ints in $\max \text{Space}$.*
- `CoinFactorizationDouble` * `workArea_`
Work area of `numberRows_`.
- int `solveMode_`
Solve mode e.g.

9.55.1 Detailed Description

Abstract base class which also has some scalars so can be used from Dense or Simp.

Definition at line 24 of file `CoinDenseFactorization.hpp`.

9.55.2 Constructor & Destructor Documentation

9.55.2.1 `CoinOtherFactorization::CoinOtherFactorization ()`

Default constructor.

9.55.2.2 `CoinOtherFactorization::CoinOtherFactorization (const CoinOtherFactorization & other)`

Copy constructor.

9.55.2.3 `virtual CoinOtherFactorization::~~CoinOtherFactorization () [virtual]`

Destructor.

9.55.3 Member Function Documentation

9.55.3.1 `CoinOtherFactorization& CoinOtherFactorization::operator= (const CoinOtherFactorization & other)`

= copy

9.55.3.2 `virtual CoinOtherFactorization* CoinOtherFactorization::clone () const [pure virtual]`

Clone.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.3 `int CoinOtherFactorization::status () const [inline]`

Returns status.

Definition at line 47 of file `CoinDenseFactorization.hpp`.

9.55.3.4 `void CoinOtherFactorization::setStatus (int value)` `[inline]`

Sets status.

Definition at line 51 of file CoinDenseFactorization.hpp.

9.55.3.5 `int CoinOtherFactorization::pivots () const` `[inline]`

Returns number of pivots since factorization.

Definition at line 54 of file CoinDenseFactorization.hpp.

9.55.3.6 `void CoinOtherFactorization::setPivots (int value)` `[inline]`

Sets number of pivots since factorization.

Definition at line 58 of file CoinDenseFactorization.hpp.

9.55.3.7 `void CoinOtherFactorization::setNumberRows (int value)` `[inline]`

Set number of Rows after factorization.

Definition at line 61 of file CoinDenseFactorization.hpp.

9.55.3.8 `int CoinOtherFactorization::numberRows () const` `[inline]`

Number of Rows after factorization.

Definition at line 64 of file CoinDenseFactorization.hpp.

9.55.3.9 `int CoinOtherFactorization::numberColumns () const` `[inline]`

Total number of columns in factorization.

Definition at line 68 of file CoinDenseFactorization.hpp.

9.55.3.10 `int CoinOtherFactorization::numberGoodColumns () const` `[inline]`

Number of good columns in factorization.

Definition at line 72 of file CoinDenseFactorization.hpp.

9.55.3.11 `void CoinOtherFactorization::relaxAccuracyCheck (double value)` `[inline]`

Allows change of pivot accuracy check 1.0 == none >1.0 relaxed.

Definition at line 76 of file CoinDenseFactorization.hpp.

9.55.3.12 `double CoinOtherFactorization::getAccuracyCheck () const` `[inline]`

Definition at line 78 of file CoinDenseFactorization.hpp.

9.55.3.13 `int CoinOtherFactorization::maximumPivots () const` `[inline]`

Maximum number of pivots between factorizations.

Definition at line 81 of file CoinDenseFactorization.hpp.

9.55.3.14 `virtual void CoinOtherFactorization::maximumPivots (int value)` `[virtual]`

Set maximum pivots.

Reimplemented in [CoinOslFactorization](#).

9.55.3.15 `double CoinOtherFactorization::pivotTolerance () const [inline]`

Pivot tolerance.

Definition at line 88 of file `CoinDenseFactorization.hpp`.

9.55.3.16 `void CoinOtherFactorization::pivotTolerance (double value)`

9.55.3.17 `double CoinOtherFactorization::zeroTolerance () const [inline]`

Zero tolerance.

Definition at line 93 of file `CoinDenseFactorization.hpp`.

9.55.3.18 `void CoinOtherFactorization::zeroTolerance (double value)`

9.55.3.19 `double CoinOtherFactorization::slackValue () const [inline]`

Whether slack value is +1 or -1.

Definition at line 99 of file `CoinDenseFactorization.hpp`.

9.55.3.20 `void CoinOtherFactorization::slackValue (double value)`

9.55.3.21 `virtual CoinFactorizationDouble* CoinOtherFactorization::elements () const [virtual]`

Returns array to put basis elements in.

Reimplemented in [CoinOslFactorization](#).

9.55.3.22 `virtual int* CoinOtherFactorization::pivotRow () const [virtual]`

Returns pivot row.

Reimplemented in [CoinOslFactorization](#).

9.55.3.23 `virtual CoinFactorizationDouble* CoinOtherFactorization::workArea () const [virtual]`

Returns work area.

Reimplemented in [CoinOslFactorization](#).

9.55.3.24 `virtual int* CoinOtherFactorization::intWorkArea () const [virtual]`

Returns int work area.

Reimplemented in [CoinOslFactorization](#).

9.55.3.25 `virtual int* CoinOtherFactorization::numberInRow () const [virtual]`

Number of entries in each row.

Reimplemented in [CoinOslFactorization](#).

9.55.3.26 `virtual int* CoinOtherFactorization::numberInColumn () const [virtual]`

Number of entries in each column.

Reimplemented in [CoinOslFactorization](#).

9.55.3.27 `virtual CoinBigIndex* CoinOtherFactorization::starts () const [virtual]`

Returns array to put basis starts in.

Reimplemented in [CoinOslFactorization](#).

9.55.3.28 `virtual int* CoinOtherFactorization::permuteBack () const [virtual]`

Returns permute back.

Reimplemented in [CoinOslFactorization](#).

9.55.3.29 `int CoinOtherFactorization::solveMode () const [inline]`

Get solve mode e.g.

0 C++ code, 1 Lapack, 2 choose If 4 set then values pass if 8 set then has iterated

Definition at line 124 of file `CoinDenseFactorization.hpp`.

9.55.3.30 `void CoinOtherFactorization::setSolveMode (int value) [inline]`

Set solve mode e.g.

0 C++ code, 1 Lapack, 2 choose If 4 set then values pass if 8 set then has iterated

Definition at line 130 of file `CoinDenseFactorization.hpp`.

9.55.3.31 `virtual bool CoinOtherFactorization::wantsTableauColumn () const [virtual]`

Returns true if wants tableauColumn in replaceColumn.

Reimplemented in [CoinOslFactorization](#).

9.55.3.32 `virtual void CoinOtherFactorization::setUsefullInformation (const int * info, int whereFrom) [virtual]`

Useful information for factorization 0 - iteration number whereFrom is 0 for factorize and 1 for replaceColumn.

Reimplemented in [CoinOslFactorization](#).

9.55.3.33 `virtual void CoinOtherFactorization::clearArrays () [inline],[virtual]`

Get rid of all memory.

Reimplemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

Definition at line 140 of file `CoinDenseFactorization.hpp`.

9.55.3.34 `virtual int* CoinOtherFactorization::indices () const [pure virtual]`

Returns array to put basis indices in.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.35 `virtual int* CoinOtherFactorization::permute () const [pure virtual]`

Returns permute in.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.36 `virtual int CoinOtherFactorization::numberElements () const [pure virtual]`

Total number of elements in factorization.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.37 `virtual void CoinOtherFactorization::getAreas (int numberRows, int numberColumns, CoinBigIndex maximumL, CoinBigIndex maximumU) [pure virtual]`

Gets space for a factorization.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.38 `virtual void CoinOtherFactorization::preProcess () [pure virtual]`

PreProcesses column ordered copy of basis.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.39 `virtual int CoinOtherFactorization::factor () [pure virtual]`

Does most of factorization returning status 0 - OK -99 - needs more memory -1 - singular - use `numberGoodColumns` and redo.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.40 `virtual void CoinOtherFactorization::postProcess (const int * sequence, int * pivotVariable) [pure virtual]`

Does post processing on valid factorization - putting variables on correct rows.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.41 `virtual void CoinOtherFactorization::makeNonSingular (int * sequence, int numberColumns) [pure virtual]`

Makes a non-singular basis by replacing variables.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.42 `virtual int CoinOtherFactorization::replaceColumn (CoinIndexedVector * regionSparse, int pivotRow, double pivotCheck, bool checkBeforeModifying = false, double acceptablePivot = 1.0e-8) [pure virtual]`

Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If `checkBeforeModifying` is true will do all accuracy checks before modifying factorization.

Whether to set this depends on speed considerations. You could just do this on first iteration after factorization and thereafter re-factorize partial update already in U

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.43 `virtual int CoinOtherFactorization::updateColumnFT (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool noPermute = false) [pure virtual]`

Updates one column (FTRAN) from `regionSparse2` Tries to do FT update number returned is negative if no room `regionSparse` starts as zero and is zero at end.

Note - if `regionSparse2` packed on input - will be packed on output

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.3.44 `virtual int CoinOtherFactorization::updateColumn (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool noPermute = false) const [pure virtual]`

This version has same effect as above with `FTUpdate==false` so number returned is always ≥ 0 .

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

```
9.55.3.45 virtual int CoinOtherFactorization::updateTwoColumnsFT ( CoinIndexedVector * regionSparse1,
    CoinIndexedVector * regionSparse2, CoinIndexedVector * regionSparse3, bool noPermute = false )
    [pure virtual]
```

does FTRAN on two columns

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

```
9.55.3.46 virtual int CoinOtherFactorization::updateColumnTranspose ( CoinIndexedVector * regionSparse,
    CoinIndexedVector * regionSparse2 ) const [pure virtual]
```

Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if region-Sparse2 packed on input - will be packed on output.

Implemented in [CoinDenseFactorization](#), [CoinOslFactorization](#), and [CoinSimpFactorization](#).

9.55.4 Member Data Documentation

```
9.55.4.1 double CoinOtherFactorization::pivotTolerance_ [protected]
```

Pivot tolerance.

Definition at line 226 of file [CoinDenseFactorization.hpp](#).

```
9.55.4.2 double CoinOtherFactorization::zeroTolerance_ [protected]
```

Zero tolerance.

Definition at line 228 of file [CoinDenseFactorization.hpp](#).

```
9.55.4.3 double CoinOtherFactorization::slackValue_ [protected]
```

Whether slack value is +1 or -1.

Definition at line 231 of file [CoinDenseFactorization.hpp](#).

```
9.55.4.4 double CoinOtherFactorization::relaxCheck_ [protected]
```

Relax check on accuracy in replaceColumn.

Definition at line 238 of file [CoinDenseFactorization.hpp](#).

```
9.55.4.5 CoinBigIndex CoinOtherFactorization::factorElements_ [protected]
```

Number of elements after factorization.

Definition at line 240 of file [CoinDenseFactorization.hpp](#).

```
9.55.4.6 int CoinOtherFactorization::numberRows_ [protected]
```

Number of Rows in factorization.

Definition at line 242 of file [CoinDenseFactorization.hpp](#).

```
9.55.4.7 int CoinOtherFactorization::numberColumns_ [protected]
```

Number of Columns in factorization.

Definition at line 244 of file [CoinDenseFactorization.hpp](#).

9.55.4.8 `int CoinOtherFactorization::numberGoodU_` [protected]

Number factorized in U (not row singletons)

Definition at line 246 of file `CoinDenseFactorization.hpp`.

9.55.4.9 `int CoinOtherFactorization::maximumPivots_` [protected]

Maximum number of pivots before factorization.

Definition at line 248 of file `CoinDenseFactorization.hpp`.

9.55.4.10 `int CoinOtherFactorization::numberPivots_` [protected]

Number pivots since last factorization.

Definition at line 250 of file `CoinDenseFactorization.hpp`.

9.55.4.11 `int CoinOtherFactorization::status_` [protected]

Status of factorization.

Definition at line 252 of file `CoinDenseFactorization.hpp`.

9.55.4.12 `int CoinOtherFactorization::maximumRows_` [protected]

Maximum rows ever (i.e. use to copy arrays etc)

Definition at line 254 of file `CoinDenseFactorization.hpp`.

9.55.4.13 `CoinBigIndex CoinOtherFactorization::maximumSpace_` [protected]

Maximum length of iterating area.

Definition at line 256 of file `CoinDenseFactorization.hpp`.

9.55.4.14 `int* CoinOtherFactorization::pivotRow_` [protected]

Pivot row.

Definition at line 258 of file `CoinDenseFactorization.hpp`.

9.55.4.15 `CoinFactorizationDouble* CoinOtherFactorization::elements_` [protected]

Elements of factorization and updates length is $\max R * \max R + \max \text{Space}$ will always be long enough so can have $nR * nR$ ints in `maxSpace`.

Definition at line 263 of file `CoinDenseFactorization.hpp`.

9.55.4.16 `CoinFactorizationDouble* CoinOtherFactorization::workArea_` [protected]

Work area of `numberRows_`.

Definition at line 265 of file `CoinDenseFactorization.hpp`.

9.55.4.17 `int CoinOtherFactorization::solveMode_` [protected]

Solve mode e.g.

0 C++ code, 1 Lapack, 2 choose If 4 set then values pass if 8 set then has iterated

Definition at line 270 of file `CoinDenseFactorization.hpp`.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinDenseFactorization.hpp](#)

9.56 CoinPackedMatrix Class Reference

Sparse Matrix Base Class.

```
#include <CoinPackedMatrix.hpp>
```

Public Member Functions

Query members

- double [getExtraGap](#) () const
Return the current setting of the extra gap.
- double [getExtraMajor](#) () const
Return the current setting of the extra major.
- void [reserve](#) (const int newMaxMajorDim, const [CoinBigIndex](#) newMaxSize, bool create=false)
Reserve sufficient space for appending major-ordered vectors.
- void [clear](#) ()
Clear the data, but do not free any arrays.
- bool [isColOrdered](#) () const
Whether the packed matrix is column major ordered or not.
- bool [hasGaps](#) () const
Whether the packed matrix has gaps or not.
- [CoinBigIndex](#) [getNumElements](#) () const
Number of entries in the packed matrix.
- int [getNumCols](#) () const
Number of columns.
- int [getNumRows](#) () const
Number of rows.
- const double * [getElements](#) () const
A vector containing the elements in the packed matrix.
- const int * [getIndices](#) () const
A vector containing the minor indices of the elements in the packed matrix.
- int [getSizeVectorStarts](#) () const
The size of the `vectorStarts` array.
- int [getSizeVectorLengths](#) () const
The size of the `vectorLengths` array.
- const [CoinBigIndex](#) * [getVectorStarts](#) () const
The positions where the major-dimension vectors start in elements and indices.
- const int * [getVectorLengths](#) () const
The lengths of the major-dimension vectors.
- [CoinBigIndex](#) [getVectorFirst](#) (const int i) const
*The position of the first element in the *i*'th major-dimension vector.*
- [CoinBigIndex](#) [getVectorLast](#) (const int i) const
*The position of the last element (well, one entry past the last) in the *i*'th major-dimension vector.*
- int [getVectorSize](#) (const int i) const
*The length of *i*'th vector.*
- const [CoinShallowPackedVector](#) [getVector](#) (int i) const
*Return the *i*'th vector in matrix.*
- int * [getMajorIndices](#) () const
Returns an array containing major indices.

Modifying members

- **void setDimensions** (int numRows, int numcols)
Set the dimensions of the matrix.
- **void setExtraGap** (const double newGap)
Set the extra gap to be allocated to the specified value.
- **void setExtraMajor** (const double newMajor)
Set the extra major to be allocated to the specified value.
- **void appendCol** (const **CoinPackedVectorBase** &vec)
Append a column to the end of the matrix.
- **void appendCol** (const int vecsize, const int *vecind, const double *vecelem)
Append a column to the end of the matrix.
- **void appendCols** (const int numcols, const **CoinPackedVectorBase** *const *cols)
Append a set of columns to the end of the matrix.
- **int appendCols** (const int numcols, const **CoinBigIndex** *columnStarts, const int *row, const double *element, int numRows=-1)
Append a set of columns to the end of the matrix.
- **void appendRow** (const **CoinPackedVectorBase** &vec)
Append a row to the end of the matrix.
- **void appendRow** (const int vecsize, const int *vecind, const double *vecelem)
Append a row to the end of the matrix.
- **void appendRows** (const int numRows, const **CoinPackedVectorBase** *const *rows)
Append a set of rows to the end of the matrix.
- **int appendRows** (const int numRows, const **CoinBigIndex** *rowStarts, const int *column, const double *element, int numColumns=-1)
Append a set of rows to the end of the matrix.
- **void rightAppendPackedMatrix** (const **CoinPackedMatrix** &matrix)
Append the argument to the "right" of the current matrix.
- **void bottomAppendPackedMatrix** (const **CoinPackedMatrix** &matrix)
Append the argument to the "bottom" of the current matrix.
- **void deleteCols** (const int numDel, const int *indDel)
Delete the columns whose indices are listed in indDel.
- **void deleteRows** (const int numDel, const int *indDel)
Delete the rows whose indices are listed in indDel.
- **void replaceVector** (const int index, const int numReplace, const double *newElements)
Replace the elements of a vector.
- **void modifyCoefficient** (int row, int column, double newElement, bool keepZero=false)
Modify one element of packed matrix.
- **double getCoefficient** (int row, int column) const
Return one element of packed matrix.
- **int compress** (double threshold)
Eliminate all elements in matrix whose absolute value is less than threshold.
- **int eliminateDuplicates** (double threshold)
Eliminate all duplicate AND small elements in matrix The column starts are not affected.
- **void orderMatrix** ()
Sort all columns so indices are increasing in each column.
- **int cleanMatrix** (double threshold=1.0e-20)
Really clean up matrix.

Methods that reorganize the whole matrix

- **void removeGaps** (double removeValue=-1.0)
Remove the gaps from the matrix if there were any Can also remove small elements fabs() <= removeValue.
- **void submatrixOf** (const **CoinPackedMatrix** &matrix, const int numMajor, const int *indMajor)
Extract a submatrix from matrix.

- **void submatrixOfWithDuplicates** (const [CoinPackedMatrix](#) &matrix, const int numMajor, const int *indMajor)
Extract a submatrix from matrix.
- **void copyOf** (const [CoinPackedMatrix](#) &rhs)
Copy method.
- **void copyOf** (const bool colordered, const int minor, const int major, const [CoinBigIndex](#) numels, const double *elem, const int *ind, const [CoinBigIndex](#) *start, const int *len, const double extraMajor=0.0, const double extraGap=0.0)
Copy the arguments to the matrix.
- **void copyReuseArrays** (const [CoinPackedMatrix](#) &rhs)
Copy method.
- **void reverseOrderedCopyOf** (const [CoinPackedMatrix](#) &rhs)
Make a reverse-ordered copy.
- **void assignMatrix** (const bool colordered, const int minor, const int major, const [CoinBigIndex](#) numels, double *&elem, int *&ind, [CoinBigIndex](#) *&start, int *&len, const int maxmajor=-1, const [CoinBigIndex](#) maxsize=-1)
Assign the arguments to the matrix.
- **[CoinPackedMatrix](#) & operator=** (const [CoinPackedMatrix](#) &rhs)
Assignment operator.
- **void reverseOrdering** ()
Reverse the ordering of the packed matrix.
- **void transpose** ()
Transpose the matrix.
- **void swap** ([CoinPackedMatrix](#) &matrix)
Swap the content of two packed matrices.

Matrix times vector methods

- **void times** (const double *x, double *y) const
*Return $A * x$ in y .*
- **void times** (const [CoinPackedVectorBase](#) &x, double *y) const
*Return $A * x$ in y .*
- **void transposeTimes** (const double *x, double *y) const
*Return $x * A$ in y .*
- **void transposeTimes** (const [CoinPackedVectorBase](#) &x, double *y) const
*Return $x * A$ in y .*

Queries

- int * **countOrthoLength** () const
Count the number of entries in every minor-dimension vector and return an array containing these lengths.
- **void countOrthoLength** (int *counts) const
Count the number of entries in every minor-dimension vector and fill in an array containing these lengths.
- int **getMajorDim** () const
Major dimension.
- **void setMajorDim** (int value)
Set major dimension.
- int **getMinorDim** () const
Minor dimension.
- **void setMinorDim** (int value)
Set minor dimension.
- int **getMaxMajorDim** () const
Current maximum for major dimension.
- **void dumpMatrix** (const char *fname=NULL) const
Dump the matrix on stdout.
- **void printMatrixElement** (const int row_val, const int col_val) const

Print a single matrix element.

Append vectors

When compiled with `COIN_DEBUG` defined these methods throw an exception if the major (minor) vector contains an index that's invalid for the minor (major) dimension. Otherwise the methods assume that every index fits into the matrix.

- `void appendMajorVector` (const `CoinPackedVectorBase` &vec)
Append a major-dimension vector to the end of the matrix.
- `void appendMajorVector` (const int vecsize, const int *vecind, const double *vecelem)
Append a major-dimension vector to the end of the matrix.
- `void appendMajorVectors` (const int numvecs, const `CoinPackedVectorBase` *const *vecs)
Append several major-dimension vectors to the end of the matrix.
- `void appendMinorVector` (const `CoinPackedVectorBase` &vec)
Append a minor-dimension vector to the end of the matrix.
- `void appendMinorVector` (const int vecsize, const int *vecind, const double *vecelem)
Append a minor-dimension vector to the end of the matrix.
- `void appendMinorVectors` (const int numvecs, const `CoinPackedVectorBase` *const *vecs)
Append several minor-dimension vectors to the end of the matrix.
- `void appendMinorFast` (const int number, const `CoinBigIndex` *starts, const int *index, const double *element)
Append a set of rows (columns) to the end of a column (row) ordered matrix.

Append matrices

We'll document these methods assuming that the current matrix is column major ordered (Hence in the `... SameOrdered()` methods the argument is column ordered, in the `OrthoOrdered()` methods the argument is row ordered.)

- `void majorAppendSameOrdered` (const `CoinPackedMatrix` &matrix)
Append the columns of the argument to the right end of this matrix.
- `void minorAppendSameOrdered` (const `CoinPackedMatrix` &matrix)
Append the columns of the argument to the bottom end of this matrix.
- `void majorAppendOrthoOrdered` (const `CoinPackedMatrix` &matrix)
Append the rows of the argument to the right end of this matrix.
- `void minorAppendOrthoOrdered` (const `CoinPackedMatrix` &matrix)
Append the rows of the argument to the bottom end of this matrix.

Delete vectors

- `void deleteMajorVectors` (const int numDel, const int *indDel)
Delete the major-dimension vectors whose indices are listed in `indDel`.
- `void deleteMinorVectors` (const int numDel, const int *indDel)
Delete the minor-dimension vectors whose indices are listed in `indDel`.

Various dot products.

- `void timesMajor` (const double *x, double *y) const
*Return $A * x$ (multiplied from the "right" direction) in y .*
- `void timesMajor` (const `CoinPackedVectorBase` &x, double *y) const
*Return $A * x$ (multiplied from the "right" direction) in y .*
- `void timesMinor` (const double *x, double *y) const
*Return $A * x$ (multiplied from the "right" direction) in y .*
- `void timesMinor` (const `CoinPackedVectorBase` &x, double *y) const
*Return $A * x$ (multiplied from the "right" direction) in y .*

Logical Operations.

- `template<class FloatEqual >`
`bool isEquivalent (const CoinPackedMatrix &rhs, const FloatEqual &eq) const`
Test for equivalence.
- `bool isEquivalent2 (const CoinPackedMatrix &rhs) const`
Test for equivalence and report differences.
- `bool isEquivalent (const CoinPackedMatrix &rhs) const`
Test for equivalence.

Non-const methods

These are to be used with great care when doing column generation, etc.

- `double * getMutableElements () const`
A vector containing the elements in the packed matrix.
- `int * getMutableIndices () const`
A vector containing the minor indices of the elements in the packed matrix.
- `CoinBigIndex * getMutableVectorStarts () const`
The positions where the major-dimension vectors start in `element_` and `index_`.
- `int * getMutableVectorLengths () const`
The lengths of the major-dimension vectors.
- `void setNumElements (CoinBigIndex value)`
Change the size of the bulk store after modifying - be careful.
- `void nullElementArray ()`
NULLify element array.
- `void nullStartArray ()`
NULLify start array.
- `void nullLengthArray ()`
NULLify length array.
- `void nullIndexArray ()`
NULLify index array.

Constructors and destructors

- `CoinPackedMatrix ()`
Default Constructor creates an empty column ordered packed matrix.
- `CoinPackedMatrix (const bool colordered, const double extraMajor, const double extraGap)`
A constructor where the ordering and the gaps are specified.
- `CoinPackedMatrix (const bool colordered, const int minor, const int major, const CoinBigIndex numels, const double *elem, const int *ind, const CoinBigIndex *start, const int *len, const double extraMajor, const double extraGap)`
- `CoinPackedMatrix (const bool colordered, const int minor, const int major, const CoinBigIndex numels, const double *elem, const int *ind, const CoinBigIndex *start, const int *len)`
- `CoinPackedMatrix (const bool colordered, const int *rowIndices, const int *colIndices, const double *elements, CoinBigIndex numels)`
Create packed matrix from triples.
- `CoinPackedMatrix (const CoinPackedMatrix &m)`
Copy constructor.
- `CoinPackedMatrix (const CoinPackedMatrix &m, int extraForMajor, int extraElements, bool reverseOrdering=false)`
Copy constructor with fine tuning.
- `CoinPackedMatrix (const CoinPackedMatrix &wholeModel, int numberOfRows, const int *whichRows, int numberColumns, const int *whichColumns)`
Subset constructor (without gaps).
- `virtual ~CoinPackedMatrix ()`

Destructor.

Debug Utilities

- int [verifyMtx](#) (int verbosity=1, bool zeroesAreError=false) const
Scan the matrix for anomalies.

Protected Member Functions

- void [gutsOfDestructor](#) ()
- void [gutsOfCopyOf](#) (const bool colordered, const int minor, const int major, const [CoinBigIndex](#) numels, const double *elem, const int *ind, const [CoinBigIndex](#) *start, const int *len, const double extraMajor=0.0, const double extraGap=0.0)
- void [gutsOfCopyOfNoGaps](#) (const bool colordered, const int minor, const int major, const double *elem, const int *ind, const [CoinBigIndex](#) *start)

When no gaps we can do faster.

- void [gutsOfOpEqual](#) (const bool colordered, const int minor, const int major, const [CoinBigIndex](#) numels, const double *elem, const int *ind, const [CoinBigIndex](#) *start, const int *len)
- void [resizeForAddingMajorVectors](#) (const int numVec, const int *lengthVec)
- void [resizeForAddingMinorVectors](#) (const int *addedEntries)
- int [appendMajor](#) (const int number, const [CoinBigIndex](#) *starts, const int *index, const double *element, int numberOther=-1)

Append a set of rows (columns) to the end of a row (column) ordered matrix.

- int [appendMinor](#) (const int number, const [CoinBigIndex](#) *starts, const int *index, const double *element, int numberOther=-1)

Append a set of rows (columns) to the end of a column (row) ordered matrix.

Protected Attributes

Data members

The data members are protected to allow access for derived classes.

- bool [colOrdered_](#)
A flag indicating whether the matrix is column or row major ordered.
- double [extraGap_](#)
This much times more space should be allocated for each major-dimension vector (with respect to the number of entries in the vector) when the matrix is resized.
- double [extraMajor_](#)
his much times more space should be allocated for major-dimension vectors when the matrix is resized.
- double * [element_](#)
List of nonzero element values.
- int * [index_](#)
List of nonzero element minor-dimension indices.
- [CoinBigIndex](#) * [start_](#)
Starting positions of major-dimension vectors.
- int * [length_](#)
Lengths of major-dimension vectors.
- int [majorDim_](#)
number of vectors in matrix
- int [minorDim_](#)
size of other dimension
- [CoinBigIndex](#) [size_](#)

- *the number of nonzero entries*
- `int maxMajorDim_`
max space allocated for major-dimension
- `CoinBigIndex maxSize_`
max space allocated for entries

Friends

- `void CoinPackedMatrixUnitTest ()`
Test the methods in the [CoinPackedMatrix](#) class.

9.56.1 Detailed Description

Sparse Matrix Base Class.

This class is intended to represent sparse matrices using row-major or column-major ordering. The representation is very efficient for adding, deleting, or retrieving major-dimension vectors. Adding a minor-dimension vector is less efficient, but can be helped by providing "extra" space as described in the next paragraph. Deleting a minor-dimension vector requires inspecting all coefficients in the matrix. Retrieving a minor-dimension vector would incur the same cost and is not supported (except in the sense that you can write a loop to retrieve all coefficients one at a time). Consider physically transposing the matrix, or keeping a second copy with the other major-vector ordering.

The sparse representation can be completely compact or it can have "extra" space available at the end of each major vector. Incorporating extra space into the sparse matrix representation can improve performance in cases where new data needs to be inserted into the packed matrix against the major-vector orientation (e.g, inserting a row into a matrix stored in column-major order).

For example if the matrix:

```

3  1  0  -2  -1  0  0  -1
0  2  1.1  0  0  0  0  0
0  0  1  0  0  1  0  0
0  0  0  2.8  0  0  -1.2  0
5.6  0  0  0  1  0  0  1.9

```

was stored by rows (with no extra space) in

`CoinPackedMatrix r` then:

```

r.getElements() returns a vector containing:
3 1 -2 -1 -1 2 1.1 1 1 2.8 -1.2 5.6 1 1.9
r.getIndices() returns a vector containing:
0 1 3 4 7 1 2 2 5 3 6 0 4 7
r.getVectorStarts() returns a vector containing:
0 5 7 9 11 14
r.getNumElements() returns 14.
r.getMajorDim() returns 5.
r.getVectorSize(0) returns 5.
r.getVectorSize(1) returns 2.
r.getVectorSize(2) returns 2.
r.getVectorSize(3) returns 2.
r.getVectorSize(4) returns 3.

```

If stored by columns (with no extra space) then:

```

c.getElements() returns a vector containing:
3 5.6 1 2 1.1 1 -2 2.8 -1 1 1 -1.2 -1 1.9
c.getIndices() returns a vector containing:
0 4 0 1 1 2 0 3 0 4 2 3 0 4
c.getVectorStarts() returns a vector containing:
0 2 4 6 8 10 11 12 14
c.getNumElements() returns 14.
c.getMajorDim() returns 8.

```

Compiling this class with `CLP_NO_VECTOR` defined will excise all methods which use [CoinPackedVectorBase](#), [CoinPackedVector](#), or [CoinShallowPackedVector](#) as parameters or return types.

Compiling this class with `COIN_FAST_CODE` defined removes index range checks.

Definition at line 79 of file `CoinPackedMatrix.hpp`.

9.56.2 Constructor & Destructor Documentation

9.56.2.1 `CoinPackedMatrix::CoinPackedMatrix ()`

Default Constructor creates an empty column ordered packed matrix.

9.56.2.2 `CoinPackedMatrix::CoinPackedMatrix (const bool colordered, const double extraMajor, const double extraGap)`

A constructor where the ordering and the gaps are specified.

9.56.2.3 `CoinPackedMatrix::CoinPackedMatrix (const bool colordered, const int minor, const int major, const CoinBigIndex numels, const double * elem, const int * ind, const CoinBigIndex * start, const int * len, const double extraMajor, const double extraGap)`

9.56.2.4 `CoinPackedMatrix::CoinPackedMatrix (const bool colordered, const int minor, const int major, const CoinBigIndex numels, const double * elem, const int * ind, const CoinBigIndex * start, const int * len)`

9.56.2.5 `CoinPackedMatrix::CoinPackedMatrix (const bool colordered, const int * rowIndices, const int * colIndices, const double * elements, CoinBigIndex numels)`

Create packed matrix from triples.

If `colordered` is true then the created matrix will be column ordered. Duplicate matrix elements are allowed. The created matrix will have the sum of the duplicates.

For example if:

```
rowIndices[0]=2; colIndices[0]=5; elements[0]=2.0
```

```
rowIndices[1]=2; colIndices[1]=5; elements[1]=0.5
```

then the created matrix will contain a value of 2.5 in row 2 and column 5.

The matrix is created without gaps.

9.56.2.6 `CoinPackedMatrix::CoinPackedMatrix (const CoinPackedMatrix & m)`

Copy constructor.

9.56.2.7 `CoinPackedMatrix::CoinPackedMatrix (const CoinPackedMatrix & m, int extraForMajor, int extraElements, bool reverseOrdering = false)`

Copy constructor with fine tuning.

This constructor allows for the specification of an exact amount of extra space and/or reverse ordering.

`extraForMajor` is the exact number of spare major vector slots after any possible reverse ordering. If `extraForMajor < 0`, all gaps and small elements will be removed from the copy, otherwise gaps and small elements are preserved.

`extraElements` is the exact number of spare element entries.

The usual multipliers, [extraMajor_](#) and [extraGap_](#), are set to zero.

9.56.2.8 **CoinPackedMatrix::CoinPackedMatrix** (**const** **CoinPackedMatrix** & *wholeModel*, **int** *numberOfRows*, **const** **int** * *whichRows*, **int** *numberOfColumns*, **const** **int** * *whichColumns*)

Subset constructor (without gaps).

Duplicates are allowed and order is as given

9.56.2.9 **virtual** **CoinPackedMatrix::~CoinPackedMatrix** () **[virtual]**

Destructor.

9.56.3 Member Function Documentation

9.56.3.1 **double** **CoinPackedMatrix::getExtraGap** () **const** **[inline]**

Return the current setting of the extra gap.

Definition at line 89 of file CoinPackedMatrix.hpp.

9.56.3.2 **double** **CoinPackedMatrix::getExtraMajor** () **const** **[inline]**

Return the current setting of the extra major.

Definition at line 91 of file CoinPackedMatrix.hpp.

9.56.3.3 **void** **CoinPackedMatrix::reserve** (**const** **int** *newMaxMajorDim*, **const** **CoinBigIndex** *newMaxSize*, **bool** *create* = **false**)

Reserve sufficient space for appending major-ordered vectors.

If create is true, empty columns are created (for column generation)

9.56.3.4 **void** **CoinPackedMatrix::clear** ()

Clear the data, but do not free any arrays.

9.56.3.5 **bool** **CoinPackedMatrix::isColOrdered** () **const** **[inline]**

Whether the packed matrix is column major ordered or not.

Definition at line 101 of file CoinPackedMatrix.hpp.

9.56.3.6 **bool** **CoinPackedMatrix::hasGaps** () **const** **[inline]**

Whether the packed matrix has gaps or not.

Definition at line 104 of file CoinPackedMatrix.hpp.

9.56.3.7 **CoinBigIndex** **CoinPackedMatrix::getNumElements** () **const** **[inline]**

Number of entries in the packed matrix.

Definition at line 107 of file CoinPackedMatrix.hpp.

9.56.3.8 **int** **CoinPackedMatrix::getNumCols** () **const** **[inline]**

Number of columns.

Definition at line 110 of file CoinPackedMatrix.hpp.

9.56.3.9 `int CoinPackedMatrix::getNumRows () const` `[inline]`

Number of rows.

Definition at line 114 of file `CoinPackedMatrix.hpp`.

9.56.3.10 `const double* CoinPackedMatrix::getElements () const` `[inline]`

A vector containing the elements in the packed matrix.

Returns `#elements_`. Note that there might be gaps in this vector, entries that do not belong to any major-dimension vector. To get the actual elements one should look at this vector together with `vectorStarts` ([start_](#)) and `vectorLengths` ([length_](#)).

Definition at line 124 of file `CoinPackedMatrix.hpp`.

9.56.3.11 `const int* CoinPackedMatrix::getIndices () const` `[inline]`

A vector containing the minor indices of the elements in the packed matrix.

Returns `index_`. Note that there might be gaps in this list, entries that do not belong to any major-dimension vector. To get the actual elements one should look at this vector together with `vectorStarts` ([start_](#)) and `vectorLengths` ([length_](#)).

Definition at line 134 of file `CoinPackedMatrix.hpp`.

9.56.3.12 `int CoinPackedMatrix::getSizeVectorStarts () const` `[inline]`

The size of the `vectorStarts` array.

See [start_](#).

Definition at line 140 of file `CoinPackedMatrix.hpp`.

9.56.3.13 `int CoinPackedMatrix::getSizeVectorLengths () const` `[inline]`

The size of the `vectorLengths` array.

See [length_](#).

Definition at line 147 of file `CoinPackedMatrix.hpp`.

9.56.3.14 `const CoinBigIndex* CoinPackedMatrix::getVectorStarts () const` `[inline]`

The positions where the major-dimension vectors start in elements and indices.

See [start_](#).

Definition at line 154 of file `CoinPackedMatrix.hpp`.

9.56.3.15 `const int* CoinPackedMatrix::getVectorLengths () const` `[inline]`

The lengths of the major-dimension vectors.

See [length_](#).

Definition at line 160 of file `CoinPackedMatrix.hpp`.

9.56.3.16 `CoinBigIndex CoinPackedMatrix::getVectorFirst (const int i) const` `[inline]`

The position of the first element in the *i*'th major-dimension vector.

Definition at line 164 of file `CoinPackedMatrix.hpp`.

9.56.3.17 CoinBigIndex CoinPackedMatrix::getVectorLast (const int *i*) const [inline]

The position of the last element (well, one entry *past* the last) in the *i*'th major-dimension vector.

Definition at line 173 of file CoinPackedMatrix.hpp.

9.56.3.18 int CoinPackedMatrix::getVectorSize (const int *i*) const [inline]

The length of *i*'th vector.

Definition at line 181 of file CoinPackedMatrix.hpp.

9.56.3.19 const CoinShallowPackedVector CoinPackedMatrix::getVector (int *i*) const [inline]

Return the *i*'th vector in matrix.

Definition at line 190 of file CoinPackedMatrix.hpp.

9.56.3.20 int* CoinPackedMatrix::getMajorIndices () const

Returns an array containing major indices.

The array is getNumElements long and if [getVectorStarts\(\)](#) is 0,2,5 then the array would start 0,0,1,1,1,2... This method is provided to go back from a packed format to a triple format. It returns NULL if there are gaps in matrix so user should use [removeGaps\(\)](#) if there are any gaps. It does this as this array has to match [getElements\(\)](#) and [getIndices\(\)](#) and because it makes no sense otherwise. The returned array is allocated with `new int []`, free it with `delete []`.

9.56.3.21 void CoinPackedMatrix::setDimensions (int *numrows*, int *numcols*)

Set the dimensions of the matrix.

The method name is deceptive; the effect is to append empty columns and/or rows to the matrix to reach the specified dimensions. A negative number for either dimension means that that dimension doesn't change. An exception will be thrown if the specified dimensions are smaller than the current dimensions.

9.56.3.22 void CoinPackedMatrix::setExtraGap (const double *newGap*)

Set the extra gap to be allocated to the specified value.

9.56.3.23 void CoinPackedMatrix::setExtraMajor (const double *newMajor*)

Set the extra major to be allocated to the specified value.

9.56.3.24 void CoinPackedMatrix::appendCol (const CoinPackedVectorBase & *vec*)

Append a column to the end of the matrix.

When compiled with COIN_DEBUG defined this method throws an exception if the column vector specifies a nonexistent row index. Otherwise the method assumes that every index fits into the matrix.

9.56.3.25 void CoinPackedMatrix::appendCol (const int *vecsize*, const int * *vecind*, const double * *vecelem*)

Append a column to the end of the matrix.

When compiled with COIN_DEBUG defined this method throws an exception if the column vector specifies a nonexistent row index. Otherwise the method assumes that every index fits into the matrix.

9.56.3.26 void CoinPackedMatrix::appendCols (const int *numcols*, const CoinPackedVectorBase *const * *cols*)

Append a set of columns to the end of the matrix.

When compiled with COIN_DEBUG defined this method throws an exception if any of the column vectors specify a nonexistent row index. Otherwise the method assumes that every index fits into the matrix.

9.56.3.27 `int CoinPackedMatrix::appendCols (const int numcols, const CoinBigIndex * columnStarts, const int * row, const double * element, int numberOfRows = -1)`

Append a set of columns to the end of the matrix.

Returns the number of errors (nonexistent or duplicate row index). No error checking is performed if `numberOfRows < 0`.

9.56.3.28 `void CoinPackedMatrix::appendRow (const CoinPackedVectorBase & vec)`

Append a row to the end of the matrix.

When compiled with COIN_DEBUG defined this method throws an exception if the row vector specifies a nonexistent column index. Otherwise the method assumes that every index fits into the matrix.

9.56.3.29 `void CoinPackedMatrix::appendRow (const int vecsize, const int * vecind, const double * vecelem)`

Append a row to the end of the matrix.

When compiled with COIN_DEBUG defined this method throws an exception if the row vector specifies a nonexistent column index. Otherwise the method assumes that every index fits into the matrix.

9.56.3.30 `void CoinPackedMatrix::appendRows (const int numRows, const CoinPackedVectorBase *const * rows)`

Append a set of rows to the end of the matrix.

When compiled with COIN_DEBUG defined this method throws an exception if any of the row vectors specify a nonexistent column index. Otherwise the method assumes that every index fits into the matrix.

9.56.3.31 `int CoinPackedMatrix::appendRows (const int numRows, const CoinBigIndex * rowStarts, const int * column, const double * element, int numberOfColumns = -1)`

Append a set of rows to the end of the matrix.

Returns the number of errors (nonexistent or duplicate column index). No error checking is performed if `numberOfColumns < 0`.

9.56.3.32 `void CoinPackedMatrix::rightAppendPackedMatrix (const CoinPackedMatrix & matrix)`

Append the argument to the "right" of the current matrix.

Imagine this as adding new columns (don't worry about how the matrices are ordered, that is taken care of). An exception is thrown if the number of rows is different in the matrices.

9.56.3.33 `void CoinPackedMatrix::bottomAppendPackedMatrix (const CoinPackedMatrix & matrix)`

Append the argument to the "bottom" of the current matrix.

Imagine this as adding new rows (don't worry about how the matrices are ordered, that is taken care of). An exception is thrown if the number of columns is different in the matrices.

9.56.3.34 `void CoinPackedMatrix::deleteCols (const int numDel, const int * indDel)`

Delete the columns whose indices are listed in `indDel`.

9.56.3.35 void CoinPackedMatrix::deleteRows (const int *numDel*, const int * *indDel*)

Delete the rows whose indices are listed in *indDel*.

9.56.3.36 void CoinPackedMatrix::replaceVector (const int *index*, const int *numReplace*, const double * *newElements*)

Replace the elements of a vector.

The indices remain the same. At most the number specified will be replaced. The index is between 0 and major dimension of matrix

9.56.3.37 void CoinPackedMatrix::modifyCoefficient (int *row*, int *column*, double *newElement*, bool *keepZero* = false)

Modify one element of packed matrix.

An element may be added. This works for either ordering If the new element is zero it will be deleted unless *keepZero* true

9.56.3.38 double CoinPackedMatrix::getCoefficient (int *row*, int *column*) const

Return one element of packed matrix.

This works for either ordering If it is not present will return 0.0

9.56.3.39 int CoinPackedMatrix::compress (double *threshold*)

Eliminate all elements in matrix whose absolute value is less than threshold.

The column starts are not affected. Returns number of elements eliminated. Elements eliminated are at end of each vector

9.56.3.40 int CoinPackedMatrix::eliminateDuplicates (double *threshold*)

Eliminate all duplicate AND small elements in matrix The column starts are not affected.

Returns number of elements eliminated.

9.56.3.41 void CoinPackedMatrix::orderMatrix ()

Sort all columns so indices are increasing.in each column.

9.56.3.42 int CoinPackedMatrix::cleanMatrix (double *threshold* = 1.0e-20)

Really clean up matrix.

a) eliminate all duplicate AND small elements in matrix b) remove all gaps and set *extraGap_* and *extraMajor_* to 0.0 c) reallocate arrays and make max lengths equal to lengths d) orders elements returns number of elements eliminated

9.56.3.43 void CoinPackedMatrix::removeGaps (double *removeValue* = -1.0)

Remove the gaps from the matrix if there were any Can also remove small elements fabs() <= *removeValue*.

9.56.3.44 void CoinPackedMatrix::submatrixOf (const CoinPackedMatrix & *matrix*, const int *numMajor*, const int * *indMajor*)

Extract a submatrix from matrix.

Those major-dimension vectors of the matrix comprise the submatrix whose indices are given in the arguments. Does not allow duplicates.

9.56.3.45 `void CoinPackedMatrix::submatrixOfWithDuplicates (const CoinPackedMatrix & matrix, const int numMajor, const int * indMajor)`

Extract a submatrix from matrix.

Those major-dimension vectors of the matrix comprise the submatrix whose indices are given in the arguments. Allows duplicates and keeps order.

9.56.3.46 `void CoinPackedMatrix::copyOf (const CoinPackedMatrix & rhs)`

Copy method.

This method makes an exact replica of the argument, including the extra space parameters.

9.56.3.47 `void CoinPackedMatrix::copyOf (const bool colorordered, const int minor, const int major, const CoinBigIndex numels, const double * elem, const int * ind, const CoinBigIndex * start, const int * len, const double extraMajor = 0.0, const double extraGap = 0.0)`

Copy the arguments to the matrix.

If `len` is a NULL pointer then the matrix is assumed to have no gaps in it and `len` will be created accordingly.

9.56.3.48 `void CoinPackedMatrix::copyReuseArrays (const CoinPackedMatrix & rhs)`

Copy method.

This method makes an exact replica of the argument, including the extra space parameters. If there is room it will re-use arrays

9.56.3.49 `void CoinPackedMatrix::reverseOrderedCopyOf (const CoinPackedMatrix & rhs)`

Make a reverse-ordered copy.

This method makes an exact replica of the argument with the major vector orientation changed from row (column) to column (row). The extra space parameters are also copied and reversed. (Cf. [reverseOrdering](#), which does the same thing in place.)

9.56.3.50 `void CoinPackedMatrix::assignMatrix (const bool colorordered, const int minor, const int major, const CoinBigIndex numels, double * elem, int * ind, CoinBigIndex * start, int * len, const int maxmajor = -1, const CoinBigIndex maxsize = -1)`

Assign the arguments to the matrix.

If `len` is a NULL pointer then the matrix is assumed to have no gaps in it and `len` will be created accordingly.

NOTE 1: After this method returns the pointers passed to the method will be NULL pointers!

NOTE 2: When the matrix is eventually destructed the arrays will be deleted by `delete[]`. Hence one should use this method ONLY if all array swere allocated by `new[]`!

9.56.3.51 `CoinPackedMatrix& CoinPackedMatrix::operator= (const CoinPackedMatrix & rhs)`

Assignment operator.

This copies out the data, but uses the current matrix's extra space parameters.

9.56.3.52 `void CoinPackedMatrix::reverseOrdering ()`

Reverse the ordering of the packed matrix.

Change the major vector orientation of the matrix data structures from row (column) to column (row). (Cf. [reverse-](#)

[OrderedCopyOf](#), which does the same thing but produces a new matrix.)

9.56.3.53 void CoinPackedMatrix::transpose ()

Transpose the matrix.

Note

If you start with a column-ordered matrix and invoke transpose, you will have a row-ordered transposed matrix. To change the major vector orientation (e.g., to transform a column-ordered matrix to a column-ordered transposed matrix), invoke [transpose\(\)](#) followed by [reverseOrdering\(\)](#).

9.56.3.54 void CoinPackedMatrix::swap (CoinPackedMatrix & matrix)

Swap the content of two packed matrices.

9.56.3.55 void CoinPackedMatrix::times (const double * x, double * y) const

Return $A * x$ in y .

Precondition

x must be of size `numColumns()`
 y must be of size `numRows()`

9.56.3.56 void CoinPackedMatrix::times (const CoinPackedVectorBase & x, double * y) const

Return $A * x$ in y .

Same as the previous method, just x is given in the form of a packed vector.

9.56.3.57 void CoinPackedMatrix::transposeTimes (const double * x, double * y) const

Return $x * A$ in y .

Precondition

x must be of size `numRows()`
 y must be of size `numColumns()`

9.56.3.58 void CoinPackedMatrix::transposeTimes (const CoinPackedVectorBase & x, double * y) const

Return $x * A$ in y .

Same as the previous method, just x is given in the form of a packed vector.

9.56.3.59 int* CoinPackedMatrix::countOrthoLength () const

Count the number of entries in every minor-dimension vector and return an array containing these lengths.

The returned array is allocated with `new int[]`, free it with `delete[]`.

9.56.3.60 void CoinPackedMatrix::countOrthoLength (int * counts) const

Count the number of entries in every minor-dimension vector and fill in an array containing these lengths.

9.56.3.61 `int CoinPackedMatrix::getMajorDim () const [inline]`

Major dimension.

For row ordered matrix this would be the number of rows.

Definition at line 498 of file CoinPackedMatrix.hpp.

9.56.3.62 `void CoinPackedMatrix::setMajorDim (int value) [inline]`

Set major dimension.

For row ordered matrix this would be the number of rows. Use with great care.

Definition at line 501 of file CoinPackedMatrix.hpp.

9.56.3.63 `int CoinPackedMatrix::getMinorDim () const [inline]`

Minor dimension.

For row ordered matrix this would be the number of columns.

Definition at line 504 of file CoinPackedMatrix.hpp.

9.56.3.64 `void CoinPackedMatrix::setMinorDim (int value) [inline]`

Set minor dimension.

For row ordered matrix this would be the number of columns. Use with great care.

Definition at line 507 of file CoinPackedMatrix.hpp.

9.56.3.65 `int CoinPackedMatrix::getMaxMajorDim () const [inline]`

Current maximum for major dimension.

For row ordered matrix this many rows can be added without reallocating the vector related to the major dimension (*start_* and *length_*).

Definition at line 511 of file CoinPackedMatrix.hpp.

9.56.3.66 `void CoinPackedMatrix::dumpMatrix (const char * fname = NULL) const`

Dump the matrix on stdout.

When in dire straits this method can help.

9.56.3.67 `void CoinPackedMatrix::printMatrixElement (const int row_val, const int col_val) const`

Print a single matrix element.

9.56.3.68 `void CoinPackedMatrix::appendMajorVector (const CoinPackedVectorBase & vec)`

Append a major-dimension vector to the end of the matrix.

9.56.3.69 `void CoinPackedMatrix::appendMajorVector (const int vecsize, const int * vecind, const double * vecelem)`

Append a major-dimension vector to the end of the matrix.

9.56.3.70 `void CoinPackedMatrix::appendMajorVectors (const int numvecs, const CoinPackedVectorBase *const * vecs)`

Append several major-dimension vectors to the end of the matrix.

9.56.3.71 **void** CoinPackedMatrix::appendMinorVector (**const** CoinPackedVectorBase & *vec*)

Append a minor-dimension vector to the end of the matrix.

9.56.3.72 **void** CoinPackedMatrix::appendMinorVector (**const** int *vecsize*, **const** int * *vecind*, **const** double * *vecelem*)

Append a minor-dimension vector to the end of the matrix.

9.56.3.73 **void** CoinPackedMatrix::appendMinorVectors (**const** int *numvecs*, **const** CoinPackedVectorBase ***const** * *vecs*)

Append several minor-dimension vectors to the end of the matrix.

9.56.3.74 **void** CoinPackedMatrix::appendMinorFast (**const** int *number*, **const** CoinBigIndex * *starts*, **const** int * *index*, **const** double * *element*)

Append a set of rows (columns) to the end of a column (row) ordered matrix.

This case is when we know there are no gaps and majorDim_ will not change.

Todo This method really belongs in the group of protected methods with [appendMinor](#); there are no safeties here even with COIN_DEBUG. Apparently this method was needed in ClpPackedMatrix and giving it proper visibility was too much trouble. Should be moved.

9.56.3.75 **void** CoinPackedMatrix::majorAppendSameOrdered (**const** CoinPackedMatrix & *matrix*)

Append the columns of the argument to the right end of this matrix.

Precondition

```
minorDim_ == matrix.minorDim_
```

This method throws an exception if the minor dimensions are not the same.

9.56.3.76 **void** CoinPackedMatrix::minorAppendSameOrdered (**const** CoinPackedMatrix & *matrix*)

Append the columns of the argument to the bottom end of this matrix.

Precondition

```
majorDim_ == matrix.majorDim_
```

This method throws an exception if the major dimensions are not the same.

9.56.3.77 **void** CoinPackedMatrix::majorAppendOrthoOrdered (**const** CoinPackedMatrix & *matrix*)

Append the rows of the argument to the right end of this matrix.

Precondition

```
minorDim_ == matrix.majorDim_
```

This method throws an exception if the minor dimension of the current matrix is not the same as the major dimension of the argument matrix.

9.56.3.78 **void** CoinPackedMatrix::minorAppendOrthoOrdered (**const** CoinPackedMatrix & *matrix*)

Append the rows of the argument to the bottom end of this matrix.

Precondition

```
majorDim_ == matrix.minorDim_
```

This method throws an exception if the major dimension of the current matrix is not the same as the minor dimension of the argument matrix.

9.56.3.79 void CoinPackedMatrix::deleteMajorVectors (const int *numDel*, const int * *indDel*)

Delete the major-dimension vectors whose indices are listed in *indDel*.

9.56.3.80 void CoinPackedMatrix::deleteMinorVectors (const int *numDel*, const int * *indDel*)

Delete the minor-dimension vectors whose indices are listed in *indDel*.

9.56.3.81 void CoinPackedMatrix::timesMajor (const double * *x*, double * *y*) const

Return $A * x$ (multiplied from the "right" direction) in *y*.

Precondition

x must be of size `majorDim()`

y must be of size `minorDim()`

9.56.3.82 void CoinPackedMatrix::timesMajor (const CoinPackedVectorBase & *x*, double * *y*) const

Return $A * x$ (multiplied from the "right" direction) in *y*.

Same as the previous method, just *x* is given in the form of a packed vector.

9.56.3.83 void CoinPackedMatrix::timesMinor (const double * *x*, double * *y*) const

Return $A * x$ (multiplied from the "right" direction) in *y*.

Precondition

x must be of size `minorDim()`

y must be of size `majorDim()`

9.56.3.84 void CoinPackedMatrix::timesMinor (const CoinPackedVectorBase & *x*, double * *y*) const

Return $A * x$ (multiplied from the "right" direction) in *y*.

Same as the previous method, just *x* is given in the form of a packed vector.

9.56.3.85 template<class FloatEqual > bool CoinPackedMatrix::isEquivalent (const CoinPackedMatrix & *rhs*, const FloatEqual & *eq*) const [inline]

Test for equivalence.

Two matrices are equivalent if they are both row- or column-ordered, they have the same dimensions, and each (major) vector is equivalent. The operator used to test for equality can be specified using the `FloatEqual` template parameter.

Definition at line 656 of file `CoinPackedMatrix.hpp`.

9.56.3.86 bool CoinPackedMatrix::isEquivalent2 (const CoinPackedMatrix & *rhs*) const

Test for equivalence and report differences.

Equivalence is defined as for [isEquivalent](#). In addition, this method will print differences to `std::cerr`. Intended for use in unit tests and for debugging.

9.56.3.87 `bool CoinPackedMatrix::isEquivalent (const CoinPackedMatrix & rhs) const`

Test for equivalence.

The test for element equality is the default [CoinRelFltEq](#) operator.

9.56.3.88 `double* CoinPackedMatrix::getMutableElements () const` `[inline]`

A vector containing the elements in the packed matrix.

Note that there might be gaps in this list, entries that do not belong to any major-dimension vector. To get the actual elements one should look at this vector together with [start_](#) and [length_](#).

Definition at line 709 of file CoinPackedMatrix.hpp.

9.56.3.89 `int* CoinPackedMatrix::getMutableIndices () const` `[inline]`

A vector containing the minor indices of the elements in the packed matrix.

Note that there might be gaps in this list, entries that do not belong to any major-dimension vector. To get the actual elements one should look at this vector together with [start_](#) and [length_](#).

Definition at line 715 of file CoinPackedMatrix.hpp.

9.56.3.90 `CoinBigIndex* CoinPackedMatrix::getMutableVectorStarts () const` `[inline]`

The positions where the major-dimension vectors start in [element_](#) and [index_](#).

Definition at line 719 of file CoinPackedMatrix.hpp.

9.56.3.91 `int* CoinPackedMatrix::getMutableVectorLengths () const` `[inline]`

The lengths of the major-dimension vectors.

Definition at line 721 of file CoinPackedMatrix.hpp.

9.56.3.92 `void CoinPackedMatrix::setNumElements (CoinBigIndex value)` `[inline]`

Change the size of the bulk store after modifying - be careful.

Definition at line 723 of file CoinPackedMatrix.hpp.

9.56.3.93 `void CoinPackedMatrix::nullElementArray ()` `[inline]`

NULLify element array.

Used when space is very tight. Does not free the space!

Definition at line 729 of file CoinPackedMatrix.hpp.

9.56.3.94 `void CoinPackedMatrix::nullStartArray ()` `[inline]`

NULLify start array.

Used when space is very tight. Does not free the space!

Definition at line 735 of file CoinPackedMatrix.hpp.

9.56.3.95 `void CoinPackedMatrix::nullLengthArray ()` `[inline]`

NULLify length array.

Used when space is very tight. Does not free the space!

Definition at line 741 of file CoinPackedMatrix.hpp.

9.56.3.96 `void CoinPackedMatrix::nullIndexArray () [inline]`

NULLify index array.

Used when space is very tight. Does not free the space!

Definition at line 747 of file CoinPackedMatrix.hpp.

9.56.3.97 `int CoinPackedMatrix::verifyMtx (int verbosity = 1, bool zeroesAreError = false) const`

Scan the matrix for anomalies.

Returns the number of anomalies. Scans the structure for gaps, obviously bogus indices and coefficients, and inconsistencies. Gaps are not an error unless `hasGaps()` says the matrix should be gap-free. Zeroes are not an error unless `zeroesAreError` is set to true.

Values for verbosity are:

- 0: No messages, just the return value
- 1: Messages about errors
- 2: If there are no errors, a message indicating the matrix was checked is printed (positive confirmation).
- 3: Adds a bit more information about the matrix.
- 4: Prints warnings about zeroes even if they're not considered errors.

Obviously bogus coefficients are coefficients that are NaN or have absolute value greater than 1e50. Zeros have absolute value less than 1e-50.

9.56.3.98 `void CoinPackedMatrix::gutsOfDestructor () [protected]`

9.56.3.99 `void CoinPackedMatrix::gutsOfCopyOf (const bool colordered, const int minor, const int major, const CoinBigIndex numels, const double * elem, const int * ind, const CoinBigIndex * start, const int * len, const double extraMajor = 0.0, const double extraGap = 0.0) [protected]`

9.56.3.100 `void CoinPackedMatrix::gutsOfCopyOfNoGaps (const bool colordered, const int minor, const int major, const double * elem, const int * ind, const CoinBigIndex * start) [protected]`

When no gaps we can do faster.

9.56.3.101 `void CoinPackedMatrix::gutsOfOpEqual (const bool colordered, const int minor, const int major, const CoinBigIndex numels, const double * elem, const int * ind, const CoinBigIndex * start, const int * len) [protected]`

9.56.3.102 `void CoinPackedMatrix::resizeForAddingMajorVectors (const int numVec, const int * lengthVec) [protected]`

9.56.3.103 `void CoinPackedMatrix::resizeForAddingMinorVectors (const int * addedEntries) [protected]`

9.56.3.104 `int CoinPackedMatrix::appendMajor (const int number, const CoinBigIndex * starts, const int * index, const double * element, int numberOther = -1) [protected]`

Append a set of rows (columns) to the end of a row (column) ordered matrix.

If `numberOther > 0` the method will check if any of the new rows (columns) contain duplicate indices or invalid indices and return the number of errors. A valid minor index must satisfy

`0 <= k < numberOther`

If `numberOther < 0` no checking is performed.

9.56.3.105 `int CoinPackedMatrix::appendMinor (const int number, const CoinBigIndex * starts, const int * index, const double * element, int numberOther = -1)` [protected]

Append a set of rows (columns) to the end of a column (row) ordered matrix.

If `numberOther > 0` the method will check if any of the new rows (columns) contain duplicate indices or indices outside the current range for the major dimension and return the number of violations. If `numberOther <= 0` the major dimension will be expanded as necessary and there are no checks for duplicate indices.

9.56.4 Friends And Related Function Documentation

9.56.4.1 `void CoinPackedMatrixUnitTest ()` [friend]

Test the methods in the [CoinPackedMatrix](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

9.56.5 Member Data Documentation

9.56.5.1 `bool CoinPackedMatrix::colOrdered_` [protected]

A flag indicating whether the matrix is column or row major ordered.

Definition at line 900 of file `CoinPackedMatrix.hpp`.

9.56.5.2 `double CoinPackedMatrix::extraGap_` [protected]

This much times more space should be allocated for each major-dimension vector (with respect to the number of entries in the vector) when the matrix is resized.

The purpose of these gaps is to allow fast insertion of new minor-dimension vectors.

Definition at line 905 of file `CoinPackedMatrix.hpp`.

9.56.5.3 `double CoinPackedMatrix::extraMajor_` [protected]

his much times more space should be allocated for major-dimension vectors when the matrix is resized.

The purpose of these gaps is to allow fast addition of new major-dimension vectors.

Definition at line 909 of file `CoinPackedMatrix.hpp`.

9.56.5.4 `double* CoinPackedMatrix::element_` [protected]

List of nonzero element values.

The entries in the gaps between major-dimension vectors are undefined.

Definition at line 913 of file `CoinPackedMatrix.hpp`.

9.56.5.5 `int* CoinPackedMatrix::index_` [protected]

List of nonzero element minor-dimension indices.

The entries in the gaps between major-dimension vectors are undefined.

Definition at line 916 of file CoinPackedMatrix.hpp.

9.56.5.6 `CoinBigIndex* CoinPackedMatrix::start_` [protected]

Starting positions of major-dimension vectors.

Definition at line 918 of file CoinPackedMatrix.hpp.

9.56.5.7 `int* CoinPackedMatrix::length_` [protected]

Lengths of major-dimension vectors.

Definition at line 920 of file CoinPackedMatrix.hpp.

9.56.5.8 `int CoinPackedMatrix::majorDim_` [protected]

number of vectors in matrix

Definition at line 923 of file CoinPackedMatrix.hpp.

9.56.5.9 `int CoinPackedMatrix::minorDim_` [protected]

size of other dimension

Definition at line 925 of file CoinPackedMatrix.hpp.

9.56.5.10 `CoinBigIndex CoinPackedMatrix::size_` [protected]

the number of nonzero entries

Definition at line 927 of file CoinPackedMatrix.hpp.

9.56.5.11 `int CoinPackedMatrix::maxMajorDim_` [protected]

max space allocated for major-dimension

Definition at line 930 of file CoinPackedMatrix.hpp.

9.56.5.12 `CoinBigIndex CoinPackedMatrix::maxSize_` [protected]

max space allocated for entries

Definition at line 932 of file CoinPackedMatrix.hpp.

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

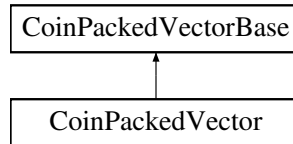
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPackedMatrix.hpp](#)

9.57 CoinPackedVector Class Reference

Sparse Vector.

```
#include <CoinPackedVector.hpp>
```

Inheritance diagram for CoinPackedVector:



Public Member Functions

Get methods.

- virtual int [getNumElements](#) () const
Get the size.
- virtual const int * [getIndices](#) () const
Get indices of elements.
- virtual const double * [getElements](#) () const
Get element values.
- int * [getIndices](#) ()
Get indices of elements.
- int [getVectorNumElements](#) () const
Get the size.
- const int * [getVectorIndices](#) () const
Get indices of elements.
- const double * [getVectorElements](#) () const
Get element values.
- double * [getElements](#) ()
Get element values.
- const int * [getOriginalPosition](#) () const
*Get pointer to int * vector of original postions.*

Set methods

- void [clear](#) ()
Reset the vector (as if were just created an empty vector)
- [CoinPackedVector](#) & [operator=](#) (const [CoinPackedVector](#) &)
Assignment operator.
- [CoinPackedVector](#) & [operator=](#) (const [CoinPackedVectorBase](#) &rhs)
Assignment operator from a [CoinPackedVectorBase](#).
- void [assignVector](#) (int size, int *inds, double *elems, bool [testForDuplicateIndex](#)=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Assign the ownership of the arguments to this vector.
- void [setVector](#) (int size, const int *inds, const double *elems, bool [testForDuplicateIndex](#)=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Set vector size, indices, and elements.
- void [setConstant](#) (int size, const int *inds, double elems, bool [testForDuplicateIndex](#)=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Elements set to have the same scalar value.
- void [setFull](#) (int size, const double *elems, bool [testForDuplicateIndex](#)=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Indices are not specified and are taken to be 0,1,...,size-1.
- void [setFullNonZero](#) (int size, const double *elems, bool [testForDuplicateIndex](#)=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Indices are not specified and are taken to be 0,1,...,size-1, but only where non zero.
- void [setElement](#) (int index, double element)

- *Set an existing element in the packed vector The first argument is the "index" into the elements() array.*
- **void insert** (int index, double element)
Insert an element into the vector.
- **void append** (const **CoinPackedVectorBase** &caboose)
*Append a **CoinPackedVector** to the end.*
- **void swap** (int i, int j)
Swap values in positions i and j of indices and elements.
- **void truncate** (int newSize)
Resize the packed vector to be the first newSize elements.

Arithmetic operators.

- **void operator+=** (double value)
add value to every entry
- **void operator-=** (double value)
subtract value from every entry
- **void operator*=** (double value)
multiply every entry by value
- **void operator/=** (double value)
divide every entry by value

Sorting

- **template<class CoinCompare3 >**
void sort (const CoinCompare3 &tc)
Sort the packed storage vector.
- **void sortIncrIndex** ()
- **void sortDecrIndex** ()
- **void sortIncrElement** ()
- **void sortDecrElement** ()
- **void sortOriginalOrder** ()
Sort in original order.

Memory usage

- **void reserve** (int n)
Reserve space.
- **int capacity** () const
capacity returns the size which could be accomodated without having to reallocate storage.

Constructors and destructors

- **CoinPackedVector** (bool testForDuplicateIndex=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Default constructor.
- **CoinPackedVector** (int size, const int *inds, const double *elems, bool testForDuplicateIndex=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Alternate Constructors - set elements to vector of doubles.
- **CoinPackedVector** (int capacity, int size, int *inds, double *elems, bool testForDuplicateIndex=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Alternate Constructors - set elements to vector of doubles.
- **CoinPackedVector** (int size, const int *inds, double element, bool testForDuplicateIndex=COIN_DEFAULT_VALUE_FOR_DUPLICATE)
Alternate Constructors - set elements to same scalar value.
- **CoinPackedVector** (int size, const double *elements, bool testForDuplicateIndex=COIN_DEFAULT_VALUE_FOR_DUPLICATE)

Alternate Constructors - construct full storage with indices 0 through size-1.

- [CoinPackedVector](#) (const [CoinPackedVector](#) &)
Copy constructor.
- [CoinPackedVector](#) (const [CoinPackedVectorBase](#) &rhs)
Copy constructor from a PackedVectorBase.
- virtual [~CoinPackedVector](#) ()
Destructor.

Friends

- [void CoinPackedVectorUnitTest](#) ()
A function that tests the methods in the [CoinPackedVector](#) class.

Additional Inherited Members

9.57.1 Detailed Description

Sparse Vector.

Stores vector of indices and associated element values. Supports sorting of vector while maintaining the original indices.

Here is a sample usage:

```
const int ne = 4;
int inx[ne] = { 1, 4, 0, 2 }
double el[ne] = { 10., 40., 1., 50. }

// Create vector and set its value
CoinPackedVector r(ne,inx,el);

// access each index and element
assert( r.indices() [0]== 1 );
assert( r.elements() [0]==10. );
assert( r.indices() [1]== 4 );
assert( r.elements() [1]==40. );
assert( r.indices() [2]== 0 );
assert( r.elements() [2]== 1. );
assert( r.indices() [3]== 2 );
assert( r.elements() [3]==50. );

// access original position of index
assert( r.originalPosition() [0]==0 );
assert( r.originalPosition() [1]==1 );
assert( r.originalPosition() [2]==2 );
assert( r.originalPosition() [3]==3 );

// access as a full storage vector
assert( r[ 0]==1. );
assert( r[ 1]==10.);
assert( r[ 2]==50.);
assert( r[ 3]==0. );
assert( r[ 4]==40.);

// sort Elements in increasing order
r.sortIncrElement();

// access each index and element
assert( r.indices() [0]== 0 );
assert( r.elements() [0]== 1. );
assert( r.indices() [1]== 1 );
assert( r.elements() [1]==10. );
assert( r.indices() [2]== 4 );
```

```

assert( r.elements()[2]==40. );
assert( r.indices ()[3]== 2 );
assert( r.elements()[3]==50. );

// access original position of index
assert( r.originalPosition()[0]==2 );
assert( r.originalPosition()[1]==0 );
assert( r.originalPosition()[2]==1 );
assert( r.originalPosition()[3]==3 );

// access as a full storage vector
assert( r[ 0]==1. );
assert( r[ 1]==10.);
assert( r[ 2]==50.);
assert( r[ 3]==0. );
assert( r[ 4]==40.);

// Restore original sort order
r.sortOriginalOrder();

assert( r.indices ()[0]== 1 );
assert( r.elements()[0]==10. );
assert( r.indices ()[1]== 4 );
assert( r.elements()[1]==40. );
assert( r.indices ()[2]== 0 );
assert( r.elements()[2]== 1. );
assert( r.indices ()[3]== 2 );
assert( r.elements()[3]==50. );

// Tests for equality and equivalence
CoinPackedVector r1;
r1=r;
assert( r==r1 );
assert( r.equivalent(r1) );
r.sortIncrElement();
assert( r!=r1 );
assert( r.equivalent(r1) );

// Add packed vectors.
// Similarly for subtraction, multiplication,
// and division.
CoinPackedVector add = r + r1;
assert( add[0] == 1.+ 1. );
assert( add[1] == 10.+10. );
assert( add[2] == 50.+50. );
assert( add[3] == 0.+ 0. );
assert( add[4] == 40.+40. );

assert( r.sum() == 10.+40.+1.+50. );

```

Definition at line 123 of file CoinPackedVector.hpp.

9.57.2 Constructor & Destructor Documentation

9.57.2.1 CoinPackedVector::CoinPackedVector (bool *testForDuplicateIndex* = COIN_DEFAULT_VALUE_FOR_DUPLICATE)

Default constructor.

9.57.2.2 CoinPackedVector::CoinPackedVector (int *size*, const int * *inds*, const double * *elems*, bool *testForDuplicateIndex* = COIN_DEFAULT_VALUE_FOR_DUPLICATE)

Alternate Constructors - set elements to vector of doubles.

This constructor copies the vectors provided as parameters.

9.57.2.3 `CoinPackedVector::CoinPackedVector (int capacity, int size, int *inds, double *elems, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Alternate Constructors - set elements to vector of doubles.

This constructor takes ownership of the vectors passed as parameters. *inds* and *elems* will be NULL on return.

9.57.2.4 `CoinPackedVector::CoinPackedVector (int size, const int * inds, double element, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Alternate Constructors - set elements to same scalar value.

9.57.2.5 `CoinPackedVector::CoinPackedVector (int size, const double * elements, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Alternate Constructors - construct full storage with indices 0 through size-1.

9.57.2.6 `CoinPackedVector::CoinPackedVector (const CoinPackedVector &)`

Copy constructor.

9.57.2.7 `CoinPackedVector::CoinPackedVector (const CoinPackedVectorBase & rhs)`

Copy constructor *from a PackedVectorBase*.

9.57.2.8 `virtual CoinPackedVector::~~CoinPackedVector () [virtual]`

Destructor.

9.57.3 Member Function Documentation

9.57.3.1 `virtual int CoinPackedVector::getNumElements () const [inline],[virtual]`

Get the size.

Implements [CoinPackedVectorBase](#).

Definition at line 130 of file `CoinPackedVector.hpp`.

9.57.3.2 `virtual const int* CoinPackedVector::getIndices () const [inline],[virtual]`

Get indices of elements.

Implements [CoinPackedVectorBase](#).

Definition at line 132 of file `CoinPackedVector.hpp`.

9.57.3.3 `virtual const double* CoinPackedVector::getElements () const [inline],[virtual]`

Get element values.

Implements [CoinPackedVectorBase](#).

Definition at line 134 of file `CoinPackedVector.hpp`.

9.57.3.4 `int* CoinPackedVector::getIndices () [inline]`

Get indices of elements.

Definition at line 136 of file CoinPackedVector.hpp.

9.57.3.5 `int CoinPackedVector::getVectorNumElements () const [inline]`

Get the size.

Definition at line 138 of file CoinPackedVector.hpp.

9.57.3.6 `const int* CoinPackedVector::getVectorIndices () const [inline]`

Get indices of elements.

Definition at line 140 of file CoinPackedVector.hpp.

9.57.3.7 `const double* CoinPackedVector::getVectorElements () const [inline]`

Get element values.

Definition at line 142 of file CoinPackedVector.hpp.

9.57.3.8 `double* CoinPackedVector::getElements () [inline]`

Get element values.

Definition at line 144 of file CoinPackedVector.hpp.

9.57.3.9 `const int* CoinPackedVector::getOriginalPosition () const [inline]`

Get pointer to int * vector of original positions.

If the packed vector has not been sorted then this function returns the vector: 0, 1, 2, ..., size()-1.

Definition at line 148 of file CoinPackedVector.hpp.

9.57.3.10 `void CoinPackedVector::clear ()`

Reset the vector (as if were just created an empty vector)

9.57.3.11 `CoinPackedVector& CoinPackedVector::operator= (const CoinPackedVector &)`

Assignment operator.

NOTE: This operator keeps the current `testForDuplicateIndex` setting, and after copying the data it acts accordingly.

9.57.3.12 `CoinPackedVector& CoinPackedVector::operator= (const CoinPackedVectorBase & rhs)`

Assignment operator from a [CoinPackedVectorBase](#).

NOTE: This operator keeps the current `testForDuplicateIndex` setting, and after copying the data it acts accordingly.

9.57.3.13 `void CoinPackedVector::assignVector (int size, int *& inds, double *& elems, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Assign the ownership of the arguments to this vector.

Size is the length of both the indices and elements vectors. The indices and elements vectors are copied into this class instance's member data. The last argument indicates whether this vector will have to be tested for duplicate indices.

9.57.3.14 `void CoinPackedVector::setVector (int size, const int * inds, const double * elems, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Set vector size, indices, and elements.

Size is the length of both the indices and elements vectors. The indices and elements vectors are copied into this class instance's member data. The last argument specifies whether this vector will have to be checked for duplicate indices whenever that can happen.

9.57.3.15 `void CoinPackedVector::setConstant (int size, const int * inds, double elems, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Elements set to have the same scalar value.

9.57.3.16 `void CoinPackedVector::setFull (int size, const double * elems, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Indices are not specified and are taken to be 0,1,...,size-1.

9.57.3.17 `void CoinPackedVector::setFullNonZero (int size, const double * elems, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Indices are not specified and are taken to be 0,1,...,size-1, but only where non zero.

9.57.3.18 `void CoinPackedVector::setElement (int index, double element)`

Set an existing element in the packed vector The first argument is the "index" into the elements() array.

9.57.3.19 `void CoinPackedVector::insert (int index, double element)`

Insert an element into the vector.

9.57.3.20 `void CoinPackedVector::append (const CoinPackedVectorBase & caboose)`

Append a [CoinPackedVector](#) to the end.

9.57.3.21 `void CoinPackedVector::swap (int i, int j)`

Swap values in positions i and j of indices and elements.

9.57.3.22 `void CoinPackedVector::truncate (int newSize)`

Resize the packed vector to be the first newSize elements.

Problem with truncate: what happens with origIndices_ ???

9.57.3.23 `void CoinPackedVector::operator+= (double value)`

add *value* to every entry

9.57.3.24 `void CoinPackedVector::operator-= (double value)`

subtract *value* from every entry

9.57.3.25 `void CoinPackedVector::operator*= (double value)`

multiply every entry by *value*

9.57.3.26 void CoinPackedVector::operator/= (double value)

divide every entry by value

9.57.3.27 template<class CoinCompare3 > void CoinPackedVector::sort (const CoinCompare3 & tc) [inline]

Sort the packed storage vector.

Typical usages:

```
packedVector.sort(CoinIncrIndexOrdered()); //increasing indices
packedVector.sort(CoinIncrElementOrdered()); // increasing elements
```

Definition at line 239 of file CoinPackedVector.hpp.

9.57.3.28 void CoinPackedVector::sortIncrIndex () [inline]

Definition at line 243 of file CoinPackedVector.hpp.

9.57.3.29 void CoinPackedVector::sortDecrIndex () [inline]

Definition at line 247 of file CoinPackedVector.hpp.

9.57.3.30 void CoinPackedVector::sortIncrElement () [inline]

Definition at line 251 of file CoinPackedVector.hpp.

9.57.3.31 void CoinPackedVector::sortDecrElement () [inline]

Definition at line 255 of file CoinPackedVector.hpp.

9.57.3.32 void CoinPackedVector::sortOriginalOrder ()

Sort in original order.

If the vector has been sorted, then this method restores to its original sort order.

9.57.3.33 void CoinPackedVector::reserve (int n)

Reserve space.

If one knows the eventual size of the packed vector, then it may be more efficient to reserve the space.

9.57.3.34 int CoinPackedVector::capacity () const [inline]

capacity returns the size which could be accomodated without having to reallocate storage.

Definition at line 277 of file CoinPackedVector.hpp.

9.57.4 Friends And Related Function Documentation**9.57.4.1 void CoinPackedVectorUnitTest () [friend]**

A function that tests the methods in the [CoinPackedVector](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

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

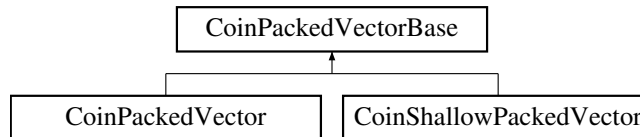
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPackedVector.hpp](#)

9.58 CoinPackedVectorBase Class Reference

Abstract base class for various sparse vectors.

```
#include <CoinPackedVectorBase.hpp>
```

Inheritance diagram for CoinPackedVectorBase:



Public Member Functions

Virtual methods that the derived classes must provide

- virtual int [getNumElements](#) () const =0
Get length of indices and elements vectors.
- virtual const int * [getIndices](#) () const =0
Get indices of elements.
- virtual const double * [getElements](#) () const =0
Get element values.

Methods related to whether duplicate-index checking is performed.

If the checking for duplicate indices is turned off, then some [CoinPackedVector](#) methods may not work correctly if there are duplicate indices.

Turning off the checking for duplicate indices may result in better run time performance.

- void [setTestForDuplicateIndex](#) (bool test) const
Set to the argument value whether to test for duplicate indices in the vector whenever they can occur.
- void [setTestForDuplicateIndexWhenTrue](#) (bool test) const
Set to the argument value whether to test for duplicate indices in the vector whenever they can occur BUT we know that right now the vector has no duplicate indices.
- bool [testForDuplicateIndex](#) () const
Returns true if the vector should be tested for duplicate indices when they can occur.
- void [setTestsOff](#) () const
Just sets test stuff false without a try etc.

Methods for getting info on the packed vector as a full vector

- double * [denseVector](#) (int denseSize) const
Get the vector as a dense vector.
- double [operator\[\]](#) (int i) const
Access the i'th element of the full storage vector.

Index methods

- int [getMaxIndex](#) () const

- *Get value of maximum index.*
- `int getMinIndex () const`
Get value of minimum index.
- `void duplicateIndex (const char *methodName=NULL, const char *className=NULL) const`
Throw an exception if there are duplicate indices.
- `bool isExistingIndex (int i) const`
Return true if the i'th element of the full storage vector exists in the packed storage vector.
- `int findIndex (int i) const`
Return the position of the i'th element of the full storage vector.

Comparison operators on two packed vectors

- `bool operator== (const CoinPackedVectorBase &rhs) const`
Equal.
- `bool operator!= (const CoinPackedVectorBase &rhs) const`
Not equal.
- `int compare (const CoinPackedVectorBase &rhs) const`
This method establishes an ordering on packed vectors.
- `template<class FloatEqual >`
`bool isEquivalent (const CoinPackedVectorBase &rhs, const FloatEqual &eq) const`
equivalent - If shallow packed vector A & B are equivalent, then they are still equivalent no matter how they are sorted.
- `bool isEquivalent (const CoinPackedVectorBase &rhs) const`

Arithmetic operators.

- `double dotProduct (const double *dense) const`
Create the dot product with a full vector.
- `double oneNorm () const`
Return the 1-norm of the vector.
- `double normSquare () const`
Return the square of the 2-norm of the vector.
- `double twoNorm () const`
Return the 2-norm of the vector.
- `double infNorm () const`
Return the infinity-norm of the vector.
- `double sum () const`
Sum elements of vector.

Protected Member Functions

Protected methods

- `void findMaxMinIndices () const`
Find Maximum and Minimum Indices.
- `std::set< int > * indexSet (const char *methodName=NULL, const char *className=NULL) const`
Return indexSetPtr_ (create it if necessary).
- `void clearIndexSet () const`
Delete the indexSet.
- `void clearBase () const`
- `void copyMaxMinIndex (const CoinPackedVectorBase &x) const`

Constructors, destructor

NOTE: All constructors are protected.

There's no need to expose them, after all, this is an abstract class.

- virtual [~CoinPackedVectorBase](#) ()

Destructor.

- [CoinPackedVectorBase](#) ()

Default constructor.

9.58.1 Detailed Description

Abstract base class for various sparse vectors.

Since this class is abstract, no object of this type can be created. The sole purpose of this class is to provide access to a *constant* packed vector. All members of this class are const methods, they can't change the object.

Definition at line 23 of file CoinPackedVectorBase.hpp.

9.58.2 Constructor & Destructor Documentation

9.58.2.1 CoinPackedVectorBase::CoinPackedVectorBase () [protected]

Default constructor.

9.58.2.2 virtual CoinPackedVectorBase::~~CoinPackedVectorBase () [virtual]

Destructor.

9.58.3 Member Function Documentation

9.58.3.1 virtual int CoinPackedVectorBase::getNumElements () const [pure virtual]

Get length of indices and elements vectors.

Implemented in [CoinPackedVector](#), and [CoinShallowPackedVector](#).

9.58.3.2 virtual const int* CoinPackedVectorBase::getIndices () const [pure virtual]

Get indices of elements.

Implemented in [CoinPackedVector](#), and [CoinShallowPackedVector](#).

9.58.3.3 virtual const double* CoinPackedVectorBase::getElements () const [pure virtual]

Get element values.

Implemented in [CoinPackedVector](#), and [CoinShallowPackedVector](#).

9.58.3.4 void CoinPackedVectorBase::setTestForDuplicateIndex (bool test) const

Set to the argument value whether to test for duplicate indices in the vector whenever they can occur.

Calling this method with `test` set to true will trigger an immediate check for duplicate indices.

9.58.3.5 void CoinPackedVectorBase::setTestForDuplicateIndexWhenTrue (bool *test*) const

Set to the argument value whether to test for duplicate indices in the vector whenever they can occur BUT we know that right now the vector has no duplicate indices.

Calling this method with `test` set to true will *not* trigger an immediate check for duplicate indices; instead, it's assumed that the result of the test will be true.

9.58.3.6 bool CoinPackedVectorBase::testForDuplicateIndex () const [inline]

Returns true if the vector should be tested for duplicate indices when they can occur.

Definition at line 63 of file `CoinPackedVectorBase.hpp`.

9.58.3.7 void CoinPackedVectorBase::setTestsOff () const [inline]

Just sets test stuff false without a try etc.

Definition at line 65 of file `CoinPackedVectorBase.hpp`.

9.58.3.8 double* CoinPackedVectorBase::denseVector (int *denseSize*) const

Get the vector as a dense vector.

The argument specifies how long this dense vector is.

NOTE: The user needs to `delete[]` this pointer after it's not needed anymore.

9.58.3.9 double CoinPackedVectorBase::operator[] (int *i*) const

Access the *i*'th element of the full storage vector.

If the *i*'th is not stored, then zero is returned. The initial use of this method has some computational and storage overhead associated with it.

NOTE: This is *very* expensive. It is probably much better to use `denseVector()`.

9.58.3.10 int CoinPackedVectorBase::getMaxIndex () const

Get value of maximum index.

9.58.3.11 int CoinPackedVectorBase::getMinIndex () const

Get value of minimum index.

9.58.3.12 void CoinPackedVectorBase::duplicateIndex (const char * *methodName* = NULL, const char * *className* = NULL) const

Throw an exception if there are duplicate indices.

9.58.3.13 bool CoinPackedVectorBase::isExistingIndex (int *i*) const

Return true if the *i*'th element of the full storage vector exists in the packed storage vector.

9.58.3.14 int CoinPackedVectorBase::findIndex (int *i*) const

Return the position of the *i*'th element of the full storage vector.

If index does not exist then -1 is returned

9.58.3.15 `bool CoinPackedVectorBase::operator==(const CoinPackedVectorBase & rhs) const`

Equal.

Returns true if vectors have same length and corresponding element of each vector is equal.

9.58.3.16 `bool CoinPackedVectorBase::operator!=(const CoinPackedVectorBase & rhs) const`

Not equal.

9.58.3.17 `int CoinPackedVectorBase::compare (const CoinPackedVectorBase & rhs) const`

This method establishes an ordering on packed vectors.

It is complete ordering, but not the same as lexicographic ordering. However, it is quick and dirty to compute and thus it is useful to keep packed vectors in a heap when all we care is to quickly check whether a particular vector is already in the heap or not. Returns negative/0/positive depending on whether `this` is smaller/equal.greater than `rhs`.

9.58.3.18 `template<class FloatEqual > bool CoinPackedVectorBase::isEquivalent (const CoinPackedVectorBase & rhs, const FloatEqual & eq) const [inline]`

equivalent - If shallow packed vector A & B are equivalent, then they are still equivalent no matter how they are sorted.

In this method the FloatEqual function operator can be specified. The default equivalence test is that the entries are relatively equal.

NOTE: This is a relatively expensive method as it sorts the two shallow packed vectors.

Definition at line 140 of file CoinPackedVectorBase.hpp.

9.58.3.19 `bool CoinPackedVectorBase::isEquivalent (const CoinPackedVectorBase & rhs) const`

9.58.3.20 `double CoinPackedVectorBase::dotProduct (const double * dense) const`

Create the dot product with a full vector.

9.58.3.21 `double CoinPackedVectorBase::oneNorm () const`

Return the 1-norm of the vector.

9.58.3.22 `double CoinPackedVectorBase::normSquare () const`

Return the square of the 2-norm of the vector.

9.58.3.23 `double CoinPackedVectorBase::twoNorm () const`

Return the 2-norm of the vector.

9.58.3.24 `double CoinPackedVectorBase::infNorm () const`

Return the infinity-norm of the vector.

9.58.3.25 `double CoinPackedVectorBase::sum () const`

Sum elements of vector.

9.58.3.26 `void CoinPackedVectorBase::findMaxMinIndices () const [protected]`

Find Maximum and Minimum Indices.

9.58.3.27 `std::set<int>* CoinPackedVectorBase::indexSet (const char * methodName = NULL, const char * className = NULL) const` [protected]

Return indexSetPtr_ (create it if necessary).

9.58.3.28 `void CoinPackedVectorBase::clearIndexSet () const` [protected]

Delete the indexSet.

9.58.3.29 `void CoinPackedVectorBase::clearBase () const` [protected]

9.58.3.30 `void CoinPackedVectorBase::copyMaxMinIndex (const CoinPackedVectorBase & x) const` [inline], [protected]

Definition at line 243 of file CoinPackedVectorBase.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinPackedVectorBase.hpp>

9.59 CoinPair< S, T > Struct Template Reference

An ordered pair.

```
#include <CoinSort.hpp>
```

Public Member Functions

- [CoinPair](#) (const S &s, const T &t)
Construct from ordered pair.

Public Attributes

- S [first](#)
First member of pair.
- T [second](#)
Second member of pair.

9.59.1 Detailed Description

```
template<class S, class T>struct CoinPair< S, T >
```

An ordered pair.

It's the same as std::pair, just this way it'll have the same look as the triple sorting.

Definition at line 30 of file CoinSort.hpp.

9.59.2 Constructor & Destructor Documentation

9.59.2.1 `template<class S, class T> CoinPair< S, T >::CoinPair (const S & s, const T & t)` [inline]

Construct from ordered pair.

Definition at line 38 of file CoinSort.hpp.

9.59.3 Member Data Documentation

9.59.3.1 `template<class S, class T> S CoinPair< S, T >::first`

First member of pair.

Definition at line 33 of file CoinSort.hpp.

9.59.3.2 `template<class S, class T> T CoinPair< S, T >::second`

Second member of pair.

Definition at line 35 of file CoinSort.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp>

9.60 CoinParam Class Reference

A base class for 'keyword value' command line parameters.

```
#include <CoinParam.hpp>
```

Public Types

Subtypes

- enum [CoinParamType](#) {
[coinParamInvalid](#) = 0, [coinParamAct](#), [coinParamInt](#), [coinParamDbl](#),
[coinParamStr](#), [coinParamKwd](#) }
Enumeration for the types of parameters supported by [CoinParam](#).
- typedef int(* [CoinParamFunc](#))(CoinParam *param)
Type declaration for push and pull functions.

Public Member Functions

Constructors and Destructors

Be careful how you specify parameters for the constructors! Some compilers are entirely too willing to convert almost anything to bool.

- [CoinParam](#) ()
Default constructor.
- [CoinParam](#) (std::string [name](#), std::string help, double lower, double upper, double dflt=0.0, bool [display](#)=true)
Constructor for a parameter with a double value.
- [CoinParam](#) (std::string [name](#), std::string help, int lower, int upper, int dflt=0, bool [display](#)=true)
Constructor for a parameter with an integer value.
- [CoinParam](#) (std::string [name](#), std::string help, std::string firstValue, int dflt, bool [display](#)=true)
Constructor for a parameter with keyword values.
- [CoinParam](#) (std::string [name](#), std::string help, std::string dflt, bool [display](#)=true)
Constructor for a string parameter.
- [CoinParam](#) (std::string [name](#), std::string help, bool [display](#)=true)

- *Constructor for an action parameter.*
- `CoinParam` (const `CoinParam` &orig)
- *Copy constructor.*
- virtual `CoinParam * clone ()`
- *Clone.*
- `CoinParam & operator=` (const `CoinParam` &rhs)
- *Assignment.*
- virtual `~CoinParam ()`
- *Destructor.*

Methods to query and manipulate the value(s) of a parameter

- `void appendKwd` (std::string kwd)
- *Add an additional value-keyword to a keyword parameter.*
- `int kwdIndex` (std::string kwd) const
- *Return the integer associated with the specified value-keyword.*
- `std::string kwdVal` () const
- *Return the value-keyword that is the current value of the keyword parameter.*
- `void setKwdVal` (int value, bool printIt=false)
- *Set the value of the keyword parameter using the integer associated with a value-keyword.*
- `void setKwdVal` (const std::string value)
- *Set the value of the keyword parameter using a value-keyword string.*
- `void printKwds` () const
- *Prints the set of value-keywords defined for this keyword parameter.*
- `void setStrVal` (std::string value)
- *Set the value of a string parameter.*
- `std::string strVal` () const
- *Get the value of a string parameter.*
- `void setDblVal` (double value)
- *Set the value of a double parameter.*
- `double dblVal` () const
- *Get the value of a double parameter.*
- `void setIntVal` (int value)
- *Set the value of an integer parameter.*
- `int intVal` () const
- *Get the value of an integer parameter.*
- `void setShortHelp` (const std::string help)
- *Add a short help string to a parameter.*
- `std::string shortHelp` () const
- *Retrieve the short help string.*
- `void setLongHelp` (const std::string help)
- *Add a long help message to a parameter.*
- `std::string longHelp` () const
- *Retrieve the long help message.*
- `void printLongHelp` () const
- *Print long help.*

Methods to query and manipulate a parameter object

- `CoinParamType type` () const
- *Return the type of the parameter.*
- `void setType` (CoinParamType type)
- *Set the type of the parameter.*
- `std::string name` () const

- *Return the parameter keyword (name) string.*
- `void setName` (std::string name)
- *Set the parameter keyword (name) string.*
- `int matches` (std::string input) const
- *Check if the specified string matches the parameter keyword (name) string.*
- `std::string matchName` () const
- *Return the parameter keyword (name) string formatted to show the minimum match length.*
- `void setDisplay` (bool display)
- *Set visibility of parameter.*
- `bool display` () const
- *Get visibility of parameter.*
- `CoinParamFunc pushFunc` ()
- *Get push function.*
- `void setPushFunc` (CoinParamFunc func)
- *Set push function.*
- `CoinParamFunc pullFunc` ()
- *Get pull function.*
- `void setPullFunc` (CoinParamFunc func)
- *Set pull function.*

Related Functions

(Note that these are not member functions.)

- `typedef std::vector< CoinParam * > CoinParamVec`
- *A type for a parameter vector.*
- `std::ostream & operator<<` (std::ostream &s, const CoinParam ¶m)
- *A stream output function for a CoinParam object.*
- `void setInputSrc` (FILE *src)
- *Take command input from the file specified by src.*
- `bool isCommandLine` ()
- *Returns true if command line parameters are being processed.*
- `bool isInteractive` ()
- *Returns true if parameters are being obtained from stdin.*
- `std::string getStringField` (int argc, const char *argv[], int *valid)
- *Attempt to read a string from the input.*
- `int getIntField` (int argc, const char *argv[], int *valid)
- *Attempt to read an integer from the input.*
- `double getDoubleField` (int argc, const char *argv[], int *valid)
- *Attempt to read a real (double) from the input.*
- `int matchParam` (const CoinParamVec ¶mVec, std::string name, int &matchNdx, int &shortCnt)
- *Scan a parameter vector for parameters whose keyword (name) string matches name using minimal match rules.*
- `std::string getCommand` (int argc, const char *argv[], const std::string prompt, std::string *pfx=0)
- *Get the next command keyword (name)*
- `int lookupParam` (std::string name, CoinParamVec ¶mVec, int *matchCnt=0, int *shortCnt=0, int *queryCnt=0)
- *Look up the command keyword (name) in the parameter vector. Print help if requested.*
- `void printIt` (const char *msg)
- *Utility to print a long message as filled lines of text.*
- `void shortOrHelpOne` (CoinParamVec ¶mVec, int matchNdx, std::string name, int numQuery)

Utility routine to print help given a short match or explicit request for help.

- `void shortOrHelpMany` (`CoinParamVec` ¶mVec, `std::string` name, `int` numQuery)

Utility routine to print help given multiple matches.

- `void printGenericHelp` ()

Print a generic 'how to use the command interface' help message.

- `void printHelp` (`CoinParamVec` ¶mVec, `int` firstParam, `int` lastParam, `std::string` prefix, `bool` shortHelp, `bool` longHelp, `bool` hidden)

Utility routine to print help messages for one or more parameters.

9.60.1 Detailed Description

A base class for 'keyword value' command line parameters.

The underlying paradigm is that a parameter specifies an action to be performed on a target object. The base class provides two function pointers, a 'push' function and a 'pull' function. By convention, a push function will set some value in the target object or perform some action using the target object. A 'pull' function will retrieve some value from the target object. This is only a convention, however; `CoinParam` and associated utilities make no use of these functions and have no hardcoded notion of how they should be used.

The action to be performed, and the target object, will be specific to a particular application. It is expected that users will derive application-specific parameter classes from this base class. A derived class will typically add fields and methods to set/get a code for the action to be performed (often, an enum class) and the target object (often, a pointer or reference).

Facilities provided by the base class and associated utility routines include:

- Support for common parameter types with numeric, string, or keyword values.
- Support for short and long help messages.
- Pointers to 'push' and 'pull' functions as described above.
- Command line parsing and keyword matching.

All utility routines are declared in the `CoinParamUtils` namespace.

The base class recognises five types of parameters: actions (which require no value); numeric parameters with integer or real (double) values; keyword parameters, where the value is one of a defined set of value-keywords; and string parameters (where the value is a string). The base class supports the definition of a valid range, a default value, and short and long help messages for a parameter.

As defined by the `CoinParamFunc` typedef, push and pull functions should take a single parameter, a pointer to a `CoinParam`. Typically this object will actually be a derived class as described above, and the implementation function will have access to all capabilities of `CoinParam` and of the derived class.

When specified as command line parameters, the expected syntax is '-keyword value' or '-keyword=value'. You can also use the Gnu double-dash style, '--keyword'. Spaces around the '=' will *not* work.

The keyword (name) for a parameter can be defined with an '!' to mark the minimal match point. For example, allow!able-Gap will be considered matched by the strings 'allow', 'allowa', 'allowab', etc. Similarly, the value-keyword strings for keyword parameters can be defined with '!' to mark the minimal match point. Matching of keywords and value-keywords is *not* case sensitive.

Definition at line 75 of file `CoinParam.hpp`.

9.60.2 Member Typedef Documentation

9.60.2.1 `typedef int(* CoinParam::CoinParamFunc)(CoinParam *param)`

Type declaration for push and pull functions.

By convention, a return code of 0 indicates execution without error, >0 indicates nonfatal error, and <0 indicates fatal error. This is only convention, however; the base class makes no use of the push and pull functions and has no hardcoded interpretation of the return code.

Definition at line 106 of file CoinParam.hpp.

9.60.3 Member Enumeration Documentation

9.60.3.1 `enum CoinParam::CoinParamType`

Enumeration for the types of parameters supported by [CoinParam](#).

[CoinParam](#) provides support for several types of parameters:

- Action parameters, which require no value.
- Integer and double numeric parameters, with upper and lower bounds.
- String parameters that take an arbitrary string value.
- Keyword parameters that take a defined set of string (value-keyword) values. Value-keywords are associated with integers in the order in which they are added, starting from zero.

Enumerator

coinParamInvalid

coinParamAct

coinParamInt

coinParamDbf

coinParamStr

coinParamKwd

Definition at line 95 of file CoinParam.hpp.

9.60.4 Constructor & Destructor Documentation

9.60.4.1 `CoinParam::CoinParam ()`

Default constructor.

9.60.4.2 `CoinParam::CoinParam (std::string name, std::string help, double lower, double upper, double dflt = 0.0, bool display = true)`

Constructor for a parameter with a double value.

The default value is 0.0. Be careful to clearly indicate that `lower` and `upper` are real (double) values to distinguish this constructor from the constructor for an integer parameter.

9.60.4.3 `CoinParam::CoinParam (std::string name, std::string help, int lower, int upper, int dflt = 0, bool display = true)`

Constructor for a parameter with an integer value.

The default value is 0.

9.60.4.4 `CoinParam::CoinParam (std::string name, std::string help, std::string firstValue, int dflt, bool display = true)`

Constructor for a parameter with keyword values.

The string supplied as `firstValue` becomes the first value-keyword. Additional value-keywords can be added using `appendKwd()`. It's necessary to specify both the first value-keyword (`firstValue`) and the default value-keyword index (`dflt`) in order to distinguish this constructor from the constructors for string and action parameters.

Value-keywords are associated with an integer, starting with zero and increasing as each keyword is added. The value-keyword given as `firstValue` will be associated with the integer zero. The integer supplied for `dflt` can be any value, as long as it will be valid once all value-keywords have been added.

9.60.4.5 `CoinParam::CoinParam (std::string name, std::string help, std::string dflt, bool display = true)`

Constructor for a string parameter.

For some compilers, the default value (`dflt`) must be specified explicitly with type `std::string` to distinguish the constructor for a string parameter from the constructor for an action parameter. For example, use `std::string("default")` instead of simply `"default"`, or use a variable of type `std::string`.

9.60.4.6 `CoinParam::CoinParam (std::string name, std::string help, bool display = true)`

Constructor for an action parameter.

9.60.4.7 `CoinParam::CoinParam (const CoinParam & orig)`

Copy constructor.

9.60.4.8 `virtual CoinParam::~CoinParam () [virtual]`

Destructor.

9.60.5 Member Function Documentation

9.60.5.1 `virtual CoinParam* CoinParam::clone () [virtual]`

Clone.

9.60.5.2 `CoinParam& CoinParam::operator= (const CoinParam & rhs)`

Assignment.

9.60.5.3 `void CoinParam::appendKwd (std::string kwd)`

Add an additional value-keyword to a keyword parameter.

9.60.5.4 `int CoinParam::kwdIndex (std::string kwd) const`

Return the integer associated with the specified value-keyword.

Returns -1 if no value-keywords match the specified string.

9.60.5.5 `std::string CoinParam::kwdVal () const`

Return the value-keyword that is the current value of the keyword parameter.

9.60.5.6 `void CoinParam::setKwdVal (int value, bool printIt = false)`

Set the value of the keyword parameter using the integer associated with a value-keyword.

If `printIt` is true, the corresponding value-keyword string will be echoed to `std::cout`.

9.60.5.7 `void CoinParam::setKwdVal (const std::string value)`

Set the value of the keyword parameter using a value-keyword string.

The given string will be tested against the set of value-keywords for the parameter using the shortest match rules.

9.60.5.8 `void CoinParam::printKwds () const`

Prints the set of value-keywords defined for this keyword parameter.

9.60.5.9 `void CoinParam::setStrVal (std::string value)`

Set the value of a string parameter.

9.60.5.10 `std::string CoinParam::strVal () const`

Get the value of a string parameter.

9.60.5.11 `void CoinParam::setDbfVal (double value)`

Set the value of a double parameter.

9.60.5.12 `double CoinParam::dblVal () const`

Get the value of a double parameter.

9.60.5.13 `void CoinParam::setIntVal (int value)`

Set the value of a integer parameter.

9.60.5.14 `int CoinParam::intVal () const`

Get the value of a integer parameter.

9.60.5.15 `void CoinParam::setShortHelp (const std::string help) [inline]`

Add a short help string to a parameter.

Definition at line 259 of file `CoinParam.hpp`.

9.60.5.16 `std::string CoinParam::shortHelp () const [inline]`

Retrieve the short help string.

Definition at line 263 of file `CoinParam.hpp`.

9.60.5.17 `void CoinParam::setLongHelp (const std::string help) [inline]`

Add a long help message to a parameter.

See [printLongHelp\(\)](#) for a description of how messages are broken into lines.

Definition at line 270 of file CoinParam.hpp.

9.60.5.18 `std::string CoinParam::longHelp () const` `[inline]`

Retrieve the long help message.

Definition at line 274 of file CoinParam.hpp.

9.60.5.19 `void CoinParam::printLongHelp () const`

Print long help.

Prints the long help string, plus the valid range and/or keywords if appropriate. The routine makes a best effort to break the message into lines appropriate for an 80-character line. Explicit line breaks in the message will be observed. The short help string will be used if long help is not available.

9.60.5.20 `CoinParamType CoinParam::type () const` `[inline]`

Return the type of the parameter.

Definition at line 293 of file CoinParam.hpp.

9.60.5.21 `void CoinParam::setType (CoinParamType type)` `[inline]`

Set the type of the parameter.

Definition at line 297 of file CoinParam.hpp.

9.60.5.22 `std::string CoinParam::name () const` `[inline]`

Return the parameter keyword (name) string.

Definition at line 301 of file CoinParam.hpp.

9.60.5.23 `void CoinParam::setName (std::string name)` `[inline]`

Set the parameter keyword (name) string.

Definition at line 305 of file CoinParam.hpp.

9.60.5.24 `int CoinParam::matches (std::string input) const`

Check if the specified string matches the parameter keyword (name) string.

Returns 1 if the string matches and meets the minimum match length, 2 if the string matches but doesn't meet the minimum match length, and 0 if the string doesn't match. Matches are *not* case-sensitive.

9.60.5.25 `std::string CoinParam::matchName () const`

Return the parameter keyword (name) string formatted to show the minimum match length.

For example, if the parameter name was defined as allow!ableGap, the string returned by matchName would be allow(ableGap).

9.60.5.26 `void CoinParam::setDisplay (bool display)` `[inline]`

Set visibility of parameter.

Intended to control whether the parameter is shown when a list of parameters is processed. Used by [CoinParamUtils::printHelp](#) when printing help messages for a list of parameters.

Definition at line 330 of file CoinParam.hpp.

9.60.5.27 `bool CoinParam::display () const [inline]`

Get visibility of parameter.

Definition at line 334 of file CoinParam.hpp.

9.60.5.28 `CoinParamFunc CoinParam::pushFunc () [inline]`

Get push function.

Definition at line 338 of file CoinParam.hpp.

9.60.5.29 `void CoinParam::setPushFunc (CoinParamFunc func) [inline]`

Set push function.

Definition at line 342 of file CoinParam.hpp.

9.60.5.30 `CoinParamFunc CoinParam::pullFunc () [inline]`

Get pull function.

Definition at line 346 of file CoinParam.hpp.

9.60.5.31 `void CoinParam::setPullFunc (CoinParamFunc func) [inline]`

Set pull function.

Definition at line 350 of file CoinParam.hpp.

9.60.6 Friends And Related Function Documentation

9.60.6.1 `typedef std::vector<CoinParam*> CoinParamVec [related]`

A type for a parameter vector.

Definition at line 429 of file CoinParam.hpp.

9.60.6.2 `std::ostream & operator<< (std::ostream & s, const CoinParam & param) [related]`

A stream output function for a [CoinParam](#) object.

9.60.6.3 `void setInputSrc (FILE * src) [related]`

Take command input from the file specified by src.

Use stdin for src to specify interactive prompting for commands.

9.60.6.4 `bool isCommandLine () [related]`

Returns true if command line parameters are being processed.

9.60.6.5 `bool isInteractive () [related]`

Returns true if parameters are being obtained from stdin.

9.60.6.6 `std::string getStringField (int argc, const char * argv[], int * valid)` [related]

Attempt to read a string from the input.

`argc` and `argv` are used only if `isCommandLine()` would return true. If `valid` is supplied, it will be set to 0 if a string is parsed without error, 2 if no field is present.

9.60.6.7 `int getIntField (int argc, const char * argv[], int * valid)` [related]

Attempt to read an integer from the input.

`argc` and `argv` are used only if `isCommandLine()` would return true. If `valid` is supplied, it will be set to 0 if an integer is parsed without error, 1 if there's a parse error, and 2 if no field is present.

9.60.6.8 `double getDoubleField (int argc, const char * argv[], int * valid)` [related]

Attempt to read a real (double) from the input.

`argc` and `argv` are used only if `isCommandLine()` would return true. If `valid` is supplied, it will be set to 0 if a real number is parsed without error, 1 if there's a parse error, and 2 if no field is present.

9.60.6.9 `int matchParam (const CoinParamVec & paramVec, std::string name, int & matchNdx, int & shortCnt)` [related]

Scan a parameter vector for parameters whose keyword (name) string matches `name` using minimal match rules.

`matchNdx` is set to the index of the last parameter that meets the minimal match criteria (but note there should be at most one matching parameter if the parameter vector is properly configured). `shortCnt` is set to the number of short matches (should be zero for a properly configured parameter vector if a minimal match is found). The return value is the number of matches satisfying the minimal match requirement (should be 0 or 1 in a properly configured vector).

9.60.6.10 `std::string getCommand (int argc, const char * argv[], const std::string prompt, std::string * pfx = 0)` [related]

Get the next command keyword (name)

To be precise, return the next field from the current command input source, after a bit of processing. In command line mode (`isCommandLine()` returns true) the next field will normally be of the form 'keyword' or '-keyword' (i.e., a parameter keyword), and the string returned would be 'keyword'. In interactive mode (`isInteractive()` returns true), the user will be prompted if necessary. It is assumed that the user knows not to use the '-' or '-' prefixes unless specifying parameters on the command line.

There are a number of special cases if we're in command line mode. The order of processing of the raw string goes like this:

- A stand-alone '-' is forced to 'stdin'.
- A stand-alone '-' is returned as a word; interpretation is up to the client.
- A prefix of '-' or '-' is stripped from the string.

If the result is the string 'stdin', command processing shifts to interactive mode and the user is immediately prompted for a new command.

Whatever results from the above sequence is returned to the user as the return value of the function. An empty string indicates end of input.

`prompt` will be used only if it's necessary to prompt the user in interactive mode.

9.60.6.11 `int lookupParam (std::string name, CoinParamVec & paramVec, int * matchCnt = 0, int * shortCnt = 0, int * queryCnt = 0)` [related]

Look up the command keyword (name) in the parameter vector. Print help if requested.

In the most straightforward use, `name` is a string without '?', and the value returned is the index in `paramVec` of the single parameter that matched `name`. One or more '?' characters at the end of `name` is a query for information. The routine prints short (one '?') or long (more than one '?') help messages for a query. Help is also printed in the case where the name is ambiguous (some of the matches did not meet the minimal match length requirement).

Note that multiple matches meeting the minimal match requirement is a configuration error. The minimal match length for the parameters involved is too short.

If provided as parameters, on return

- `matchCnt` will be set to the number of matches meeting the minimal match requirement
- `shortCnt` will be set to the number of matches that did not meet the minimal match requirement
- `queryCnt` will be set to the number of '?' characters at the end of the name

The return values are:

- >0: index in `paramVec` of the single unique match for `name`
- -1: a query was detected (one or more '?' characters at the end of `name`)
- -2: one or more short matches, not a query
- -3: no matches, not a query
- -4: multiple matches meeting the minimal match requirement (configuration error)

9.60.6.12 `void printIt (const char * msg)` [related]

Utility to print a long message as filled lines of text.

The routine makes a best effort to break lines without exceeding the standard 80 character line length. Explicit newlines in `msg` will be obeyed.

9.60.6.13 `void shortOrHelpOne (CoinParamVec & paramVec, int matchNdx, std::string name, int numQuery)` [related]

Utility routine to print help given a short match or explicit request for help.

The two really are related, in that a query (a string that ends with one or more '?' characters) will often result in a short match. The routine expects that `name` matches a single parameter, and does not look for multiple matches.

If called with `matchNdx < 0`, the routine will look up `name` in `paramVec` and print the full name from the parameter. If called with `matchNdx > 0`, it just prints the name from the specified parameter. If the name is a query, short (one '?') or long (more than one '?') help is printed.

9.60.6.14 `void shortOrHelpMany (CoinParamVec & paramVec, std::string name, int numQuery)` [related]

Utility routine to print help given multiple matches.

If the name is not a query, or asks for short help (*i.e.*, contains zero or one '?' characters), the list of matching names is printed. If the name asks for long help (contains two or more '?' characters), short help is printed for each matching name.

9.60.6.15 void printGenericHelp () [related]

Print a generic 'how to use the command interface' help message.

The message is hard coded to match the behaviour of the parsing utilities.

9.60.6.16 void printHelp (CoinParamVec & paramVec, int firstParam, int lastParam, std::string prefix, bool shortHelp, bool longHelp, bool hidden) [related]

Utility routine to print help messages for one or more parameters.

Intended as a utility to implement explicit 'help' commands. Help will be printed for all parameters in paramVec from firstParam to lastParam, inclusive. If shortHelp is true, short help messages will be printed. If longHelp is true, long help messages are printed. shortHelp overrules longHelp. If neither is true, only command keywords are printed. prefix is printed before each line; it's an imperfect attempt at indentation.

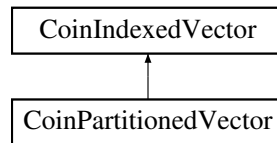
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/CoinUtils/src/CoinParam.hpp

9.61 CoinPartitionedVector Class Reference

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for CoinPartitionedVector:



Public Member Functions

Get methods.

- int [getNumElements](#) (int partition) const
Get the size of a partition.
- int [getNumPartitions](#) () const
Get number of partitions.
- int [getNumElements](#) () const
Get the size.
- int [startPartition](#) (int partition) const
Get starts.
- const int * [startPartitions](#) () const
Get starts.

Set methods

- void [setNumElementsPartition](#) (int partition, int value)
Set the size of a partition.
- void [setTempNumElementsPartition](#) (int partition, int value)
Set the size of a partition (just for a tiny while)
- void [computeNumberElements](#) ()

- *Add up number of elements in partitions.*
- `void compact ()`
- *Add up number of elements in partitions and pack and get rid of partitions.*
- `void reserve (int n)`
- *Reserve space.*
- `void setPartitions (int number, const int *starts)`
- *Setup partitions (needs end as well)*
- `void clearAndReset ()`
- *Reset the vector (as if were just created an empty vector). Gets rid of partitions.*
- `void clearAndKeep ()`
- *Reset the vector (as if were just created an empty vector). Keeps partitions.*
- `void clearPartition (int partition)`
- *Clear a partition.*
- `void checkClear ()`
- *For debug check vector is clear i.e. no elements.*
- `void checkClean ()`
- *For debug check vector is clean i.e. elements match indices.*
- `int scan (int partition, double tolerance=0.0)`
- *Scan dense region and set up indices (returns number found)*
- `void print () const`
- *Scan dense region from start to < end and set up indices returns number found.*

Sorting

- `void sort ()`
- *Sort the indexed storage vector (increasing indices).*

Constructors and destructors (not all written)

- `CoinPartitionedVector ()`
- *Default constructor.*
- `CoinPartitionedVector (int size, const int *inds, const double *elems)`
- *Alternate Constructors - set elements to vector of doubles.*
- `CoinPartitionedVector (int size, const int *inds, double element)`
- *Alternate Constructors - set elements to same scalar value.*
- `CoinPartitionedVector (int size, const double *elements)`
- *Alternate Constructors - construct full storage with indices 0 through size-1.*
- `CoinPartitionedVector (int size)`
- *Alternate Constructors - just size.*
- `CoinPartitionedVector (const CoinPartitionedVector &)`
- *Copy constructor.*
- `CoinPartitionedVector (const CoinPartitionedVector *)`
- *Copy constructor.2.*
- `CoinPartitionedVector & operator= (const CoinPartitionedVector &)`
- *Assignment operator.*
- `~CoinPartitionedVector ()`
- *Destructor.*

Protected Attributes

Private member data

- `int startPartition_ [COIN_PARTITIONS+1]`
- *Starts.*
- `int numberElementsPartition_ [COIN_PARTITIONS]`
- *Size of indices in a partition.*
- `int numberPartitions_`
- *Number of partitions (0 means off)*

9.61.1 Detailed Description

Definition at line 1055 of file CoinIndexedVector.hpp.

9.61.2 Constructor & Destructor Documentation

9.61.2.1 CoinPartitionedVector::CoinPartitionedVector ()

Default constructor.

9.61.2.2 CoinPartitionedVector::CoinPartitionedVector (int *size*, const int * *inds*, const double * *elems*)

Alternate Constructors - set elements to vector of doubles.

9.61.2.3 CoinPartitionedVector::CoinPartitionedVector (int *size*, const int * *inds*, double *element*)

Alternate Constructors - set elements to same scalar value.

9.61.2.4 CoinPartitionedVector::CoinPartitionedVector (int *size*, const double * *elements*)

Alternate Constructors - construct full storage with indices 0 through size-1.

9.61.2.5 CoinPartitionedVector::CoinPartitionedVector (int *size*)

Alternate Constructors - just size.

9.61.2.6 CoinPartitionedVector::CoinPartitionedVector (const CoinPartitionedVector &)

Copy constructor.

9.61.2.7 CoinPartitionedVector::CoinPartitionedVector (const CoinPartitionedVector *)

Copy constructor.2.

9.61.2.8 CoinPartitionedVector::~~CoinPartitionedVector ()

Destructor.

9.61.3 Member Function Documentation

9.61.3.1 int CoinPartitionedVector::getNumElements (int *partition*) const [inline]

Get the size of a partition.

Definition at line 1064 of file CoinIndexedVector.hpp.

9.61.3.2 int CoinPartitionedVector::getNumPartitions () const [inline]

Get number of partitions.

Definition at line 1067 of file CoinIndexedVector.hpp.

9.61.3.3 int CoinPartitionedVector::getNumElements () const [inline]

Get the size.

Definition at line 1070 of file CoinIndexedVector.hpp.

9.61.3.4 `int CoinPartitionedVector::startPartition (int partition) const` `[inline]`

Get starts.

Definition at line 1072 of file CoinIndexedVector.hpp.

9.61.3.5 `const int* CoinPartitionedVector::startPartitions () const` `[inline]`

Get starts.

Definition at line 1075 of file CoinIndexedVector.hpp.

9.61.3.6 `void CoinPartitionedVector::setNumElementsPartition (int partition, int value)` `[inline]`

Set the size of a partition.

Definition at line 1085 of file CoinIndexedVector.hpp.

9.61.3.7 `void CoinPartitionedVector::setTempNumElementsPartition (int partition, int value)` `[inline]`

Set the size of a partition (just for a tiny while)

Definition at line 1088 of file CoinIndexedVector.hpp.

9.61.3.8 `void CoinPartitionedVector::computeNumberElements ()`

Add up number of elements in partitions.

9.61.3.9 `void CoinPartitionedVector::compact ()`

Add up number of elements in partitions and pack and get rid of partitions.

9.61.3.10 `void CoinPartitionedVector::reserve (int n)`

Reserve space.

9.61.3.11 `void CoinPartitionedVector::setPartitions (int number, const int * starts)`

Setup partitions (needs end as well)

9.61.3.12 `void CoinPartitionedVector::clearAndReset ()`

Reset the vector (as if were just created an empty vector). Gets rid of partitions.

9.61.3.13 `void CoinPartitionedVector::clearAndKeep ()`

Reset the vector (as if were just created an empty vector). Keeps partitions.

9.61.3.14 `void CoinPartitionedVector::clearPartition (int partition)`

Clear a partition.

9.61.3.15 `void CoinPartitionedVector::checkClear ()`

For debug check vector is clear i.e. no elements.

9.61.3.16 `void CoinPartitionedVector::checkClean ()`

For debug check vector is clean i.e. elements match indices.

9.61.3.17 `int CoinPartitionedVector::scan (int partition, double tolerance = 0.0)`

Scan dense region and set up indices (returns number found)

9.61.3.18 `void CoinPartitionedVector::print () const`

Scan dense region from start to < end and set up indices returns number found.

Print out

9.61.3.19 `void CoinPartitionedVector::sort ()`

Sort the indexed storage vector (increasing indices).

9.61.3.20 `CoinPartitionedVector& CoinPartitionedVector::operator= (const CoinPartitionedVector &)`

Assignment operator.

9.61.4 Member Data Documentation

9.61.4.1 `int CoinPartitionedVector::startPartition_[COIN_PARTITIONS+1] [protected]`

Starts.

Definition at line 1155 of file `CoinIndexedVector.hpp`.

9.61.4.2 `int CoinPartitionedVector::numberElementsPartition_[COIN_PARTITIONS] [protected]`

Size of indices in a partition.

Definition at line 1157 of file `CoinIndexedVector.hpp`.

9.61.4.3 `int CoinPartitionedVector::numberPartitions_ [protected]`

Number of partitions (0 means off)

Definition at line 1159 of file `CoinIndexedVector.hpp`.

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

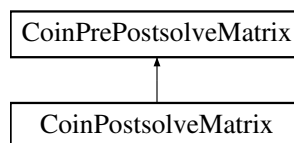
- [/home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp](#)

9.62 CoinPostsolveMatrix Class Reference

Augments [CoinPrePostsolveMatrix](#) with information about the problem that is only needed during postsolve.

```
#include <CoinPresolveMatrix.hpp>
```

Inheritance diagram for `CoinPostsolveMatrix`:



Public Member Functions

- [CoinPostsolveMatrix](#) (int ncols_alloc, int nrows_alloc, [CoinBigIndex](#) nelems_alloc)
'Native' constructor
- [CoinPostsolveMatrix](#) (ClpSimplex *si, int ncols0, int nrows0, [CoinBigIndex](#) nelems0, double maxmin_, double *sol, double *acts, unsigned char *colstat, unsigned char *rowstat)
Clp OSI constructor.
- [CoinPostsolveMatrix](#) (OsiSolverInterface *si, int ncols0, int nrows0, [CoinBigIndex](#) nelems0, double maxmin_, double *sol, double *acts, unsigned char *colstat, unsigned char *rowstat)
Generic OSI constructor.
- void [assignPresolveToPostsolve](#) ([CoinPresolveMatrix](#) *&preObj)
Load an empty [CoinPostsolveMatrix](#) from a [CoinPresolveMatrix](#).
- [~CoinPostsolveMatrix](#) ()
Destructor.
- void [check_nbasic](#) ()
debug

Public Attributes

Column thread structures

As mentioned in the class documentation, the entries for a given column do not necessarily occupy a contiguous block of space.

The [link_](#) array is used to maintain the threading. There is one thread for each column, and a single thread for all free entries in [hrow_](#) and [colels_](#).

The allocated size of [link_](#) must be at least as large as the allocated size of [hrow_](#) and [colels_](#).

- [CoinBigIndex](#) [free_list_](#)
First entry in free entries thread.
- int [maxlink_](#)
Allocated size of [link_](#).
- [CoinBigIndex](#) * [link_](#)
Thread array.

Debugging aids

These arrays are allocated only when [CoinPresolve](#) is compiled with `PRESOLVE_DEBUG` defined.

They hold codes which track the reason that a column or row is added to the problem during postsolve.

- char * [cdone_](#)
- char * [rdone_](#)

Related Functions

(Note that these are not member functions.)

- [CoinBigIndex](#) [presolve_find_col](#) (int col, [CoinBigIndex](#) krs, [CoinBigIndex](#) kre, const int *hcol)
Find position of a column in a row in a row-major matrix.
- [CoinBigIndex](#) [presolve_find_minor2](#) (int tgt, [CoinBigIndex](#) ks, int majlen, const int *minndx, const [CoinBigIndex](#) *majlinks)
Find position of a minor index in a major vector in a threaded matrix.

- [CoinBigIndex presolve_find_row2](#) (int row, [CoinBigIndex](#) kcs, int collen, const int *hrow, const [CoinBigIndex](#) *clinks)
Find position of a row in a column in a column-major threaded matrix.
- [CoinBigIndex presolve_find_minor3](#) (int tgt, [CoinBigIndex](#) ks, int majlen, const int *minndx, const [CoinBigIndex](#) *majlinks)
Find position of a minor index in a major vector in a threaded matrix.
- [CoinBigIndex presolve_find_row3](#) (int row, [CoinBigIndex](#) kcs, int collen, const int *hrow, const [CoinBigIndex](#) *clinks)
Find position of a row in a column in a column-major threaded matrix.
- [void presolve_delete_from_major2](#) (int majndx, int minndx, [CoinBigIndex](#) *majstrts, int *majlens, int *minndx, int *majlinks, [CoinBigIndex](#) *free_listp)
Delete the entry for a minor index from a major vector in a threaded matrix.
- [void presolve_delete_from_col2](#) (int row, int col, [CoinBigIndex](#) *mcstrt, int *hincol, int *hrow, int *clinks, [CoinBigIndex](#) *free_listp)
Delete the entry for row row from column col in a column-major threaded matrix.
- [void presolve_check_threads](#) (const [CoinPostsolveMatrix](#) *obj)
Checks that column threads agree with column lengths.
- [void presolve_check_free_list](#) (const [CoinPostsolveMatrix](#) *obj, bool chkElemCnt=false)
Checks the free list.
- [void presolve_check_reduced_costs](#) (const [CoinPostsolveMatrix](#) *obj)
Check stored reduced costs for accuracy and consistency with variable status.
- [void presolve_check_duals](#) (const [CoinPostsolveMatrix](#) *postObj)
Check the dual variables for consistency with row activity.
- [void presolve_check_sol](#) (const [CoinPostsolveMatrix](#) *postObj, int chkColSol=2, int chkRowAct=2, int chkStatus=1)
Check primal solution and architectural variable status.
- [void presolve_check_nbasic](#) (const [CoinPostsolveMatrix](#) *postObj)
Check for the proper number of basic variables.

Additional Inherited Members

9.62.1 Detailed Description

Augments [CoinPrePostsolveMatrix](#) with information about the problem that is only needed during postsolve.

The notable point is that the matrix representation is threaded. The representation is column-major and starts with the standard two pairs of arrays: one pair to hold the row indices and coefficients, the second pair to hold the column starting positions and lengths. But the row indices and coefficients for a column do not necessarily occupy a contiguous block in their respective arrays. Instead, a link array gives the position of the next (row index,coefficient) pair. If the row index and value of a coefficient $a_{\langle p,j \rangle}$ occupy position k_p in their arrays, then the position of the next coefficient $a_{\langle q,j \rangle}$ is found as $k_q = \text{link}[k_p]$.

This threaded representation allows for efficient expansion of columns as rows are reintroduced during postsolve transformations. The basic packed structures are allocated to the expected size of the postsolved matrix, and as new coefficients are added, their location is simply added to the thread for the column.

There is no provision to convert the threaded representation to a packed representation. In the context of postsolve, it's not required. (You did keep a copy of the original matrix, eh?)

The constructors that take an OSI or ClpSimplex as a parameter really should not be here, but for historical reasons they will likely remain for the foreseeable future. – lh, 111202 –

Definition at line 1421 of file CoinPresolveMatrix.hpp.

9.62.2 Constructor & Destructor Documentation

9.62.2.1 CoinPostsolveMatrix::CoinPostsolveMatrix (int *ncols_alloc*, int *nrows_alloc*, CoinBigIndex *nelems_alloc*)

'Native' constructor

This constructor creates an empty object which must then be loaded. On the other hand, it doesn't assume that the client is an OsiSolverInterface.

9.62.2.2 CoinPostsolveMatrix::CoinPostsolveMatrix (ClpSimplex * *si*, int *ncols0*, int *nrows0*, CoinBigIndex *nelems0*, double *maxmin_*, double * *sol*, double * *acts*, unsigned char * *colstat*, unsigned char * *rowstat*)

Clp OSI constructor.

See Clp code for the definition.

9.62.2.3 CoinPostsolveMatrix::CoinPostsolveMatrix (OsiSolverInterface * *si*, int *ncols0*, int *nrows0*, CoinBigIndex *nelems0*, double *maxmin_*, double * *sol*, double * *acts*, unsigned char * *colstat*, unsigned char * *rowstat*)

Generic OSI constructor.

See OSI code for the definition.

9.62.2.4 CoinPostsolveMatrix::~CoinPostsolveMatrix ()

Destructor.

9.62.3 Member Function Documentation

9.62.3.1 void CoinPostsolveMatrix::assignPresolveToPostsolve (CoinPresolveMatrix * & *preObj*)

Load an empty [CoinPostsolveMatrix](#) from a [CoinPresolveMatrix](#).

This routine transfers the contents of the [CoinPrePostsolveMatrix](#) object from the [CoinPresolveMatrix](#) object to the [CoinPostsolveMatrix](#) object and completes initialisation of the [CoinPostsolveMatrix](#) object. The empty shell of the [CoinPresolveMatrix](#) object is destroyed.

The routine expects an empty [CoinPostsolveMatrix](#) object. If handed a loaded object, a lot of memory will leak.

9.62.3.2 void CoinPostsolveMatrix::check_nbasic ()

debug

9.62.4 Member Data Documentation

9.62.4.1 CoinBigIndex CoinPostsolveMatrix::free_list_

First entry in free entries thread.

Definition at line 1501 of file CoinPresolveMatrix.hpp.

9.62.4.2 int CoinPostsolveMatrix::maxlink_

Allocated size of [link_](#).

Definition at line 1503 of file CoinPresolveMatrix.hpp.

9.62.4.3 `CoinBigIndex*` `CoinPostsolveMatrix::link_`

Thread array.

Within a thread, `link_[k]` points to the next entry in the thread.

Definition at line 1508 of file `CoinPresolveMatrix.hpp`.

9.62.4.4 `char*` `CoinPostsolveMatrix::cdone_`

Definition at line 1519 of file `CoinPresolveMatrix.hpp`.

9.62.4.5 `char*` `CoinPostsolveMatrix::rdone_`

Definition at line 1520 of file `CoinPresolveMatrix.hpp`.

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

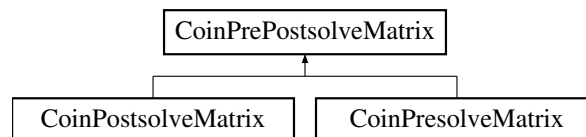
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMatrix.hpp](#)
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolvePsdebug.hpp](#)

9.63 `CoinPrePostsolveMatrix` Class Reference

Collects all the information about the problem that is needed in both presolve and postsolve.

```
#include <CoinPresolveMatrix.hpp>
```

Inheritance diagram for `CoinPrePostsolveMatrix`:



Public Types

- enum `Status` {
`isFree` = 0x00, `basic` = 0x01, `atUpperBound` = 0x02, `atLowerBound` = 0x03,
`superBasic` = 0x04 }

Enum for status of various sorts.

Public Member Functions

Constructors & Destructors

- `CoinPrePostsolveMatrix` (int `ncols_alloc`, int `nrows_alloc`, `CoinBigIndex` `nelems_alloc`)
'Native' constructor
- `CoinPrePostsolveMatrix` (const `OsiSolverInterface` *`si`, int `ncols_`, int `nrows_`, `CoinBigIndex` `nelems_`)
Generic OSI constructor.
- `CoinPrePostsolveMatrix` (const `ClpSimplex` *`si`, int `ncols_`, int `nrows_`, `CoinBigIndex` `nelems_`, double `bulk-Ratio`)
ClpOsi constructor.
- `~CoinPrePostsolveMatrix` ()

Destructor.

Functions to work with variable status

Functions to work with the [CoinPrePostsolveMatrix::Status](#) enum and related vectors.

Todo Why are we futzing around with three bit status? A holdover from the packed arrays of [CoinWarmStartBasis](#)? Big swaths of the presolve code manipulates `colstat_` and `rowstat_` as unsigned char arrays using simple assignment to set values.

- [void setRowStatus](#) (int sequence, [Status](#) status)
Set row status (i.e., status of artificial for this row)
- [Status getRowStatus](#) (int sequence) const
Get row status.
- bool [rowsBasic](#) (int sequence) const
Check if artificial for this row is basic.
- [void setColumnStatus](#) (int sequence, [Status](#) status)
Set column status (i.e., status of primal variable)
- [Status getColumnStatus](#) (int sequence) const
Get column (structural variable) status.
- bool [columnsBasic](#) (int sequence) const
Check if column (structural variable) is basic.
- [void setRowStatusUsingValue](#) (int iRow)
Set status of row (artificial variable) to the correct nonbasic status given bounds and current value.
- [void setColumnStatusUsingValue](#) (int iColumn)
Set status of column (structural variable) to the correct nonbasic status given bounds and current value.
- [void setStructuralStatus](#) (const char *strucStatus, int lenParam)
Set column (structural variable) status vector.
- [void setArtificialStatus](#) (const char *artifStatus, int lenParam)
Set row (artificial variable) status vector.
- [void setStatus](#) (const [CoinWarmStartBasis](#) *basis)
Set the status of all variables from a basis.
- [CoinWarmStartBasis](#) * [getStatus](#) ()
Get status in the form of a [CoinWarmStartBasis](#).
- const char * [columnStatusString](#) (int j) const
Return a print string for status of a column (structural variable)
- const char * [rowStatusString](#) (int i) const
Return a print string for status of a row (artificial variable)

Functions to load problem and solution information

These functions can be used to load portions of the problem definition and solution.

See also the [CoinPresolveMatrix](#) and [CoinPostsolveMatrix](#) classes.

- [void setObjOffset](#) (double offset)
Set the objective function offset for the original system.
- [void setObjSense](#) (double objSense)
Set the objective sense (max/min)
- [void setPrimalTolerance](#) (double primTol)
Set the primal feasibility tolerance.
- [void setDualTolerance](#) (double dualTol)
Set the dual feasibility tolerance.
- [void setColLower](#) (const double *colLower, int lenParam)
Set column lower bounds.
- [void setColUpper](#) (const double *colUpper, int lenParam)
Set column upper bounds.

- `void setColSolution` (const double *colSol, int lenParam)
Set column solution.
- `void setCost` (const double *cost, int lenParam)
Set objective coefficients.
- `void setReducedCost` (const double *redCost, int lenParam)
Set reduced costs.
- `void setRowLower` (const double *rowLower, int lenParam)
Set row lower bounds.
- `void setRowUpper` (const double *rowUpper, int lenParam)
Set row upper bounds.
- `void setRowPrice` (const double *rowSol, int lenParam)
Set row solution.
- `void setRowActivity` (const double *rowAct, int lenParam)
Set row activity.

Functions to retrieve problem and solution information

- int `getNumCols` () const
Get current number of columns.
- int `getNumRows` () const
Get current number of rows.
- int `getNumElems` () const
Get current number of non-zero coefficients.
- const `CoinBigIndex` * `getColStarts` () const
Get column start vector for column-major packed matrix.
- const int * `getColLengths` () const
Get column length vector for column-major packed matrix.
- const int * `getRowIndicesByCol` () const
Get vector of row indices for column-major packed matrix.
- const double * `getElementsByCol` () const
Get vector of elements for column-major packed matrix.
- const double * `getColLower` () const
Get column lower bounds.
- const double * `getColUpper` () const
Get column upper bounds.
- const double * `getCost` () const
Get objective coefficients.
- const double * `getRowLower` () const
Get row lower bounds.
- const double * `getRowUpper` () const
Get row upper bounds.
- const double * `getColSolution` () const
Get column solution (primal variable values)
- const double * `getRowActivity` () const
Get row activity (constraint lhs values)
- const double * `getRowPrice` () const
Get row solution (dual variables)
- const double * `getReducedCost` () const
Get reduced costs.
- int `countEmptyCols` ()
Count empty columns.

Public Attributes

Current and Allocated Size

During pre- and postsolve, the matrix will change in size.

During presolve it will shrink; during postsolve it will grow. Hence there are two sets of size variables, one for the current size and one for the allocated size. (See the general comments for the [CoinPrePostsolveMatrix](#) class for more information.)

- int [ncols_](#)
current number of columns
- int [nrows_](#)
current number of rows
- [CoinBigIndex](#) [nelems_](#)
current number of coefficients
- int [ncols0_](#)
Allocated number of columns.
- int [nrows0_](#)
Allocated number of rows.
- [CoinBigIndex](#) [nelems0_](#)
Allocated number of coefficients.
- [CoinBigIndex](#) [bulk0_](#)
Allocated size of bulk storage for row indices and coefficients.
- double [bulkRatio_](#)
Ratio of bulk0_ to nelems0_; default is 2.

Problem representation

The matrix is the common column-major format: A pair of vectors with positional correspondence to hold coefficients and row indices, and a second pair of vectors giving the starting position and length of each column in the first pair.

- [CoinBigIndex](#) * [mcstrt_](#)
Vector of column start positions in [hrow_](#), [colels_](#).
- int * [hincol_](#)
Vector of column lengths.
- int * [hrow_](#)
Row indices (positional correspondence with [colels_](#))
- double * [colels_](#)
Coefficients (positional correspondence with [hrow_](#))
- double * [cost_](#)
Objective coefficients.
- double [originalOffset_](#)
Original objective offset.
- double * [clo_](#)
Column (primal variable) lower bounds.
- double * [cup_](#)
Column (primal variable) upper bounds.
- double * [rlo_](#)
Row (constraint) lower bounds.
- double * [rup_](#)
Row (constraint) upper bounds.
- int * [originalColumn_](#)
Original column numbers.
- int * [originalRow_](#)
Original row numbers.
- double [ztolzb_](#)

- *Primal feasibility tolerance.*
- double [ztoldj_](#)
- *Dual feasibility tolerance.*
- double [maxmin_](#)
- *Maximization/minimization.*

Problem solution information

The presolve phase will work without any solution information (appropriate for initial optimisation) or with solution information (appropriate for reoptimisation).

When solution information is supplied, presolve will maintain it to the best of its ability. [colstat_](#) is checked to determine the presence/absence of status information. [sol_](#) is checked for primal solution information, and [rowduals_](#) for dual solution information.

The postsolve phase requires the complete solution information from the presolved problem (status, primal and dual solutions). It will be transformed into a correct solution for the original problem.

- double * [sol_](#)
Vector of primal variable values.
- double * [rowduals_](#)
Vector of dual variable values.
- double * [acts_](#)
Vector of constraint left-hand-side values (row activity)
- double * [rcosts_](#)
Vector of reduced costs.
- unsigned char * [colstat_](#)
Status of primal variables.
- unsigned char * [rowstat_](#)
Status of constraints.

Related Functions

(Note that these are not member functions.)

- const char * [statusName](#) ([CoinPrePostsolveMatrix::Status](#) status)
Generate a print string for a status code.
- void [presolve_make_memlists](#) (int *lengths, [presolvehlink](#) *link, int n)
Initialise linked list for major vector order in bulk storage.
- bool [presolve_expand_major](#) ([CoinBigIndex](#) *majstrts, double *majels, int *minndx, int *majlens, [presolvehlink](#) *majlinks, int nmaj, int k)
Make sure a major-dimension vector k has room for one more coefficient.
- bool [presolve_expand_col](#) ([CoinBigIndex](#) *mcstrt, double *colels, int *hrow, int *hincol, [presolvehlink](#) *clink, int ncols, int colx)
Make sure a column (colx) in a column-major matrix has room for one more coefficient.
- bool [presolve_expand_row](#) ([CoinBigIndex](#) *mrstrt, double *rowels, int *hcol, int *hinrow, [presolvehlink](#) *rlink, int nrow, int rowx)
Make sure a row (rowx) in a row-major matrix has room for one more coefficient.
- [CoinBigIndex](#) [presolve_find_minor](#) (int tgt, [CoinBigIndex](#) ks, [CoinBigIndex](#) ke, const int *minndx)
Find position of a minor index in a major vector.
- [CoinBigIndex](#) [presolve_find_row](#) (int row, [CoinBigIndex](#) kcs, [CoinBigIndex](#) kce, const int *hrow)
Find position of a row in a column in a column-major matrix.
- [CoinBigIndex](#) [presolve_find_minor1](#) (int tgt, [CoinBigIndex](#) ks, [CoinBigIndex](#) ke, const int *minndx)

Find position of a minor index in a major vector.

- [CoinBigIndex presolve_find_row1](#) (int row, [CoinBigIndex](#) kcs, [CoinBigIndex](#) kce, const int *hrow)

Find position of a row in a column in a column-major matrix.

- [CoinBigIndex presolve_find_col1](#) (int col, [CoinBigIndex](#) krs, [CoinBigIndex](#) kre, const int *hcol)

Find position of a column in a row in a row-major matrix.

- [void presolve_delete_from_major](#) (int majndx, int minndx, const [CoinBigIndex](#) *majstrts, int *majlens, int *minndxs, double *els)

Delete the entry for a minor index from a major vector.

- [void presolve_delete_many_from_major](#) (int majndx, char *marked, const [CoinBigIndex](#) *majstrts, int *majlens, int *minndxs, double *els)

Delete marked entries.

- [void presolve_delete_from_col](#) (int row, int col, const [CoinBigIndex](#) *mcstrt, int *hincol, int *hrow, double *colels)

Delete the entry for row `row` from column `col` in a column-major matrix.

- [void presolve_delete_from_row](#) (int row, int col, const [CoinBigIndex](#) *mrstrt, int *hinrow, int *hcol, double *rowels)

Delete the entry for column `col` from row `row` in a row-major matrix.

Message handling

Uses the standard COIN approach: a default handler is installed, and the [CoinPrePostsolveMatrix](#) object takes responsibility for it.

If the client replaces the handler with one of their own, it becomes their responsibility.

- [CoinMessageHandler * handler_](#)

Message handler.

- [bool defaultHandler_](#)

Indicates if the current [handler_](#) is default (true) or not (false).

- [CoinMessage messages_](#)

Standard COIN messages.

- [CoinMessageHandler * messageHandler](#) () const

Return message handler.

- [void setMessageHandler](#) ([CoinMessageHandler](#) *handler)

Set message handler.

- [CoinMessages messages](#) () const

Return messages.

9.63.1 Detailed Description

Collects all the information about the problem that is needed in both presolve and postsolve.

In a bit more detail, a column-major representation of the constraint matrix and upper and lower bounds on variables and constraints, plus row and column solutions, reduced costs, and status. There's also a set of arrays holding the original row and column numbers.

As presolve and postsolve transform the matrix, it will occasionally be necessary to expand the number of entries in a column. There are two aspects:

- During postsolve, the constraint system is expected to grow as the smaller presolved system is transformed back to the original system.

- During both pre- and postsolve, transforms can increase the number of coefficients in a row or column. (See the variable substitution, doubleton, and tripleton transforms.)

The first is addressed by the members `ncols0_`, `nrows0_`, and `nelems0_`. These should be set (via constructor parameters) to values large enough for the largest size taken on by the constraint system. Typically, this will be the size of the original constraint system.

The second is addressed by a generous allocation of extra (empty) space for the arrays used to hold coefficients and row indices. When columns must be expanded, they are moved into the empty space. When it is used up, the arrays are compacted. When compaction fails to produce sufficient space, presolve/postsolve will fail.

`CoinPrePostsolveMatrix` isn't really intended to be used 'bare' — the expectation is that it'll be used through `CoinPresolveMatrix` or `CoinPostsolveMatrix`. Some of the functions needed to load a problem are defined in the derived classes.

When `CoinPresolve` is applied when reoptimising, we need to be prepared to accept a basis and modify it in step with the presolve actions (otherwise we throw away all the advantages of warm start for reoptimization). But other solution components (`acts_`, `rowduals_`, `sol_`, and `rcosts_`) are needed only for postsolve, where they're used in places to determine the proper action(s) when restoring rows or columns. If presolve is provided with a solution, it will modify it in step with the presolve actions. Moving the solution components from `CoinPrePostsolveMatrix` to `CoinPostsolveMatrix` would break a lot of code. It's not clear that it's worth it, and it would preclude upgrades to the presolve side that might make use of any of these. — lh, 080501 —

The constructors that take an `OSI` or `ClpSimplex` as a parameter really should not be here, but for historical reasons they will likely remain for the foreseeable future. — lh, 111202 —

Definition at line 265 of file `CoinPresolveMatrix.hpp`.

9.63.2 Member Enumeration Documentation

9.63.2.1 `enum CoinPrePostsolveMatrix::Status`

Enum for status of various sorts.

Matches `CoinWarmStartBasis::Status` and adds `superBasic`. Most code that converts between `CoinPrePostsolveMatrix::Status` and `CoinWarmStartBasis::Status` will break if this correspondence is broken.

`superBasic` is an unresolved problem: there's no analogue in `CoinWarmStartBasis::Status`.

Enumerator

```
isFree
basic
atUpperBound
atLowerBound
superBasic
```

Definition at line 313 of file `CoinPresolveMatrix.hpp`.

9.63.3 Constructor & Destructor Documentation

9.63.3.1 `CoinPrePostsolveMatrix::CoinPrePostsolveMatrix (int ncols_alloc, int nrows_alloc, CoinBigIndex nelems_alloc)`

'Native' constructor

This constructor creates an empty object which must then be loaded. On the other hand, it doesn't assume that the client is an `OsiSolverInterface`.

9.63.3.2 `CoinPrePostsolveMatrix::CoinPrePostsolveMatrix (const OsiSolverInterface * si, int ncols_, int nrows_, CoinBigIndex nelems_)`

Generic OSI constructor.

See OSI code for the definition.

9.63.3.3 `CoinPrePostsolveMatrix::CoinPrePostsolveMatrix (const ClpSimplex * si, int ncols_, int nrows_, CoinBigIndex nelems_, double bulkRatio)`

ClpOsi constructor.

See Clp code for the definition.

9.63.3.4 `CoinPrePostsolveMatrix::~~CoinPrePostsolveMatrix ()`

Destructor.

9.63.4 Member Function Documentation

9.63.4.1 `void CoinPrePostsolveMatrix::setRowStatus (int sequence, Status status)` `[inline]`

Set row status (*i.e.*, status of artificial for this row)

Definition at line 335 of file CoinPresolveMatrix.hpp.

9.63.4.2 `Status CoinPrePostsolveMatrix::getRowStatus (int sequence) const` `[inline]`

Get row status.

Definition at line 342 of file CoinPresolveMatrix.hpp.

9.63.4.3 `bool CoinPrePostsolveMatrix::rowsIsBasic (int sequence) const` `[inline]`

Check if artificial for this row is basic.

Definition at line 345 of file CoinPresolveMatrix.hpp.

9.63.4.4 `void CoinPrePostsolveMatrix::setColumnStatus (int sequence, Status status)` `[inline]`

Set column status (*i.e.*, status of primal variable)

Definition at line 348 of file CoinPresolveMatrix.hpp.

9.63.4.5 `Status CoinPrePostsolveMatrix::getColumnStatus (int sequence) const` `[inline]`

Get column (structural variable) status.

Definition at line 388 of file CoinPresolveMatrix.hpp.

9.63.4.6 `bool CoinPrePostsolveMatrix::columnsIsBasic (int sequence) const` `[inline]`

Check if column (structural variable) is basic.

Definition at line 391 of file CoinPresolveMatrix.hpp.

9.63.4.7 `void CoinPrePostsolveMatrix::setRowStatusUsingValue (int iRow)`

Set status of row (artificial variable) to the correct nonbasic status given bounds and current value.

9.63.4.8 **void** CoinPrePostsolveMatrix::setColumnStatusUsingValue (int *iColumn*)

Set status of column (structural variable) to the correct nonbasic status given bounds and current value.

9.63.4.9 **void** CoinPrePostsolveMatrix::setStructuralStatus (const char * *strucStatus*, int *lenParam*)

Set column (structural variable) status vector.

9.63.4.10 **void** CoinPrePostsolveMatrix::setArtificialStatus (const char * *artifStatus*, int *lenParam*)

Set row (artificial variable) status vector.

9.63.4.11 **void** CoinPrePostsolveMatrix::setStatus (const CoinWarmStartBasis * *basis*)

Set the status of all variables from a basis.

9.63.4.12 **CoinWarmStartBasis*** CoinPrePostsolveMatrix::getStatus ()

Get status in the form of a [CoinWarmStartBasis](#).

9.63.4.13 **const char*** CoinPrePostsolveMatrix::columnStatusString (int *j*) **const**

Return a print string for status of a column (structural variable)

9.63.4.14 **const char*** CoinPrePostsolveMatrix::rowStatusString (int *i*) **const**

Return a print string for status of a row (artificial variable)

9.63.4.15 **void** CoinPrePostsolveMatrix::setObjOffset (double *offset*)

Set the objective function offset for the original system.

9.63.4.16 **void** CoinPrePostsolveMatrix::setObjSense (double *objSense*)

Set the objective sense (max/min)

Coded as 1.0 for min, -1.0 for max. Yes, there's a method, and a matching attribute. No, you really don't want to set this to maximise.

9.63.4.17 **void** CoinPrePostsolveMatrix::setPrimalTolerance (double *primTol*)

Set the primal feasibility tolerance.

9.63.4.18 **void** CoinPrePostsolveMatrix::setDualTolerance (double *dualTol*)

Set the dual feasibility tolerance.

9.63.4.19 **void** CoinPrePostsolveMatrix::setColLower (const double * *colLower*, int *lenParam*)

Set column lower bounds.

9.63.4.20 **void** CoinPrePostsolveMatrix::setColUpper (const double * *colUpper*, int *lenParam*)

Set column upper bounds.

9.63.4.21 **void** CoinPrePostsolveMatrix::setColSolution (const double * *colSol*, int *lenParam*)

Set column solution.

9.63.4.22 **void** CoinPrePostsolveMatrix::setCost (*const double * cost*, *int lenParam*)

Set objective coefficients.

9.63.4.23 **void** CoinPrePostsolveMatrix::setReducedCost (*const double * redCost*, *int lenParam*)

Set reduced costs.

9.63.4.24 **void** CoinPrePostsolveMatrix::setRowLower (*const double * rowLower*, *int lenParam*)

Set row lower bounds.

9.63.4.25 **void** CoinPrePostsolveMatrix::setRowUpper (*const double * rowUpper*, *int lenParam*)

Set row upper bounds.

9.63.4.26 **void** CoinPrePostsolveMatrix::setRowPrice (*const double * rowSol*, *int lenParam*)

Set row solution.

9.63.4.27 **void** CoinPrePostsolveMatrix::setRowActivity (*const double * rowAct*, *int lenParam*)

Set row activity.

9.63.4.28 **int** CoinPrePostsolveMatrix::getNumCols () **const** [inline]

Get current number of columns.

Definition at line 462 of file CoinPresolveMatrix.hpp.

9.63.4.29 **int** CoinPrePostsolveMatrix::getNumRows () **const** [inline]

Get current number of rows.

Definition at line 465 of file CoinPresolveMatrix.hpp.

9.63.4.30 **int** CoinPrePostsolveMatrix::getNumElems () **const** [inline]

Get current number of non-zero coefficients.

Definition at line 468 of file CoinPresolveMatrix.hpp.

9.63.4.31 **const CoinBigIndex*** CoinPrePostsolveMatrix::getColStarts () **const** [inline]

Get column start vector for column-major packed matrix.

Definition at line 471 of file CoinPresolveMatrix.hpp.

9.63.4.32 **const int*** CoinPrePostsolveMatrix::getColLengths () **const** [inline]

Get column length vector for column-major packed matrix.

Definition at line 474 of file CoinPresolveMatrix.hpp.

9.63.4.33 **const int*** CoinPrePostsolveMatrix::getRowIndicesByCol () **const** [inline]

Get vector of row indices for column-major packed matrix.

Definition at line 477 of file CoinPresolveMatrix.hpp.

9.63.4.34 `const double* CoinPrePostsolveMatrix::getElementsByCol () const` `[inline]`

Get vector of elements for column-major packed matrix.

Definition at line 480 of file CoinPresolveMatrix.hpp.

9.63.4.35 `const double* CoinPrePostsolveMatrix::getColLower () const` `[inline]`

Get column lower bounds.

Definition at line 483 of file CoinPresolveMatrix.hpp.

9.63.4.36 `const double* CoinPrePostsolveMatrix::getColUpper () const` `[inline]`

Get column upper bounds.

Definition at line 486 of file CoinPresolveMatrix.hpp.

9.63.4.37 `const double* CoinPrePostsolveMatrix::getCost () const` `[inline]`

Get objective coefficients.

Definition at line 489 of file CoinPresolveMatrix.hpp.

9.63.4.38 `const double* CoinPrePostsolveMatrix::getRowLower () const` `[inline]`

Get row lower bounds.

Definition at line 492 of file CoinPresolveMatrix.hpp.

9.63.4.39 `const double* CoinPrePostsolveMatrix::getRowUpper () const` `[inline]`

Get row upper bounds.

Definition at line 495 of file CoinPresolveMatrix.hpp.

9.63.4.40 `const double* CoinPrePostsolveMatrix::getColSolution () const` `[inline]`

Get column solution (primal variable values)

Definition at line 498 of file CoinPresolveMatrix.hpp.

9.63.4.41 `const double* CoinPrePostsolveMatrix::getRowActivity () const` `[inline]`

Get row activity (constraint lhs values)

Definition at line 501 of file CoinPresolveMatrix.hpp.

9.63.4.42 `const double* CoinPrePostsolveMatrix::getRowPrice () const` `[inline]`

Get row solution (dual variables)

Definition at line 504 of file CoinPresolveMatrix.hpp.

9.63.4.43 `const double* CoinPrePostsolveMatrix::getReducedCost () const` `[inline]`

Get reduced costs.

Definition at line 507 of file CoinPresolveMatrix.hpp.

9.63.4.44 `int CoinPrePostsolveMatrix::countEmptyCols ()` [inline]

Count empty columns.

Definition at line 510 of file CoinPresolveMatrix.hpp.

9.63.4.45 `CoinMessageHandler* CoinPrePostsolveMatrix::messageHandler ()` const [inline]

Return message handler.

Definition at line 520 of file CoinPresolveMatrix.hpp.

9.63.4.46 `void CoinPrePostsolveMatrix::setMessageHandler (CoinMessageHandler * handler)` [inline]

Set message handler.

The client retains responsibility for the handler — it will not be destroyed with the [CoinPrePostsolveMatrix](#) object.

Definition at line 527 of file CoinPresolveMatrix.hpp.

9.63.4.47 `CoinMessages CoinPrePostsolveMatrix::messages ()` const [inline]

Return messages.

Definition at line 533 of file CoinPresolveMatrix.hpp.

9.63.5 Friends And Related Function Documentation

9.63.5.1 `const char * statusName (CoinPrePostsolveMatrix::Status status)` [related]

Generate a print string for a status code.

9.63.6 Member Data Documentation

9.63.6.1 `int CoinPrePostsolveMatrix::ncols_`

current number of columns

Definition at line 548 of file CoinPresolveMatrix.hpp.

9.63.6.2 `int CoinPrePostsolveMatrix::nrows_`

current number of rows

Definition at line 550 of file CoinPresolveMatrix.hpp.

9.63.6.3 `CoinBigIndex CoinPrePostsolveMatrix::nelems_`

current number of coefficients

Definition at line 552 of file CoinPresolveMatrix.hpp.

9.63.6.4 `int CoinPrePostsolveMatrix::ncols0_`

Allocated number of columns.

Definition at line 555 of file CoinPresolveMatrix.hpp.

9.63.6.5 int CoinPrePostsolveMatrix::nrows0_

Allocated number of rows.

Definition at line 557 of file CoinPresolveMatrix.hpp.

9.63.6.6 CoinBigIndex CoinPrePostsolveMatrix::nelems0_

Allocated number of coefficients.

Definition at line 559 of file CoinPresolveMatrix.hpp.

9.63.6.7 CoinBigIndex CoinPrePostsolveMatrix::bulk0_

Allocated size of bulk storage for row indices and coefficients.

This is the space allocated for `hrow_` and `colels_`. This must be large enough to allow columns to be copied into empty space when they need to be expanded. For efficiency (to minimize the number of times the representation must be compressed) it's recommended that this be at least $2 * \text{nelems0_}$.

Definition at line 568 of file CoinPresolveMatrix.hpp.

9.63.6.8 double CoinPrePostsolveMatrix::bulkRatio_

Ratio of `bulk0_` to `nelems0_`; default is 2.

Definition at line 570 of file CoinPresolveMatrix.hpp.

9.63.6.9 CoinBigIndex* CoinPrePostsolveMatrix::mcstrt_

Vector of column start positions in [hrow_](#), [colels_](#).

Definition at line 582 of file CoinPresolveMatrix.hpp.

9.63.6.10 int* CoinPrePostsolveMatrix::hincol_

Vector of column lengths.

Definition at line 584 of file CoinPresolveMatrix.hpp.

9.63.6.11 int* CoinPrePostsolveMatrix::hrow_

Row indices (positional correspondence with [colels_](#))

Definition at line 586 of file CoinPresolveMatrix.hpp.

9.63.6.12 double* CoinPrePostsolveMatrix::colels_

Coefficients (positional correspondence with [hrow_](#))

Definition at line 588 of file CoinPresolveMatrix.hpp.

9.63.6.13 double* CoinPrePostsolveMatrix::cost_

Objective coefficients.

Definition at line 591 of file CoinPresolveMatrix.hpp.

9.63.6.14 double CoinPrePostsolveMatrix::originalOffset_

Original objective offset.

Definition at line 593 of file CoinPresolveMatrix.hpp.

9.63.6.15 `double* CoinPrePostsolveMatrix::clo_`

Column (primal variable) lower bounds.

Definition at line 596 of file CoinPresolveMatrix.hpp.

9.63.6.16 `double* CoinPrePostsolveMatrix::cup_`

Column (primal variable) upper bounds.

Definition at line 598 of file CoinPresolveMatrix.hpp.

9.63.6.17 `double* CoinPrePostsolveMatrix::rlo_`

Row (constraint) lower bounds.

Definition at line 601 of file CoinPresolveMatrix.hpp.

9.63.6.18 `double* CoinPrePostsolveMatrix::rup_`

Row (constraint) upper bounds.

Definition at line 603 of file CoinPresolveMatrix.hpp.

9.63.6.19 `int* CoinPrePostsolveMatrix::originalColumn_`

Original column numbers.

Over the current range of column numbers in the presolved problem, the entry for column *j* will contain the index of the corresponding column in the original problem.

Definition at line 611 of file CoinPresolveMatrix.hpp.

9.63.6.20 `int* CoinPrePostsolveMatrix::originalRow_`

Original row numbers.

Over the current range of row numbers in the presolved problem, the entry for row *i* will contain the index of the corresponding row in the original problem.

Definition at line 618 of file CoinPresolveMatrix.hpp.

9.63.6.21 `double CoinPrePostsolveMatrix::ztolzb_`

Primal feasibility tolerance.

Definition at line 621 of file CoinPresolveMatrix.hpp.

9.63.6.22 `double CoinPrePostsolveMatrix::ztoldj_`

Dual feasibility tolerance.

Definition at line 623 of file CoinPresolveMatrix.hpp.

9.63.6.23 `double CoinPrePostsolveMatrix::maxmin_`

Maximization/minimization.

Yes, there's a variable here. No, you really don't want to set this to maximise. See the main notes for [CoinPresolveMatrix](#).

Definition at line 630 of file CoinPresolveMatrix.hpp.

9.63.6.24 double* CoinPrePostsolveMatrix::sol_

Vector of primal variable values.

If [sol_](#) exists, it is assumed that primal solution information should be updated and that [acts_](#) also exists.

Definition at line 653 of file CoinPresolveMatrix.hpp.

9.63.6.25 double* CoinPrePostsolveMatrix::rowduals_

Vector of dual variable values.

If [rowduals_](#) exists, it is assumed that dual solution information should be updated and that [rcosts_](#) also exists.

Definition at line 659 of file CoinPresolveMatrix.hpp.

9.63.6.26 double* CoinPrePostsolveMatrix::acts_

Vector of constraint left-hand-side values (row activity)

Produced by evaluating constraints according to [sol_](#). Updated iff [sol_](#) exists.

Definition at line 665 of file CoinPresolveMatrix.hpp.

9.63.6.27 double* CoinPrePostsolveMatrix::rcosts_

Vector of reduced costs.

Produced by evaluating dual constraints according to [rowduals_](#). Updated iff [rowduals_](#) exists.

Definition at line 671 of file CoinPresolveMatrix.hpp.

9.63.6.28 unsigned char* CoinPrePostsolveMatrix::colstat_

Status of primal variables.

Coded with `CoinPrePostSolveMatrix::Status`, one code per char. `colstat_` and `rowstat_` **MUST** be allocated as a single vector. This is to maintain compatibility with `ClpPresolve` and `OsiPresolve`, which do it this way.

Definition at line 679 of file CoinPresolveMatrix.hpp.

9.63.6.29 unsigned char* CoinPrePostsolveMatrix::rowstat_

Status of constraints.

More accurately, the status of the logical variable associated with the constraint. Coded with `CoinPrePostSolveMatrix::Status`, one code per char. Note that this must be allocated as a single vector with `colstat_`.

Definition at line 687 of file CoinPresolveMatrix.hpp.

9.63.6.30 CoinMessageHandler* CoinPrePostsolveMatrix::handler_

Message handler.

Definition at line 699 of file CoinPresolveMatrix.hpp.

9.63.6.31 bool CoinPrePostsolveMatrix::defaultHandler_

Indicates if the current [handler_](#) is default (true) or not (false).

Definition at line 701 of file CoinPresolveMatrix.hpp.

9.63.6.32 CoinMessage CoinPrePostsolveMatrix::messages_

Standard COIN messages.

Definition at line 703 of file CoinPresolveMatrix.hpp.

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

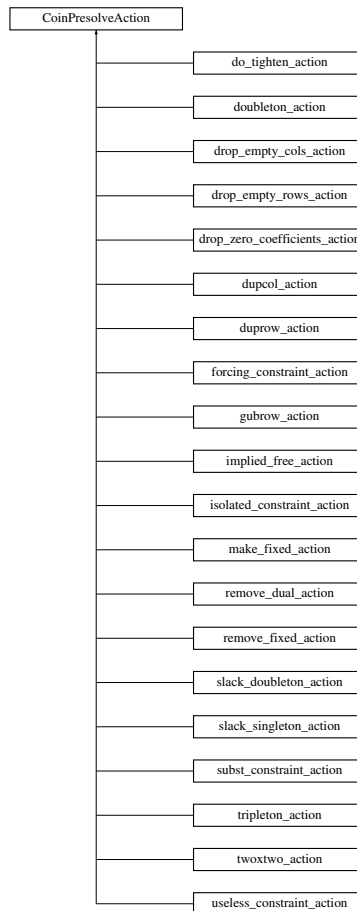
- /home/ted/COIN/trunk/CoinUtils/src/[CoinPresolveMatrix.hpp](#)

9.64 CoinPresolveAction Class Reference

Abstract base class of all presolve routines.

```
#include <CoinPresolveMatrix.hpp>
```

Inheritance diagram for CoinPresolveAction:



Public Member Functions

- [CoinPresolveAction](#) (const [CoinPresolveAction](#) *next)
Construct a postsolve object and add it to the transformation list.
- [void setNext](#) (const [CoinPresolveAction](#) *nextAction)
modify next (when building rather than passing)

- virtual const char * [name](#) () const =0
A name for debug printing.
- virtual void [postsolve](#) ([CoinPostsolveMatrix](#) *prob) const =0
Apply the postsolve transformation for this particular presolve action.
- virtual ~[CoinPresolveAction](#) ()
Virtual destructor.

Static Public Member Functions

- static void [throwCoinError](#) (const char *error, const char *ps_routine)
Stub routine to throw exceptions.

Public Attributes

- const [CoinPresolveAction](#) * [next](#)
The next presolve transformation.

9.64.1 Detailed Description

Abstract base class of all presolve routines.

The details will make more sense after a quick overview of the grand plan: A presolve object is handed a problem object, which it is expected to modify in some useful way. Assuming that it succeeds, the presolve object should create a postsolve object, *i.e.*, an object that contains instructions for backing out the presolve transform to recover the original problem. These postsolve objects are accumulated in a linked list, with each successive presolve action adding its postsolve action to the head of the list. The end result of all this is a presolved problem object, and a list of postsolve objects. The presolved problem object is then handed to a solver for optimization, and the problem object augmented with the results. The list of postsolve objects is then traversed. Each of them (un)modifies the problem object, with the end result being the original problem, augmented with solution information.

The problem object representation is [CoinPrePostsolveMatrix](#) and subclasses. Check there for details. The [CoinPresolveAction](#) class and subclasses represent the presolve and postsolve objects.

In spite of the name, the only information held in a [CoinPresolveAction](#) object is the information needed to postsolve (*i.e.*, the information needed to back out the presolve transformation). This information is not expected to change, so the fields are all `const`.

A subclass of [CoinPresolveAction](#), implementing a specific pre/postsolve action, is expected to declare a static function that attempts to perform a presolve transformation. This function will be handed a [CoinPresolveMatrix](#) to transform, and a pointer to the head of the list of postsolve objects. If the transform is successful, the function will create a new [CoinPresolveAction](#) object, link it at the head of the list of postsolve objects, and return a pointer to the postsolve object it has just created. Otherwise, it should return 0. It is expected that these static functions will be the only things that can create new [CoinPresolveAction](#) objects; this is expressed by making each subclass' constructor(s) private.

Every subclass must also define a `postsolve` method. This function will be handed a [CoinPostsolveMatrix](#) to transform.

It is the client's responsibility to implement presolve and postsolve driver routines. See [OsiPresolve](#) for examples.

Note

Since the only fields in a [CoinPresolveAction](#) are `const`, anything one can do with a variable declared `CoinPresolveAction*` can also be done with a variable declared `const CoinPresolveAction*`. It is expected that all derived subclasses of [CoinPresolveAction](#) also have this property.

Definition at line 155 of file `CoinPresolveMatrix.hpp`.

9.64.2 Constructor & Destructor Documentation

9.64.2.1 `CoinPresolveAction::CoinPresolveAction (const CoinPresolveAction * next)` `[inline]`

Construct a postsolve object and add it to the transformation list.

This is an 'add to head' operation. This object will point to the one passed as the parameter.

Definition at line 178 of file `CoinPresolveMatrix.hpp`.

9.64.2.2 `virtual CoinPresolveAction::~CoinPresolveAction ()` `[inline]`, `[virtual]`

Virtual destructor.

Definition at line 195 of file `CoinPresolveMatrix.hpp`.

9.64.3 Member Function Documentation

9.64.3.1 `static void CoinPresolveAction::throwCoinError (const char * error, const char * ps_routine)` `[inline]`, `[static]`

Stub routine to throw exceptions.

Exceptions are inefficient, particularly with g++. Even with x8C, the use of exceptions adds a long prologue to a routine. Therefore, rather than use `throw` directly in the routine, I use it in a stub routine.

Definition at line 164 of file `CoinPresolveMatrix.hpp`.

9.64.3.2 `void CoinPresolveAction::setNext (const CoinPresolveAction * nextAction)` `[inline]`

modify next (when building rather than passing)

Definition at line 180 of file `CoinPresolveMatrix.hpp`.

9.64.3.3 `virtual const char* CoinPresolveAction::name () const` `[pure virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implemented in [twotwo_action](#), [gubrow_action](#), [make_fixed_action](#), [drop_empty_rows_action](#), [duprow_action](#), [slack_singleton_action](#), [subst_constraint_action](#), [doubleton_action](#), [dupcol_action](#), [drop_empty_cols_action](#), [tripleton_action](#), [remove_fixed_action](#), [forcing_constraint_action](#), [slack_doubleton_action](#), [implied_free_action](#), [drop_zero_coefficients_action](#), [remove_dual_action](#), [do_tighten_action](#), [isolated_constraint_action](#), and [useless_constraint_action](#).

9.64.3.4 `virtual void CoinPresolveAction::postsolve (CoinPostsolveMatrix * prob) const` `[pure virtual]`

Apply the postsolve transformation for this particular presolve action.

Implemented in [twotwo_action](#), [make_fixed_action](#), [gubrow_action](#), [drop_empty_rows_action](#), [duprow_action](#), [slack_singleton_action](#), [subst_constraint_action](#), [remove_fixed_action](#), [doubleton_action](#), [drop_empty_cols_action](#), [dupcol-](#)

[_action](#), [slack_doubleton_action](#), [triplet_action](#), [remove_dual_action](#), [forcing_constraint_action](#), [implied_free_action](#), [drop_zero_coefficients_action](#), [do_tighten_action](#), [isolated_constraint_action](#), and [useless_constraint_action](#).

9.64.4 Member Data Documentation

9.64.4.1 `const CoinPresolveAction*` `CoinPresolveAction::next`

The next presolve transformation.

Set at object construction.

Definition at line 171 of file `CoinPresolveMatrix.hpp`.

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

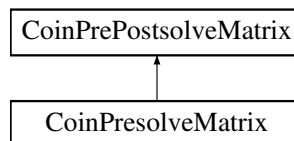
- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMatrix.hpp`

9.65 CoinPresolveMatrix Class Reference

Augments [CoinPrePostsolveMatrix](#) with information about the problem that is only needed during presolve.

```
#include <CoinPresolveMatrix.hpp>
```

Inheritance diagram for `CoinPresolveMatrix`:



Public Member Functions

- [CoinPresolveMatrix](#) (int ncols_alloc, int nrows_alloc, [CoinBigIndex](#) nelems_alloc)
'Native' constructor
- [CoinPresolveMatrix](#) (int ncols0, double maxmin, ClpSimplex *si, int nrows, [CoinBigIndex](#) nelems, bool doStatus, double nonLinearVariable, double bulkRatio)
Clp OSI constructor.
- [void update_model](#) (ClpSimplex *si, int nrows0, int ncols0, [CoinBigIndex](#) nelems0)
Update the model held by a Clp OSI.
- [CoinPresolveMatrix](#) (int ncols0, double maxmin, OsiSolverInterface *si, int nrows, [CoinBigIndex](#) nelems, bool doStatus, double nonLinearVariable, const char *prohibited, const char *rowProhibited=NULL)
Generic OSI constructor.
- [void update_model](#) (OsiSolverInterface *si, int nrows0, int ncols0, [CoinBigIndex](#) nelems0)
Update the model held by a generic OSI.
- [~CoinPresolveMatrix](#) ()
Destructor.
- [void change_bias](#) (double change_amount)
Adjust objective function constant offset.
- [void statistics](#) ()
Say we want statistics - also set time.

- double [feasibilityTolerance](#) ()
Return feasibility tolerance.
- void [setFeasibilityTolerance](#) (double val)
Set feasibility tolerance.
- int [status](#) ()
Returns problem status (0 = feasible, 1 = infeasible, 2 = unbounded)
- void [setStatus](#) (int status)
Set problem status.
- void [setPass](#) (int pass=0)
Set pass number.
- void [setMaximumSubstitutionLevel](#) (int level)
Set Maximum substitution level (normally 3)
- int [recomputeSums](#) (int whichRow)
Recompute row lhs bounds.
- void [initializeStuff](#) ()
Allocate scratch arrays.
- void [deleteStuff](#) ()
Free scratch arrays.

Functions to load the problem representation

- void [setMatrix](#) (const [CoinPackedMatrix](#) *mtx)
Load the coefficient matrix.
- int [countEmptyRows](#) ()
Count number of empty rows.
- void [setVariableType](#) (int i, int variableType)
Set variable type information for a single variable.
- void [setVariableType](#) (const unsigned char *variableType, int lenParam)
Set variable type information for all variables.
- void [setVariableType](#) (bool allIntegers, int lenParam)
Set the type of all variables.
- void [setAnyInteger](#) (bool [anyInteger](#)=true)
Set a flag for presence (true) or absence (false) of integer variables.

Functions to retrieve problem information

- const [CoinBigIndex](#) * [getRowStarts](#) () const
Get row start vector for row-major packed matrix.
- const int * [getColIndicesByRow](#) () const
Get vector of column indices for row-major packed matrix.
- const double * [getElementsByRow](#) () const
Get vector of elements for row-major packed matrix.
- bool [isInteger](#) (int i) const
Check for integrality of the specified variable.
- bool [anyInteger](#) () const
Check if there are any integer variables.
- int [presolveOptions](#) () const
Picks up any special options.
- void [setPresolveOptions](#) (int value)
Sets any special options (see [presolveOptions_](#))

Functions to manipulate row and column processing status

- `void initColsToDo ()`
Initialise the column ToDo lists.
- `int stepColsToDo ()`
Step column ToDo lists.
- `int numberColsToDo ()`
Return the number of columns on the `colsToDo_` list.
- `bool colChanged (int i) const`
Has column been changed?
- `void unsetColChanged (int i)`
Mark column as not changed.
- `void setColChanged (int i)`
Mark column as changed.
- `void addCol (int i)`
Mark column as changed and add to list of columns to process next.
- `bool colProhibited (int i) const`
Test if column is eligible for preprocessing.
- `bool colProhibited2 (int i) const`
Test if column is eligible for preprocessing.
- `void setColProhibited (int i)`
Mark column as ineligible for preprocessing.
- `bool colUsed (int i) const`
Test if column is marked as used.
- `void setColUsed (int i)`
Mark column as used.
- `void unsetColUsed (int i)`
Mark column as unused.
- `bool collInfinite (int i) const`
Has column infinite ub (originally)
- `void unsetCollInfinite (int i)`
Mark column as not infinite ub (originally)
- `void setCollInfinite (int i)`
Mark column as infinite ub (originally)
- `void initRowsToDo ()`
Initialise the row ToDo lists.
- `int stepRowsToDo ()`
Step row ToDo lists.
- `int numberRowsToDo ()`
Return the number of rows on the `rowsToDo_` list.
- `bool rowChanged (int i) const`
Has row been changed?
- `void unsetRowChanged (int i)`
Mark row as not changed.
- `void setRowChanged (int i)`
Mark row as changed.
- `void addRow (int i)`
Mark row as changed and add to list of rows to process next.
- `bool rowProhibited (int i) const`
Test if row is eligible for preprocessing.
- `bool rowProhibited2 (int i) const`
Test if row is eligible for preprocessing.
- `void setRowProhibited (int i)`
Mark row as ineligible for preprocessing.
- `bool rowUsed (int i) const`

- *Test if row is marked as used.*
• `void setRowUsed` (int i)
- *Mark row as used.*
• `void unsetRowUsed` (int i)
- *Mark row as unused.*
• `bool anyProhibited` () const
- *Check if there are any prohibited rows or columns.*
• `void setAnyProhibited` (bool val=true)
- *Set a flag for presence of prohibited rows or columns.*

Public Attributes

- double `dobias_`
Objective function offset introduced during presolve.
- unsigned char * `integerType_`
Tracks integrality of columns (1 for integer, 0 for continuous)
- bool `anyInteger_`
Flag to say if any variables are integer.
- bool `tuning_`
Print statistics for tuning.
- double `startTime_`
Start time of presolve.
- double `feasibilityTolerance_`
Bounds can be moved by this to retain feasibility.
- int `status_`
Output status: 0 = feasible, 1 = infeasible, 2 = unbounded.
- int `pass_`
Presolve pass number.
- int `maxSubstLevel_`
Maximum substitution level.

Matrix storage management links

Linked lists, modelled after the linked lists used in OSL factorization.

They are used for management of the bulk coefficient and minor index storage areas.

- `presolvehlink` * `clink_`
Linked list for the column-major representation.
- `presolvehlink` * `rlink_`
Linked list for the row-major representation.

Row-major representation

Common row-major format: A pair of vectors with positional correspondence to hold coefficients and column indices, and a second pair of vectors giving the starting position and length of each row in the first pair.

- `CoinBigIndex` * `mrstrt_`
Vector of row start positions in #hcol, `rowels_`.
- int * `hinrow_`
Vector of row lengths.
- double * `rowels_`
Coefficients (positional correspondence with `hcol_`)

- int * [hcol_](#)
Column indices (positional correspondence with [rowels_](#))

Row and column processing status

Information used to determine if rows or columns can be changed and if they require further processing due to changes.

There are four major lists: the [\[row,col\]ToDo](#) list, and the [\[row,col\]NextToDo](#) list. In general, a transform processes entries from the [ToDo](#) list and adds entries to the [NextToDo](#) list.

There are two vectors, [\[row,col\]Changed](#), which track the status of individual rows and columns.

- unsigned char * [colChanged_](#)
Column change status information.
- int * [colsToDo_](#)
Input list of columns to process.
- int [numberColsToDo_](#)
Length of [colsToDo_](#).
- int * [nextColsToDo_](#)
Output list of columns to process next.
- int [numberNextColsToDo_](#)
Length of [nextColsToDo_](#).
- unsigned char * [rowChanged_](#)
Row change status information.
- int * [rowsToDo_](#)
Input list of rows to process.
- int [numberRowsToDo_](#)
Length of [rowsToDo_](#).
- int * [nextRowsToDo_](#)
Output list of rows to process next.
- int [numberNextRowsToDo_](#)
Length of [nextRowsToDo_](#).
- int [presolveOptions_](#)
Fine control over presolve actions.
- bool [anyProhibited_](#)
Flag to say if any rows or columns are marked as prohibited.

Scratch work arrays

Preallocated work arrays are useful to avoid having to allocate and free work arrays in individual presolve methods.

All are allocated from [setMatrix](#) by [initializeStuff](#), freed from [~CoinPresolveMatrix](#). You can use [deleteStuff](#) followed by [initializeStuff](#) to remove and recreate them.

- int * [usefulRowInt_](#)
*Preallocated scratch work array, 3*nrows_.*
- double * [usefulRowDouble_](#)
*Preallocated scratch work array, 2*nrows_.*
- int * [usefulColumnInt_](#)
*Preallocated scratch work array, 2*ncols_.*
- double * [usefulColumnDouble_](#)
Preallocated scratch work array, ncols_.
- double * [randomNumber_](#)
Array of random numbers (max row,column)
- int * [infiniteUp_](#)
Work array for count of infinite contributions to row lhs upper bound.
- double * [sumUp_](#)

- *Work array for sum of finite contributions to row lhs upper bound.*
- `int * infiniteDown_`
Work array for count of infinite contributions to row lhs lower bound.
- `double * sumDown_`
Work array for sum of finite contributions to row lhs lower bound.

Friends

- `void assignPresolveToPostsolve (CoinPresolveMatrix * &preObj)`
Initialize a [CoinPostsolveMatrix](#) object, destroying the [CoinPresolveMatrix](#) object.

Related Functions

(Note that these are not member functions.)

- `void presolve_no_dups (const CoinPresolveMatrix *preObj, bool doCol=true, bool doRow=true)`
Check column-major and/or row-major matrices for duplicate entries in the major vectors.
- `void presolve_links_ok (const CoinPresolveMatrix *preObj, bool doCol=true, bool doRow=true)`
Check the links which track storage order for major vectors in the bulk storage area.
- `void presolve_no_zeros (const CoinPresolveMatrix *preObj, bool doCol=true, bool doRow=true)`
Check for explicit zeros in the column- and/or row-major matrices.
- `void presolve_consistent (const CoinPresolveMatrix *preObj, bool chkvals=true)`
Checks for equivalence of the column- and row-major matrices.
- `void presolve_check_sol (const CoinPresolveMatrix *preObj, int chkColSol=2, int chkRowAct=1, int chkStatus=1)`
Check primal solution and architectural variable status.
- `void presolve_check_nbasic (const CoinPresolveMatrix *preObj)`
Check for the proper number of basic variables.

Additional Inherited Members

9.65.1 Detailed Description

Augments [CoinPrePostsolveMatrix](#) with information about the problem that is only needed during presolve.

For problem manipulation, this class adds a row-major matrix representation, linked lists that allow for easy manipulation of the matrix when applying presolve transforms, and vectors to track row and column processing status (changed, needs further processing, change prohibited)

For problem representation, this class adds information about variable type (integer or continuous), an objective offset, and a feasibility tolerance.

NOTE that the [anyInteger_](#) and [anyProhibited_](#) flags are independent of the vectors used to track this information for individual variables ([integerType_](#) and [rowChanged_](#) and [colChanged_](#), respectively).

NOTE also that at the end of presolve the column-major and row-major matrix representations are loosely packed (*i.e.*, there may be gaps between columns in the bulk storage arrays).

NOTE that while you might think that CoinPresolve is prepared to handle minimisation or maximisation, it's unlikely that this still works. This is a good thing: better to convert objective coefficients and duals once, before starting presolve, rather than doing it over and over in each transform that considers dual variables.

The constructors that take an OSI or ClpSimplex as a parameter really should not be here, but for historical reasons they will likely remain for the foreseeable future. – lh, 111202 –

Definition at line 835 of file CoinPresolveMatrix.hpp.

9.65.2 Constructor & Destructor Documentation

9.65.2.1 `CoinPresolveMatrix::CoinPresolveMatrix (int ncols_alloc, int nrows_alloc, CoinBigIndex nelems_alloc)`

'Native' constructor

This constructor creates an empty object which must then be loaded. On the other hand, it doesn't assume that the client is an `OsiSolverInterface`.

9.65.2.2 `CoinPresolveMatrix::CoinPresolveMatrix (int ncols0, double maxmin, ClpSimplex * si, int nrows, CoinBigIndex nelems, bool doStatus, double nonLinearVariable, double bulkRatio)`

Clp OSI constructor.

See Clp code for the definition.

9.65.2.3 `CoinPresolveMatrix::CoinPresolveMatrix (int ncols0, double maxmin, OsiSolverInterface * si, int nrows, CoinBigIndex nelems, bool doStatus, double nonLinearVariable, const char * prohibited, const char * rowProhibited = NULL)`

Generic OSI constructor.

See OSI code for the definition.

9.65.2.4 `CoinPresolveMatrix::~~CoinPresolveMatrix ()`

Destructor.

9.65.3 Member Function Documentation

9.65.3.1 `void CoinPresolveMatrix::update_model (ClpSimplex * si, int nrows0, int ncols0, CoinBigIndex nelems0)`

Update the model held by a Clp OSI.

9.65.3.2 `void CoinPresolveMatrix::update_model (OsiSolverInterface * si, int nrows0, int ncols0, CoinBigIndex nelems0)`

Update the model held by a generic OSI.

9.65.3.3 `void CoinPresolveMatrix::setMatrix (const CoinPackedMatrix * mtx)`

Load the coefficient matrix.

Load the coefficient matrix before loading the other vectors (bounds, objective, variable type) required to define the problem.

9.65.3.4 `int CoinPresolveMatrix::countEmptyRows ()` `[inline]`

Count number of empty rows.

Definition at line 913 of file `CoinPresolveMatrix.hpp`.

9.65.3.5 `void CoinPresolveMatrix::setVariableType (int i, int variableType)` `[inline]`

Set variable type information for a single variable.

Set `variableType` to 0 for continuous, 1 for integer. Does not manipulate the `anyInteger_` flag.

Definition at line 923 of file `CoinPresolveMatrix.hpp`.

9.65.3.6 `void CoinPresolveMatrix::setVariableType (const unsigned char * variableType, int lenParam)`

Set variable type information for all variables.

Set `variableType[i]` to 0 for continuous, 1 for integer. Does not manipulate the [anyInteger_](#) flag.

9.65.3.7 `void CoinPresolveMatrix::setVariableType (bool allIntegers, int lenParam)`

Set the type of all variables.

`allIntegers` should be true to set the type to integer, false to set the type to continuous.

9.65.3.8 `void CoinPresolveMatrix::setAnyInteger (bool anyInteger = true) [inline]`

Set a flag for presence (true) or absence (false) of integer variables.

Definition at line 942 of file `CoinPresolveMatrix.hpp`.

9.65.3.9 `const CoinBigIndex* CoinPresolveMatrix::getRowStarts () const [inline]`

Get row start vector for row-major packed matrix.

Definition at line 951 of file `CoinPresolveMatrix.hpp`.

9.65.3.10 `const int* CoinPresolveMatrix::getColIndicesByRow () const [inline]`

Get vector of column indices for row-major packed matrix.

Definition at line 954 of file `CoinPresolveMatrix.hpp`.

9.65.3.11 `const double* CoinPresolveMatrix::getElementsByRow () const [inline]`

Get vector of elements for row-major packed matrix.

Definition at line 957 of file `CoinPresolveMatrix.hpp`.

9.65.3.12 `bool CoinPresolveMatrix::isInteger (int i) const [inline]`

Check for integrality of the specified variable.

Consults the [integerType_](#) vector if present; fallback is the [anyInteger_](#) flag.

Definition at line 965 of file `CoinPresolveMatrix.hpp`.

9.65.3.13 `bool CoinPresolveMatrix::anyInteger () const [inline]`

Check if there are any integer variables.

Consults the [anyInteger_](#) flag

Definition at line 978 of file `CoinPresolveMatrix.hpp`.

9.65.3.14 `int CoinPresolveMatrix::presolveOptions () const [inline]`

Picks up any special options.

Definition at line 981 of file `CoinPresolveMatrix.hpp`.

9.65.3.15 `void CoinPresolveMatrix::setPresolveOptions (int value) [inline]`

Sets any special options (see [presolveOptions_](#))

Definition at line 984 of file `CoinPresolveMatrix.hpp`.

9.65.3.16 void CoinPresolveMatrix::change_bias (double *change_amount*) [inline]

Adjust objective function constant offset.

Definition at line 1005 of file CoinPresolveMatrix.hpp.

9.65.3.17 void CoinPresolveMatrix::statistics ()

Say we want statistics - also set time.

9.65.3.18 double CoinPresolveMatrix::feasibilityTolerance () [inline]

Return feasibility tolerance.

Definition at line 1052 of file CoinPresolveMatrix.hpp.

9.65.3.19 void CoinPresolveMatrix::setFeasibilityTolerance (double *val*) [inline]

Set feasibility tolerance.

Definition at line 1055 of file CoinPresolveMatrix.hpp.

9.65.3.20 int CoinPresolveMatrix::status () [inline]

Returns problem status (0 = feasible, 1 = infeasible, 2 = unbounded)

Definition at line 1065 of file CoinPresolveMatrix.hpp.

9.65.3.21 void CoinPresolveMatrix::setStatus (int *status*) [inline]

Set problem status.

Definition at line 1068 of file CoinPresolveMatrix.hpp.

9.65.3.22 void CoinPresolveMatrix::setPass (int *pass* = 0) [inline]

Set pass number.

Definition at line 1080 of file CoinPresolveMatrix.hpp.

9.65.3.23 void CoinPresolveMatrix::setMaximumSubstitutionLevel (int *level*) [inline]

Set Maximum substitution level (normally 3)

Definition at line 1089 of file CoinPresolveMatrix.hpp.

9.65.3.24 int CoinPresolveMatrix::recomputeSums (int *whichRow*)

Recompute row lhs bounds.

Calculate finite contributions to row lhs upper and lower bounds and count infinite contributions. Returns the number of rows found to be infeasible.

If *whichRow* < 0, bounds are recomputed for all rows.

As of 110611, this seems to be a work in progress in the sense that it's barely used by the existing presolve code.

9.65.3.25 void CoinPresolveMatrix::initializeStuff ()

Allocate scratch arrays.

9.65.3.26 void CoinPresolveMatrix::deleteStuff ()

Free scratch arrays.

9.65.3.27 void CoinPresolveMatrix::initColsToDo ()

Initialise the column ToDo lists.

Places all columns in the `colsToDo_` list except for columns marked as prohibited (viz. `colChanged_`).

9.65.3.28 int CoinPresolveMatrix::stepColsToDo ()

Step column ToDo lists.

Moves columns on the `nextColsToDo_` list to the `colsToDo_` list, emptying `nextColsToDo_`. Returns the number of columns transferred.

9.65.3.29 int CoinPresolveMatrix::numberColsToDo () [inline]

Return the number of columns on the `colsToDo_` list.

Definition at line 1237 of file `CoinPresolveMatrix.hpp`.

9.65.3.30 bool CoinPresolveMatrix::colChanged (int i) const [inline]

Has column been changed?

Definition at line 1241 of file `CoinPresolveMatrix.hpp`.

9.65.3.31 void CoinPresolveMatrix::unsetColChanged (int i) [inline]

Mark column as not changed.

Definition at line 1245 of file `CoinPresolveMatrix.hpp`.

9.65.3.32 void CoinPresolveMatrix::setColChanged (int i) [inline]

Mark column as changed.

Definition at line 1249 of file `CoinPresolveMatrix.hpp`.

9.65.3.33 void CoinPresolveMatrix::addCol (int i) [inline]

Mark column as changed and add to list of columns to process next.

Definition at line 1253 of file `CoinPresolveMatrix.hpp`.

9.65.3.34 bool CoinPresolveMatrix::colProhibited (int i) const [inline]

Test if column is eligible for preprocessing.

Definition at line 1260 of file `CoinPresolveMatrix.hpp`.

9.65.3.35 bool CoinPresolveMatrix::colProhibited2 (int i) const [inline]

Test if column is eligible for preprocessing.

The difference between this method and `colProhibited()` is that this method first tests `anyProhibited_` before examining the specific entry for the specified column.

Definition at line 1269 of file `CoinPresolveMatrix.hpp`.

9.65.3.36 void CoinPresolveMatrix::setColProhibited (int i) [inline]

Mark column as ineligible for preprocessing.

Definition at line 1276 of file CoinPresolveMatrix.hpp.

9.65.3.37 bool CoinPresolveMatrix::colUsed (int i) const [inline]

Test if column is marked as used.

This is for doing faster lookups to see where two columns have entries in common.

Definition at line 1284 of file CoinPresolveMatrix.hpp.

9.65.3.38 void CoinPresolveMatrix::setColUsed (int i) [inline]

Mark column as used.

Definition at line 1288 of file CoinPresolveMatrix.hpp.

9.65.3.39 void CoinPresolveMatrix::unsetColUsed (int i) [inline]

Mark column as unused.

Definition at line 1292 of file CoinPresolveMatrix.hpp.

9.65.3.40 bool CoinPresolveMatrix::colInfinite (int i) const [inline]

Has column infinite ub (originally)

Definition at line 1296 of file CoinPresolveMatrix.hpp.

9.65.3.41 void CoinPresolveMatrix::unsetCollInfinite (int i) [inline]

Mark column as not infinite ub (originally)

Definition at line 1300 of file CoinPresolveMatrix.hpp.

9.65.3.42 void CoinPresolveMatrix::setCollInfinite (int i) [inline]

Mark column as infinite ub (originally)

Definition at line 1304 of file CoinPresolveMatrix.hpp.

9.65.3.43 void CoinPresolveMatrix::initRowsToDo ()

Initialise the row ToDo lists.

Places all rows in the [rowsToDo_](#) list except for rows marked as prohibited (viz. [rowChanged_](#)).

9.65.3.44 int CoinPresolveMatrix::stepRowsToDo ()

Step row ToDo lists.

Moves rows on the [nextRowsToDo_](#) list to the [rowsToDo_](#) list, emptying [nextRowsToDo_](#). Returns the number of rows transferred.

9.65.3.45 int CoinPresolveMatrix::numberOfRowsToDo () [inline]

Return the number of rows on the [rowsToDo_](#) list.

Definition at line 1323 of file CoinPresolveMatrix.hpp.

9.65.3.46 `bool CoinPresolveMatrix::rowChanged (int i) const` `[inline]`

Has row been changed?

Definition at line 1327 of file CoinPresolveMatrix.hpp.

9.65.3.47 `void CoinPresolveMatrix::unsetRowChanged (int i)` `[inline]`

Mark row as not changed.

Definition at line 1331 of file CoinPresolveMatrix.hpp.

9.65.3.48 `void CoinPresolveMatrix::setRowChanged (int i)` `[inline]`

Mark row as changed.

Definition at line 1335 of file CoinPresolveMatrix.hpp.

9.65.3.49 `void CoinPresolveMatrix::addRow (int i)` `[inline]`

Mark row as changed and add to list of rows to process next.

Definition at line 1339 of file CoinPresolveMatrix.hpp.

9.65.3.50 `bool CoinPresolveMatrix::rowProhibited (int i) const` `[inline]`

Test if row is eligible for preprocessing.

Definition at line 1346 of file CoinPresolveMatrix.hpp.

9.65.3.51 `bool CoinPresolveMatrix::rowProhibited2 (int i) const` `[inline]`

Test if row is eligible for preprocessing.

The difference between this method and [rowProhibited\(\)](#) is that this method first tests [anyProhibited_](#) before examining the specific entry for the specified row.

Definition at line 1355 of file CoinPresolveMatrix.hpp.

9.65.3.52 `void CoinPresolveMatrix::setRowProhibited (int i)` `[inline]`

Mark row as ineligible for preprocessing.

Definition at line 1362 of file CoinPresolveMatrix.hpp.

9.65.3.53 `bool CoinPresolveMatrix::rowUsed (int i) const` `[inline]`

Test if row is marked as used.

This is for doing faster lookups to see where two rows have entries in common. It can be used anywhere as long as it ends up zeroed out.

Definition at line 1370 of file CoinPresolveMatrix.hpp.

9.65.3.54 `void CoinPresolveMatrix::setRowUsed (int i)` `[inline]`

Mark row as used.

Definition at line 1374 of file CoinPresolveMatrix.hpp.

9.65.3.55 `void CoinPresolveMatrix::unsetRowUsed (int i)` `[inline]`

Mark row as unused.

Definition at line 1378 of file `CoinPresolveMatrix.hpp`.

9.65.3.56 `bool CoinPresolveMatrix::anyProhibited () const` `[inline]`

Check if there are any prohibited rows or columns.

Definition at line 1384 of file `CoinPresolveMatrix.hpp`.

9.65.3.57 `void CoinPresolveMatrix::setAnyProhibited (bool val =true)` `[inline]`

Set a flag for presence of prohibited rows or columns.

Definition at line 1387 of file `CoinPresolveMatrix.hpp`.

9.65.4 Friends And Related Function Documentation

9.65.4.1 `void assignPresolveToPostsolve (CoinPresolveMatrix * & preObj)` `[friend]`

Initialize a [CoinPostsolveMatrix](#) object, destroying the [CoinPresolveMatrix](#) object.

See [CoinPostsolveMatrix::assignPresolveToPostsolve](#).

9.65.5 Member Data Documentation

9.65.5.1 `presolvehlink* CoinPresolveMatrix::clink_`

Linked list for the column-major representation.

Definition at line 996 of file `CoinPresolveMatrix.hpp`.

9.65.5.2 `presolvehlink* CoinPresolveMatrix::rlink_`

Linked list for the row-major representation.

Definition at line 998 of file `CoinPresolveMatrix.hpp`.

9.65.5.3 `double CoinPresolveMatrix::dobias_`

Objective function offset introduced during presolve.

Definition at line 1002 of file `CoinPresolveMatrix.hpp`.

9.65.5.4 `CoinBigIndex* CoinPresolveMatrix::mrstrt_`

Vector of row start positions in `#hcol`, [rowels_](#).

Definition at line 1025 of file `CoinPresolveMatrix.hpp`.

9.65.5.5 `int* CoinPresolveMatrix::hinrow_`

Vector of row lengths.

Definition at line 1027 of file `CoinPresolveMatrix.hpp`.

9.65.5.6 double* CoinPresolveMatrix::rowels_

Coefficients (positional correspondence with [hcol_](#))

Definition at line 1029 of file CoinPresolveMatrix.hpp.

9.65.5.7 int* CoinPresolveMatrix::hcol_

Column indices (positional correspondence with [rowels_](#))

Definition at line 1031 of file CoinPresolveMatrix.hpp.

9.65.5.8 unsigned char* CoinPresolveMatrix::integerType_

Tracks integrality of columns (1 for integer, 0 for continuous)

Definition at line 1035 of file CoinPresolveMatrix.hpp.

9.65.5.9 bool CoinPresolveMatrix::anyInteger_

Flag to say if any variables are integer.

Note that this flag is *not* manipulated by the various `setVariableType` routines.

Definition at line 1041 of file CoinPresolveMatrix.hpp.

9.65.5.10 bool CoinPresolveMatrix::tuning_

Print statistics for tuning.

Definition at line 1043 of file CoinPresolveMatrix.hpp.

9.65.5.11 double CoinPresolveMatrix::startTime_

Start time of presolve.

Definition at line 1047 of file CoinPresolveMatrix.hpp.

9.65.5.12 double CoinPresolveMatrix::feasibilityTolerance_

Bounds can be moved by this to retain feasibility.

Definition at line 1050 of file CoinPresolveMatrix.hpp.

9.65.5.13 int CoinPresolveMatrix::status_

Output status: 0 = feasible, 1 = infeasible, 2 = unbounded.

Actually implemented as single bit flags: 1^0 = infeasible, 1^1 = unbounded.

Definition at line 1063 of file CoinPresolveMatrix.hpp.

9.65.5.14 int CoinPresolveMatrix::pass_

Presolve pass number.

Should be incremented externally by the method controlling application of presolve transforms. Used to control the execution of `testRedundant` (evoked by the `implied_free` transform).

Definition at line 1078 of file CoinPresolveMatrix.hpp.

9.65.5.15 int CoinPresolveMatrix::maxSubstLevel_

Maximum substitution level.

Used to control the execution of subst from implied_free

Definition at line 1087 of file CoinPresolveMatrix.hpp.

9.65.5.16 unsigned char* CoinPresolveMatrix::colChanged_

Column change status information.

Coded using the following bits:

- 0x01: Column has changed
- 0x02: preprocessing prohibited
- 0x04: Column has been used
- 0x08: Column originally had infinite ub

Definition at line 1116 of file CoinPresolveMatrix.hpp.

9.65.5.17 int* CoinPresolveMatrix::colsToDo_

Input list of columns to process.

Definition at line 1118 of file CoinPresolveMatrix.hpp.

9.65.5.18 int CoinPresolveMatrix::numberColsToDo_

Length of [colsToDo_](#).

Definition at line 1120 of file CoinPresolveMatrix.hpp.

9.65.5.19 int* CoinPresolveMatrix::nextColsToDo_

Output list of columns to process next.

Definition at line 1122 of file CoinPresolveMatrix.hpp.

9.65.5.20 int CoinPresolveMatrix::numberNextColsToDo_

Length of [nextColsToDo_](#).

Definition at line 1124 of file CoinPresolveMatrix.hpp.

9.65.5.21 unsigned char* CoinPresolveMatrix::rowChanged_

Row change status information.

Coded using the following bits:

- 0x01: Row has changed
- 0x02: preprocessing prohibited
- 0x04: Row has been used

Definition at line 1135 of file CoinPresolveMatrix.hpp.

9.65.5.22 int* CoinPresolveMatrix::rowsToDo_

Input list of rows to process.

Definition at line 1137 of file CoinPresolveMatrix.hpp.

9.65.5.23 int CoinPresolveMatrix::numberOfRowsToDo_

Length of [rowsToDo_](#).

Definition at line 1139 of file CoinPresolveMatrix.hpp.

9.65.5.24 int* CoinPresolveMatrix::nextRowsToDo_

Output list of rows to process next.

Definition at line 1141 of file CoinPresolveMatrix.hpp.

9.65.5.25 int CoinPresolveMatrix::numberOfNextRowsToDo_

Length of [nextRowsToDo_](#).

Definition at line 1143 of file CoinPresolveMatrix.hpp.

9.65.5.26 int CoinPresolveMatrix::presolveOptions_

Fine control over presolve actions.

Set/clear the following bits to allow or suppress actions:

- 0x01 allow duplicate column tests for integer variables
- 0x02 not used
- 0x04 set to inhibit $x+y+z=1$ mods
- 0x08 not used
- 0x10 set to allow stuff which won't unroll easily (overlapping duplicate rows; opportunistic fixing of variables from bound propagation).
- 0x04000 allow presolve transforms to arbitrarily ignore infeasibility and set arbitrary feasible bounds.
- 0x10000 instructs [implied_free_action](#) to be 'more lightweight'; will return without doing anything after 15 presolve passes.
- 0x20000 instructs [implied_free_action](#) to remove small created elements
- 0x80000000 set by presolve to say [dupcol_action](#) compressed columns

Definition at line 1161 of file CoinPresolveMatrix.hpp.

9.65.5.27 bool CoinPresolveMatrix::anyProhibited_

Flag to say if any rows or columns are marked as prohibited.

Note that this flag is *not* manipulated by any of the various `set*Prohibited` routines.

Definition at line 1167 of file CoinPresolveMatrix.hpp.

9.65.5.28 int* CoinPresolveMatrix::usefulRowInt_

Preallocated scratch work array, 3*nrows_.

Definition at line 1181 of file CoinPresolveMatrix.hpp.

9.65.5.29 double* CoinPresolveMatrix::usefulRowDouble_

Preallocated scratch work array, 2*nrows_.

Definition at line 1183 of file CoinPresolveMatrix.hpp.

9.65.5.30 int* CoinPresolveMatrix::usefulColumnInt_

Preallocated scratch work array, 2*ncols_.

Definition at line 1185 of file CoinPresolveMatrix.hpp.

9.65.5.31 double* CoinPresolveMatrix::usefulColumnDouble_

Preallocated scratch work array, ncols_.

Definition at line 1187 of file CoinPresolveMatrix.hpp.

9.65.5.32 double* CoinPresolveMatrix::randomNumber_

Array of random numbers (max row,column)

Definition at line 1189 of file CoinPresolveMatrix.hpp.

9.65.5.33 int* CoinPresolveMatrix::infiniteUp_

Work array for count of infinite contributions to row lhs upper bound.

Definition at line 1192 of file CoinPresolveMatrix.hpp.

9.65.5.34 double* CoinPresolveMatrix::sumUp_

Work array for sum of finite contributions to row lhs upper bound.

Definition at line 1194 of file CoinPresolveMatrix.hpp.

9.65.5.35 int* CoinPresolveMatrix::infiniteDown_

Work array for count of infinite contributions to row lhs lower bound.

Definition at line 1196 of file CoinPresolveMatrix.hpp.

9.65.5.36 double* CoinPresolveMatrix::sumDown_

Work array for sum of finite contributions to row lhs lower bound.

Definition at line 1198 of file CoinPresolveMatrix.hpp.

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinPresolveMatrix.hpp](#)
- /home/ted/COIN/trunk/CoinUtils/src/[CoinPresolvePsdebug.hpp](#)

9.66 CoinPresolveMonitor Class Reference

Monitor a row or column for modification.

```
#include <CoinPresolveMonitor.hpp>
```

Public Member Functions

- [CoinPresolveMonitor](#) ()
Default constructor.
- [CoinPresolveMonitor](#) (const [CoinPresolveMatrix](#) *mtx, bool isRow, int k)
Initialise from a [CoinPresolveMatrix](#).
- [CoinPresolveMonitor](#) (const [CoinPostsolveMatrix](#) *mtx, bool isRow, int k)
Initialise from a [CoinPostsolveMatrix](#).
- [void checkAndTell](#) (const [CoinPresolveMatrix](#) *mtx)
Compare the present row or column against the stored copy and report differences.
- [void checkAndTell](#) (const [CoinPostsolveMatrix](#) *mtx)
Compare the present row or column against the stored copy and report differences.

9.66.1 Detailed Description

Monitor a row or column for modification.

The purpose of this class is to monitor a row or column for modifications during presolve and postsolve. Each object can monitor one row or column. The initial copy of the row or column is loaded by the constructor. Each subsequent call to [checkAndTell\(\)](#) compares the current state of the row or column with the stored state and reports any modifications.

Internally the row or column is held as a [CoinPackedVector](#) so that it's possible to follow a row or column through presolve ([CoinPresolveMatrix](#)) and postsolve ([CoinPostsolveMatrix](#)).

Do not underestimate the amount of work required here. Extracting a row from the [CoinPostsolveMatrix](#) requires a scan of every element in the matrix. That's one scan by the constructor and one scan with every call to modify. But that's precisely why it's virtually impossible to debug presolve without aids.

Parameter overloads for [CoinPresolveMatrix](#) and [CoinPostsolveMatrix](#) are a little clumsy, but not a problem in use. The alternative is to add methods to the [CoinPresolveMatrix](#) and [CoinPostsolveMatrix](#) classes that will only be used for debugging. That's not too attractive either.

Definition at line 29 of file [CoinPresolveMonitor.hpp](#).

9.66.2 Constructor & Destructor Documentation

9.66.2.1 [CoinPresolveMonitor::CoinPresolveMonitor](#) ()

Default constructor.

Creates an empty monitor.

9.66.2.2 [CoinPresolveMonitor::CoinPresolveMonitor](#) (const [CoinPresolveMatrix](#) * mtx, bool isRow, int k)

Initialise from a [CoinPresolveMatrix](#).

Load the initial row or column from a [CoinPresolveMatrix](#). Set `isRow` true for a row, false for a column.

9.66.2.3 CoinPresolveMonitor::CoinPresolveMonitor (const CoinPostsolveMatrix * mtx, bool isRow, int k)

Initialise from a [CoinPostsolveMatrix](#).

Load the initial row or column from a [CoinPostsolveMatrix](#). Set `isRow` true for a row, false for a column.

9.66.3 Member Function Documentation

9.66.3.1 void CoinPresolveMonitor::checkAndTell (const CoinPresolveMatrix * mtx)

Compare the present row or column against the stored copy and report differences.

Load the current row or column from a [CoinPresolveMatrix](#) and compare. Differences are printed to `std::cout`.

9.66.3.2 void CoinPresolveMonitor::checkAndTell (const CoinPostsolveMatrix * mtx)

Compare the present row or column against the stored copy and report differences.

Load the current row or column from a [CoinPostsolveMatrix](#) and compare. Differences are printed to `std::cout`.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMonitor.hpp](#)

9.67 CoinRelFltEq Class Reference

Equality to a scaled tolerance.

```
#include <CoinFloatEqual.hpp>
```

Public Member Functions

- `bool operator() (const double f1, const double f2) const`
Compare function.

Constructors and destructors

- `CoinRelFltEq ()`
Default constructor.
- `CoinRelFltEq (const double epsilon)`
Alternate constructor with epsilon as a parameter.
- `virtual ~CoinRelFltEq ()`
Destructor.
- `CoinRelFltEq (const CoinRelFltEq &src)`
Copy constructor.
- `CoinRelFltEq & operator= (const CoinRelFltEq &rhs)`
Assignment.

9.67.1 Detailed Description

Equality to a scaled tolerance.

Operands are considered equal if their difference is within a scaled epsilon calculated as `epsilon_*(1+CoinMax(|f1|,|f2|))`.

Definition at line 110 of file `CoinFloatEqual.hpp`.

9.67.2 Constructor & Destructor Documentation

9.67.2.1 CoinRelFitEq::CoinRelFitEq () [inline]

Default constructor.

Default tolerance is 1.0e-10.

Definition at line 134 of file CoinFloatEqual.hpp.

9.67.2.2 CoinRelFitEq::CoinRelFitEq (const double *epsilon*) [inline]

Alternate constructor with epsilon as a parameter.

Definition at line 145 of file CoinFloatEqual.hpp.

9.67.2.3 virtual CoinRelFitEq::~CoinRelFitEq () [inline], [virtual]

Destructor.

Definition at line 149 of file CoinFloatEqual.hpp.

9.67.2.4 CoinRelFitEq::CoinRelFitEq (const CoinRelFitEq & *src*) [inline]

Copy constructor.

Definition at line 153 of file CoinFloatEqual.hpp.

9.67.3 Member Function Documentation

9.67.3.1 bool CoinRelFitEq::operator() (const double *f1*, const double *f2*) const [inline]

Compare function.

Definition at line 116 of file CoinFloatEqual.hpp.

9.67.3.2 CoinRelFitEq& CoinRelFitEq::operator= (const CoinRelFitEq & *rhs*) [inline]

Assignment.

Definition at line 157 of file CoinFloatEqual.hpp.

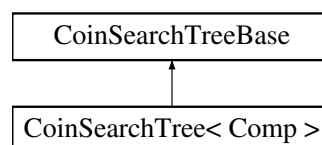
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/CoinUtils/src/CoinFloatEqual.hpp](#)

9.68 CoinSearchTree< Comp > Class Template Reference

```
#include <CoinSearchTree.hpp>
```

Inheritance diagram for CoinSearchTree< Comp >:



Public Member Functions

- [CoinSearchTree](#) ()
- [CoinSearchTree](#) (const [CoinSearchTreeBase](#) &t)
- virtual [~CoinSearchTree](#) ()
- const char * [compName](#) () const

Protected Member Functions

- virtual [void realpop](#) ()
 - virtual [void fixTop](#) ()
 - virtual [void realpush](#) ([CoinTreeSiblings](#) *s)
- After changing data in the top node, fix the heap.*

Additional Inherited Members**9.68.1 Detailed Description**

`template<class Comp>class CoinSearchTree< Comp >`

Definition at line 331 of file `CoinSearchTree.hpp`.

9.68.2 Constructor & Destructor Documentation

9.68.2.1 `template<class Comp > CoinSearchTree< Comp >::CoinSearchTree () [inline]`

Definition at line 382 of file `CoinSearchTree.hpp`.

9.68.2.2 `template<class Comp > CoinSearchTree< Comp >::CoinSearchTree (const CoinSearchTreeBase & t) [inline]`

Definition at line 383 of file `CoinSearchTree.hpp`.

9.68.2.3 `template<class Comp > virtual CoinSearchTree< Comp >::~~CoinSearchTree () [inline],[virtual]`

Definition at line 390 of file `CoinSearchTree.hpp`.

9.68.3 Member Function Documentation

9.68.3.1 `template<class Comp > virtual void CoinSearchTree< Comp >::realtop () [inline],[protected],[virtual]`

Implements [CoinSearchTreeBase](#).

Definition at line 337 of file `CoinSearchTree.hpp`.

9.68.3.2 `template<class Comp > virtual void CoinSearchTree< Comp >::fixTop () [inline],[protected],[virtual]`

After changing data in the top node, fix the heap.

Implements [CoinSearchTreeBase](#).

Definition at line 343 of file `CoinSearchTree.hpp`.

9.68.3.3 `template<class Comp > virtual void CoinSearchTree< Comp >::realpush (CoinTreeSiblings * s)`
`[inline], [protected], [virtual]`

Implements [CoinSearchTreeBase](#).

Definition at line 367 of file `CoinSearchTree.hpp`.

9.68.3.4 `template<class Comp > const char* CoinSearchTree< Comp >::compName () const` `[inline],`
`[virtual]`

Implements [CoinSearchTreeBase](#).

Definition at line 391 of file `CoinSearchTree.hpp`.

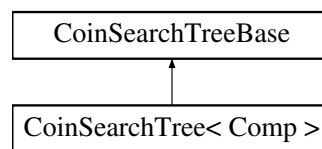
The documentation for this class was generated from the following file:

- `/home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp`

9.69 CoinSearchTreeBase Class Reference

`#include <CoinSearchTree.hpp>`

Inheritance diagram for `CoinSearchTreeBase`:



Public Member Functions

- `virtual ~CoinSearchTreeBase ()`
- `virtual const char * compName () const =0`
- `const std::vector< CoinTreeSiblings * > & getCandidates () const`
- `bool empty () const`
- `int size () const`
- `int numInserted () const`
- `CoinTreeNode * top () const`
- `void pop ()`
pop will advance the next pointer among the siblings on the top and then moves the top to its correct position.
- `void push (int numNodes, CoinTreeNode **nodes, const bool incrInserted=true)`
- `void push (const CoinTreeSiblings &sib, const bool incrInserted=true)`

Protected Member Functions

- `CoinSearchTreeBase ()`
- `virtual void realpop ()=0`
- `virtual void realpush (CoinTreeSiblings *s)=0`
- `virtual void fixTop ()=0`

Protected Attributes

- `std::vector< CoinTreeSiblings * > candidateList_`
- `int numInserted_`
- `int size_`

9.69.1 Detailed Description

Definition at line 217 of file `CoinSearchTree.hpp`.

9.69.2 Constructor & Destructor Documentation

9.69.2.1 `CoinSearchTreeBase::CoinSearchTreeBase ()` `[inline]`, `[protected]`

Definition at line 229 of file `CoinSearchTree.hpp`.

9.69.2.2 `virtual CoinSearchTreeBase::~CoinSearchTreeBase ()` `[inline]`, `[virtual]`

Definition at line 236 of file `CoinSearchTree.hpp`.

9.69.3 Member Function Documentation

9.69.3.1 `virtual void CoinSearchTreeBase::realpop ()` `[protected]`, `[pure virtual]`

Implemented in `CoinSearchTree< Comp >`.

9.69.3.2 `virtual void CoinSearchTreeBase::realpush (CoinTreeSiblings * s)` `[protected]`, `[pure virtual]`

Implemented in `CoinSearchTree< Comp >`.

9.69.3.3 `virtual void CoinSearchTreeBase::fixTop ()` `[protected]`, `[pure virtual]`

Implemented in `CoinSearchTree< Comp >`.

9.69.3.4 `virtual const char* CoinSearchTreeBase::compName () const` `[pure virtual]`

Implemented in `CoinSearchTree< Comp >`.

9.69.3.5 `const std::vector<CoinTreeSiblings*>& CoinSearchTreeBase::getCandidates () const` `[inline]`

Definition at line 239 of file `CoinSearchTree.hpp`.

9.69.3.6 `bool CoinSearchTreeBase::empty () const` `[inline]`

Definition at line 242 of file `CoinSearchTree.hpp`.

9.69.3.7 `int CoinSearchTreeBase::size () const` `[inline]`

Definition at line 243 of file `CoinSearchTree.hpp`.

9.69.3.8 `int CoinSearchTreeBase::numInserted () const` `[inline]`

Definition at line 244 of file `CoinSearchTree.hpp`.

9.69.3.9 **CoinTreeNode*** CoinSearchTreeBase::top () const [inline]

Definition at line 245 of file CoinSearchTree.hpp.

9.69.3.10 **void** CoinSearchTreeBase::pop () [inline]

pop will advance the `next` pointer among the siblings on the top and then moves the top to its correct position.

[realpop](#) is the method that actually removes the element from the heap

Definition at line 259 of file CoinSearchTree.hpp.

9.69.3.11 **void** CoinSearchTreeBase::push (int *numNodes*, **CoinTreeNode** ** *nodes*, const bool *incrInserted* = true) [inline]

Definition at line 269 of file CoinSearchTree.hpp.

9.69.3.12 **void** CoinSearchTreeBase::push (const **CoinTreeSiblings** & *sib*, const bool *incrInserted* = true) [inline]

Definition at line 278 of file CoinSearchTree.hpp.

9.69.4 Member Data Documentation

9.69.4.1 **std::vector<CoinTreeSiblings*>** CoinSearchTreeBase::candidateList_ [protected]

Definition at line 224 of file CoinSearchTree.hpp.

9.69.4.2 **int** CoinSearchTreeBase::numInserted_ [protected]

Definition at line 225 of file CoinSearchTree.hpp.

9.69.4.3 **int** CoinSearchTreeBase::size_ [protected]

Definition at line 226 of file CoinSearchTree.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp](#)

9.70 CoinSearchTreeCompareBest Struct Reference

Best first search.

```
#include <CoinSearchTree.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (const [CoinTreeSiblings](#) *x, const [CoinTreeSiblings](#) *y) const

Static Public Member Functions

- static const char * [name](#) ()

9.70.1 Detailed Description

Best first search.

Definition at line 207 of file CoinSearchTree.hpp.

9.70.2 Member Function Documentation

9.70.2.1 `static const char* CoinSearchTreeCompareBest::name () [inline],[static]`

Definition at line 208 of file CoinSearchTree.hpp.

9.70.2.2 `bool CoinSearchTreeCompareBest::operator() (const CoinTreeSiblings * x, const CoinTreeSiblings * y) const [inline]`

Definition at line 209 of file CoinSearchTree.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp](#)

9.71 CoinSearchTreeCompareBreadth Struct Reference

```
#include <CoinSearchTree.hpp>
```

Public Member Functions

- `bool operator() (const CoinTreeSiblings *x, const CoinTreeSiblings *y) const`

Static Public Member Functions

- `static const char * name ()`

9.71.1 Detailed Description

Definition at line 197 of file CoinSearchTree.hpp.

9.71.2 Member Function Documentation

9.71.2.1 `static const char* CoinSearchTreeCompareBreadth::name () [inline],[static]`

Definition at line 198 of file CoinSearchTree.hpp.

9.71.2.2 `bool CoinSearchTreeCompareBreadth::operator() (const CoinTreeSiblings * x, const CoinTreeSiblings * y) const [inline]`

Definition at line 199 of file CoinSearchTree.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp](#)

9.72 CoinSearchTreeCompareDepth Struct Reference

Depth First Search.

```
#include <CoinSearchTree.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (const [CoinTreeSiblings](#) *x, const [CoinTreeSiblings](#) *y) const

Static Public Member Functions

- static const char * [name](#) ()

9.72.1 Detailed Description

Depth First Search.

Definition at line 178 of file CoinSearchTree.hpp.

9.72.2 Member Function Documentation

9.72.2.1 static const char* CoinSearchTreeCompareDepth::name () [\[inline\]](#), [\[static\]](#)

Definition at line 179 of file CoinSearchTree.hpp.

9.72.2.2 bool CoinSearchTreeCompareDepth::operator() (const [CoinTreeSiblings](#) * x, const [CoinTreeSiblings](#) * y) const [\[inline\]](#)

Definition at line 180 of file CoinSearchTree.hpp.

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinSearchTree.hpp](#)

9.73 CoinSearchTreeComparePreferred Struct Reference

Function objects to compare search tree nodes.

```
#include <CoinSearchTree.hpp>
```

Public Member Functions

- bool [operator\(\)](#) (const [CoinTreeSiblings](#) *x, const [CoinTreeSiblings](#) *y) const

Static Public Member Functions

- static const char * [name](#) ()

9.73.1 Detailed Description

Function objects to compare search tree nodes.

The comparison function must return true if the first argument is "better" than the second one, i.e., it should be processed first. Depth First Search.

Definition at line 152 of file CoinSearchTree.hpp.

9.73.2 Member Function Documentation

9.73.2.1 `static const char* CoinSearchTreeComparePreferred::name () [inline],[static]`

Definition at line 153 of file CoinSearchTree.hpp.

9.73.2.2 `bool CoinSearchTreeComparePreferred::operator() (const CoinTreeSiblings * x, const CoinTreeSiblings * y) const [inline]`

Definition at line 154 of file CoinSearchTree.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp](#)

9.74 CoinSearchTreeManager Class Reference

```
#include <CoinSearchTree.hpp>
```

Public Member Functions

- [CoinSearchTreeManager \(\)](#)
- [virtual ~CoinSearchTreeManager \(\)](#)
- [void setTree \(CoinSearchTreeBase *t\)](#)
- [CoinSearchTreeBase * getTree \(\) const](#)
- [bool empty \(\) const](#)
- [size_t size \(\) const](#)
- [size_t numInserted \(\) const](#)
- [CoinTreeNode * top \(\) const](#)
- [void pop \(\)](#)
- [void push \(CoinTreeNode *node, const bool incrInserted=true\)](#)
- [void push \(const CoinTreeSiblings &s, const bool incrInserted=true\)](#)
- [void push \(const int n, CoinTreeNode **nodes, const bool incrInserted=true\)](#)
- [CoinTreeNode * bestQualityCandidate \(\) const](#)
- [double bestQuality \(\) const](#)
- [void newSolution \(double solValue\)](#)
- [void reevaluateSearchStrategy \(\)](#)

9.74.1 Detailed Description

Definition at line 404 of file CoinSearchTree.hpp.

9.74.2 Constructor & Destructor Documentation

9.74.2.1 CoinSearchTreeManager::CoinSearchTreeManager () [inline]

Definition at line 420 of file CoinSearchTree.hpp.

9.74.2.2 virtual CoinSearchTreeManager::~CoinSearchTreeManager () [inline],[virtual]

Definition at line 425 of file CoinSearchTree.hpp.

9.74.3 Member Function Documentation

9.74.3.1 void CoinSearchTreeManager::setTree (CoinSearchTreeBase * t) [inline]

Definition at line 429 of file CoinSearchTree.hpp.

9.74.3.2 CoinSearchTreeBase* CoinSearchTreeManager::getTree () const [inline]

Definition at line 433 of file CoinSearchTree.hpp.

9.74.3.3 bool CoinSearchTreeManager::empty () const [inline]

Definition at line 437 of file CoinSearchTree.hpp.

9.74.3.4 size_t CoinSearchTreeManager::size () const [inline]

Definition at line 438 of file CoinSearchTree.hpp.

9.74.3.5 size_t CoinSearchTreeManager::numInserted () const [inline]

Definition at line 439 of file CoinSearchTree.hpp.

9.74.3.6 CoinTreeNode* CoinSearchTreeManager::top () const [inline]

Definition at line 440 of file CoinSearchTree.hpp.

9.74.3.7 void CoinSearchTreeManager::pop () [inline]

Definition at line 441 of file CoinSearchTree.hpp.

9.74.3.8 void CoinSearchTreeManager::push (CoinTreeNode * node, const bool *incrInserted* = true) [inline]

Definition at line 442 of file CoinSearchTree.hpp.

9.74.3.9 void CoinSearchTreeManager::push (const CoinTreeSiblings & s, const bool *incrInserted* = true) [inline]

Definition at line 445 of file CoinSearchTree.hpp.

9.74.3.10 void CoinSearchTreeManager::push (const int n, CoinTreeNode ** nodes, const bool *incrInserted* = true) [inline]

Definition at line 448 of file CoinSearchTree.hpp.

9.74.3.11 CoinTreeNode* CoinSearchTreeManager::bestQualityCandidate () const [inline]

Definition at line 453 of file CoinSearchTree.hpp.

9.74.3.12 `double CoinSearchTreeManager::bestQuality () const [inline]`

Definition at line 456 of file CoinSearchTree.hpp.

9.74.3.13 `void CoinSearchTreeManager::newSolution (double solValue)`

9.74.3.14 `void CoinSearchTreeManager::reevaluateSearchStrategy ()`

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

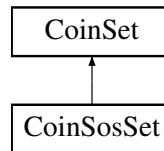
- </home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp>

9.75 CoinSet Class Reference

Very simple class for containing data on set.

```
#include <CoinMpsIO.hpp>
```

Inheritance diagram for CoinSet:



Public Member Functions

Constructor and destructor

- [CoinSet](#) ()
Default constructor.
- [CoinSet](#) (int [numberEntries](#), const int *[which](#))
Constructor.
- [CoinSet](#) (const [CoinSet](#) &)
Copy constructor.
- [CoinSet](#) & [operator=](#) (const [CoinSet](#) &rhs)
Assignment operator.
- virtual [~CoinSet](#) ()
Destructor.

gets

- int [numberEntries](#) () const
Returns number of entries.
- int [setType](#) () const
Returns type of set - 1 =SOS1, 2 =SOS2.
- const int * [which](#) () const
Returns list of variables.
- const double * [weights](#) () const
Returns weights.

Protected Attributes

data

- int `numberEntries_`
Number of entries.
- int `setType_`
type of set
- int * `which_`
Which variables are in set.
- double * `weights_`
Weights.

9.75.1 Detailed Description

Very simple class for containing data on set.

Definition at line 221 of file CoinMpsIO.hpp.

9.75.2 Constructor & Destructor Documentation

9.75.2.1 `CoinSet::CoinSet ()`

Default constructor.

9.75.2.2 `CoinSet::CoinSet (int numberEntries, const int * which)`

Constructor.

9.75.2.3 `CoinSet::CoinSet (const CoinSet &)`

Copy constructor.

9.75.2.4 `virtual CoinSet::~CoinSet () [virtual]`

Destructor.

9.75.3 Member Function Documentation

9.75.3.1 `CoinSet& CoinSet::operator= (const CoinSet & rhs)`

Assignment operator.

9.75.3.2 `int CoinSet::numberEntries () const [inline]`

Returns number of entries.

Definition at line 246 of file CoinMpsIO.hpp.

9.75.3.3 `int CoinSet::setType () const [inline]`

Returns type of set - 1 =SOS1, 2 =SOS2.

Definition at line 249 of file CoinMpsIO.hpp.

9.75.3.4 `const int* CoinSet::which () const [inline]`

Returns list of variables.

Definition at line 252 of file CoinMpslO.hpp.

9.75.3.5 `const double* CoinSet::weights () const [inline]`

Returns weights.

Definition at line 255 of file CoinMpslO.hpp.

9.75.4 Member Data Documentation

9.75.4.1 `int CoinSet::numberEntries_ [protected]`

Number of entries.

Definition at line 274 of file CoinMpslO.hpp.

9.75.4.2 `int CoinSet::setType_ [protected]`

type of set

Definition at line 276 of file CoinMpslO.hpp.

9.75.4.3 `int* CoinSet::which_ [protected]`

Which variables are in set.

Definition at line 278 of file CoinMpslO.hpp.

9.75.4.4 `double* CoinSet::weights_ [protected]`

Weights.

Definition at line 280 of file CoinMpslO.hpp.

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

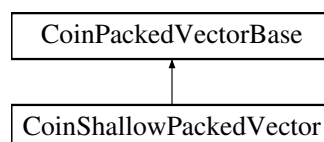
- [/home/ted/COIN/trunk/CoinUtils/src/CoinMpslO.hpp](#)

9.76 CoinShallowPackedVector Class Reference

Shallow Sparse Vector.

```
#include <CoinShallowPackedVector.hpp>
```

Inheritance diagram for CoinShallowPackedVector:



Public Member Functions

Get methods

- virtual int [getNumElements](#) () const
Get length of indices and elements vectors.
- virtual const int * [getIndices](#) () const
Get indices of elements.
- virtual const double * [getElements](#) () const
Get element values.

Set methods

- void [clear](#) ()
Reset the vector (as if were just created an empty vector)
- [CoinShallowPackedVector](#) & [operator=](#) (const [CoinShallowPackedVector](#) &x)
Assignment operator.
- [CoinShallowPackedVector](#) & [operator=](#) (const [CoinPackedVectorBase](#) &x)
Assignment operator from a [CoinPackedVectorBase](#).
- void [setVector](#) (int size, const int *indices, const double *elements, bool [testForDuplicateIndex](#)=true)
just like the explicit constructor

Methods to create, set and destroy

- [CoinShallowPackedVector](#) (bool [testForDuplicateIndex](#)=true)
Default constructor.
- [CoinShallowPackedVector](#) (int size, const int *indices, const double *elements, bool [testForDuplicateIndex](#)=true)
Explicit Constructor.
- [CoinShallowPackedVector](#) (const [CoinPackedVectorBase](#) &)
Copy constructor from the base class.
- [CoinShallowPackedVector](#) (const [CoinShallowPackedVector](#) &)
Copy constructor.
- virtual [~CoinShallowPackedVector](#) ()
Destructor.
- void [print](#) ()
Print vector information.

Friends

- void [CoinShallowPackedVectorUnitTest](#) ()
A function that tests the methods in the [CoinShallowPackedVector](#) class.

Additional Inherited Members

9.76.1 Detailed Description

Shallow Sparse Vector.

This class is for sparse vectors where the indices and elements are stored elsewhere. This class only maintains pointers to the indices and elements. Since this class does not own the index and element data it provides read only access to to the data. An [CoinSparsePackedVector](#) must be used when the sparse vector's data will be altered.

This class stores pointers to the vectors. It does not actually contain the vectors.

Here is a sample usage:

```

const int ne = 4;
int inx[ne] = { 1, 4, 0, 2 };
double el[ne] = { 10., 40., 1., 50. };

// Create vector and set its value
CoinShallowPackedVector r(ne,inx,el);

// access each index and element
assert( r.indices()[0]==1 );
assert( r.elements()[0]==10. );
assert( r.indices()[1]==4 );
assert( r.elements()[1]==40. );
assert( r.indices()[2]==0 );
assert( r.elements()[2]==1. );
assert( r.indices()[3]==2 );
assert( r.elements()[3]==50. );

// access as a full storage vector
assert( r[0]==1. );
assert( r[1]==10. );
assert( r[2]==50. );
assert( r[3]==0. );
assert( r[4]==40. );

// Tests for equality and equivalence
CoinShallowPackedVector r1;
r1=r;
assert( r==r1 );
r.sort(CoinIncrElementOrdered());
assert( r!=r1 );

// Add packed vectors.
// Similarly for subtraction, multiplication,
// and division.
CoinPackedVector add = r + r1;
assert( add[0] == 1.+1. );
assert( add[1] == 10.+10. );
assert( add[2] == 50.+50. );
assert( add[3] == 0.+0. );
assert( add[4] == 40.+40. );
assert( r.sum() == 10.+40.+1.+50. );

```

Definition at line 74 of file CoinShallowPackedVector.hpp.

9.76.2 Constructor & Destructor Documentation

9.76.2.1 CoinShallowPackedVector::CoinShallowPackedVector (bool *testForDuplicateIndex* = true)

Default constructor.

9.76.2.2 CoinShallowPackedVector::CoinShallowPackedVector (int *size*, const int * *indices*, const double * *elements*, bool *testForDuplicateIndex* = true)

Explicit Constructor.

Set vector size, indices, and elements. Size is the length of both the indices and elements vectors. The indices and elements vectors are not copied into this class instance. The ShallowPackedVector only maintains the pointers to the indices and elements vectors.

The last argument specifies whether the creator of the object knows in advance that there are no duplicate indices.

9.76.2.3 CoinShallowPackedVector::CoinShallowPackedVector (const CoinPackedVectorBase &)

Copy constructor from the base class.

9.76.2.4 CoinShallowPackedVector::CoinShallowPackedVector (const CoinShallowPackedVector &)

Copy constructor.

9.76.2.5 virtual CoinShallowPackedVector::~CoinShallowPackedVector () [inline],[virtual]

Destructor.

Definition at line 122 of file CoinShallowPackedVector.hpp.

9.76.3 Member Function Documentation**9.76.3.1 virtual int CoinShallowPackedVector::getNumElements () const [inline],[virtual]**

Get length of indices and elements vectors.

Implements [CoinPackedVectorBase](#).

Definition at line 82 of file CoinShallowPackedVector.hpp.

9.76.3.2 virtual const int* CoinShallowPackedVector::getIndices () const [inline],[virtual]

Get indices of elements.

Implements [CoinPackedVectorBase](#).

Definition at line 84 of file CoinShallowPackedVector.hpp.

9.76.3.3 virtual const double* CoinShallowPackedVector::getElements () const [inline],[virtual]

Get element values.

Implements [CoinPackedVectorBase](#).

Definition at line 86 of file CoinShallowPackedVector.hpp.

9.76.3.4 void CoinShallowPackedVector::clear ()

Reset the vector (as if were just created an empty vector)

9.76.3.5 CoinShallowPackedVector& CoinShallowPackedVector::operator= (const CoinShallowPackedVector & x)

Assignment operator.

9.76.3.6 CoinShallowPackedVector& CoinShallowPackedVector::operator= (const CoinPackedVectorBase & x)

Assignment operator from a [CoinPackedVectorBase](#).

9.76.3.7 void CoinShallowPackedVector::setVector (int size, const int * indices, const double * elements, bool testForDuplicateIndex = true)

just like the explicit constructor

9.76.3.8 void CoinShallowPackedVector::print ()

Print vector information.

9.76.4 Friends And Related Function Documentation

9.76.4.1 void CoinShallowPackedVectorUnitTest () [friend]

A function that tests the methods in the [CoinShallowPackedVector](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

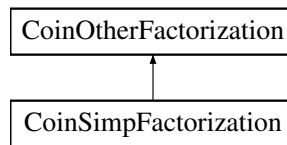
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/CoinUtils/src/CoinShallowPackedVector.hpp](#)

9.77 CoinSimpFactorization Class Reference

```
#include <CoinSimpFactorization.hpp>
```

Inheritance diagram for CoinSimpFactorization:



Public Member Functions

- [void gutsOfDestructor \(\)](#)
The real work of destructor.
- [void gutsOfInitialize \(\)](#)
The real work of constructor.
- [void gutsOfCopy \(const \[CoinSimpFactorization\]\(#\) &other\)](#)
The real work of copy.
- [void factorize \(int numberOfRows, int numberOfColumns, const int colStarts\[\], const int indicesRow\[\], const double elements\[\]\)](#)
calls factorization
- [int mainLoopFactor \(\[FactorPointers\]\(#\) &pointers\)](#)
main loop of factorization
- [void copyLbyRows \(\)](#)
copies L by rows
- [void copyUbyColumns \(\)](#)
copies U by columns
- [int findPivot \(\[FactorPointers\]\(#\) &pointers, int &r, int &s, bool &ifSlack\)](#)
finds a pivot element using Markowitz count
- [int findPivotShCol \(\[FactorPointers\]\(#\) &pointers, int &r, int &s\)](#)
finds a pivot in a shortest column
- [int findPivotSimp \(\[FactorPointers\]\(#\) &pointers, int &r, int &s\)](#)
finds a pivot in the first column available
- [void GaussEliminate \(\[FactorPointers\]\(#\) &pointers, int &r, int &s\)](#)
does Gauss elimination
- [int findShortRow \(const int column, const int length, int &minRow, int &minRowLength, \[FactorPointers\]\(#\) &pointers\)](#)

- finds short row that intersects a given column*
- `int findShortColumn` (const int row, const int length, int &minCol, int &minColLength, [FactorPointers](#) &pointers)
- finds short column that intersects a given row*
- `double findMaxInRow` (const int row, [FactorPointers](#) &pointers)
- finds maximum absolute value in a row*
- `void pivoting` (const int [pivotRow](#), const int pivotColumn, const double invPivot, [FactorPointers](#) &pointers)
- does pivoting*
- `void updateCurrentRow` (const int [pivotRow](#), const int row, const double multiplier, [FactorPointers](#) &pointers, int &newNonZeros)
- part of pivoting*
- `void increaseLsize` ()
- allocates more space for L*
- `void increaseRowSize` (const int row, const int newSize)
- allocates more space for a row of U*
- `void increaseColSize` (const int column, const int newSize, const bool b)
- allocates more space for a column of U*
- `void enlargeUrow` (const int numNewElements)
- allocates more space for rows of U*
- `void enlargeUcol` (const int numNewElements, const bool b)
- allocates more space for columns of U*
- `int findInRow` (const int row, const int column)
- finds a given row in a column*
- `int findInColumn` (const int column, const int row)
- finds a given column in a row*
- `void removeRowFromActSet` (const int row, [FactorPointers](#) &pointers)
- declares a row inactive*
- `void removeColumnFromActSet` (const int column, [FactorPointers](#) &pointers)
- declares a column inactive*
- `void allocateSpaceForU` ()
- allocates space for U*
- `void allocateSomeArrays` ()
- allocates several working arrays*
- `void initialSomeNumbers` ()
- initializes some numbers*
- `void Lxeqb` (double *b) const
- solves $Lx = b$*
- `void Lxeqb2` (double *b1, double *b2) const
- same as above but with two rhs*
- `void Uxeqb` (double *b, double *sol) const
- solves $Ux = b$*
- `void Uxeqb2` (double *b1, double *sol1, double *sol2, double *b2) const
- same as above but with two rhs*
- `void xLeqb` (double *b) const
- solves $xL = b$*
- `void xUeqb` (double *b, double *sol) const
- solves $xU = b$*
- `int LUupdate` (int newBasicCol)

- updates factorization after a Simplex iteration*
- [void newEta](#) (int row, int numNewElements)
 - creates a new eta vector*
- [void copyRowPermutations](#) ()
 - makes a copy of row permutations*
- [void Hxeqb](#) (double *b) const
 - solves $Hx = b$, where H is a product of eta matrices*
- [void Hxeqb2](#) (double *b1, double *b2) const
 - same as above but with two rhs*
- [void xHeqb](#) (double *b) const
 - solves $xH = b$*
- [void ftran](#) (double *b, double *sol, bool save) const
 - does FTRAN*
- [void ftran2](#) (double *b1, double *sol1, double *b2, double *sol2) const
 - same as above but with two columns*
- [void btran](#) (double *b, double *sol) const
 - does BTRAN*

Constructors and destructor and copy

- [CoinSimpFactorization](#) ()
 - Default constructor.*
- [CoinSimpFactorization](#) (const [CoinSimpFactorization](#) &other)
 - Copy constructor.*
- virtual [~CoinSimpFactorization](#) ()
 - Destructor.*
- [CoinSimpFactorization](#) & [operator=](#) (const [CoinSimpFactorization](#) &other)
 - = copy*
- virtual [CoinOtherFactorization](#) * [clone](#) () const
 - Clone.*

Do factorization - public

- virtual [void getAreas](#) (int [numberRows](#), int [numberColumns](#), [CoinBigIndex](#) maximumL, [CoinBigIndex](#) maximumU)
 - Gets space for a factorization.*
- virtual [void preProcess](#) ()
 - PreProcesses column ordered copy of basis.*
- virtual int [factor](#) ()
 - Does most of factorization returning status 0 - OK -99 - needs more memory -1 - singular - use numberGoodColumns and redo.*
- virtual [void postProcess](#) (const int *sequence, int *pivotVariable)
 - Does post processing on valid factorization - putting variables on correct rows.*
- virtual [void makeNonSingular](#) (int *sequence, int [numberColumns](#))
 - Makes a non-singular basis by replacing variables.*

general stuff such as status

- virtual int [numberElements](#) () const
 - Total number of elements in factorization.*
- double [maximumCoefficient](#) () const
 - Returns maximum absolute value in factorization.*

rank one updates which do exist

- virtual int `replaceColumn` (`CoinIndexedVector` *regionSparse, int `pivotRow`, double `pivotCheck`, bool `checkBeforeModifying=false`, double `acceptablePivot=1.0e-8`)
Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If checkBeforeModifying is true will do all accuracy checks before modifying factorization.

various uses of factorization (return code number elements)

which user may want to know about

- virtual int `updateColumnFT` (`CoinIndexedVector` *regionSparse, `CoinIndexedVector` *regionSparse2, bool `noPermute=false`)
Updates one column (FTRAN) from regionSparse2 Tries to do FT update number returned is negative if no room regionSparse starts as zero and is zero at end.
- virtual int `updateColumn` (`CoinIndexedVector` *regionSparse, `CoinIndexedVector` *regionSparse2, bool `noPermute=false`) const
This version has same effect as above with FTUpdate==false so number returned is always >=0.
- virtual int `updateTwoColumnsFT` (`CoinIndexedVector` *regionSparse1, `CoinIndexedVector` *regionSparse2, `CoinIndexedVector` *regionSparse3, bool `noPermute=false`)
does FTRAN on two columns
- int `upColumn` (`CoinIndexedVector` *regionSparse, `CoinIndexedVector` *regionSparse2, bool `noPermute=false`, bool `save=false`) const
does updatecolumn if save==true keeps column for replace column
- virtual int `updateColumnTranspose` (`CoinIndexedVector` *regionSparse, `CoinIndexedVector` *regionSparse2) const
Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if regionSparse2 packed on input - will be packed on output.
- int `upColumnTranspose` (`CoinIndexedVector` *regionSparse, `CoinIndexedVector` *regionSparse2) const
does updateColumnTranspose, the other is a wrapper

various uses of factorization

*** *Below this user may not want to know about*

which user may not want to know about (left over from my LP code)

- void `clearArrays` ()
Get rid of all memory.
- int * `indices` () const
Returns array to put basis indices in.
- virtual int * `permute` () const
Returns permute in.

Protected Member Functions

- int `checkPivot` (double `saveFromU`, double `oldPivot`) const

Protected Attributes**data**

- double * `denseVector_`
work array (should be initialized to zero)
- double * `workArea2_`

- work array*
- double * [workArea3_](#)
- work array*
- int * [vecLabels_](#)
- array of labels (should be initialized to zero)*
- int * [indVector_](#)
- array of indices*
- double * [auxVector_](#)
- auxiliary vector*
- int * [auxInd_](#)
- auxiliary vector*
- double * [vecKeep_](#)
- vector to keep for LUupdate*
- int * [indKeep_](#)
- indices of this vector*
- int [keepSize_](#)
- number of nonzeros*
- int * [LrowStarts_](#)
- Starts of the rows of L.*
- int * [LrowLengths_](#)
- Lengths of the rows of L.*
- double * [Lrows_](#)
- L by rows.*
- int * [LrowInd_](#)
- indices in the rows of L*
- int [LrowSize_](#)
- Size of Lrows_.*
- int [LrowCap_](#)
- Capacity of Lrows_.*
- int * [LcolStarts_](#)
- Starts of the columns of L.*
- int * [LcolLengths_](#)
- Lengths of the columns of L.*
- double * [Lcolumns_](#)
- L by columns.*
- int * [LcolInd_](#)
- indices in the columns of L*
- int [LcolSize_](#)
- numbers of elements in L*
- int [LcolCap_](#)
- maximum capacity of L*
- int * [UrowStarts_](#)
- Starts of the rows of U.*
- int * [UrowLengths_](#)
- Lengths of the rows of U.*
- double * [Urows_](#)
- U by rows.*
- int * [UrowInd_](#)
- Indices in the rows of U.*
- int [UrowMaxCap_](#)
- maximum capacity of Urows*
- int [UrowEnd_](#)
- number of used places in Urows*
- int [firstRowInU_](#)
- first row in U*

- int [lastRowInU_](#)
last row in U
- int * [prevRowInU_](#)
previous row in U
- int * [nextRowInU_](#)
next row in U
- int * [UcolStarts_](#)
Starts of the columns of U.
- int * [UcolLengths_](#)
Lengths of the columns of U.
- double * [Ucolumns_](#)
U by columns.
- int * [UcolInd_](#)
Indices in the columns of U.
- int * [prevColInU_](#)
previous column in U
- int * [nextColInU_](#)
next column in U
- int [firstColInU_](#)
first column in U
- int [lastColInU_](#)
last column in U
- int [UcolMaxCap_](#)
maximum capacity of Ucolumns_
- int [UcolEnd_](#)
last used position in Ucolumns_
- int * [colSlack_](#)
indicator of slack variables
- double * [invOfPivots_](#)
inverse values of the elements of diagonal of U
- int * [colOfU_](#)
permutation of columns
- int * [colPosition_](#)
position of column after permutation
- int * [rowOfU_](#)
permutations of rows
- int * [rowPosition_](#)
position of row after permutation
- int * [secRowOfU_](#)
permutations of rows during LUupdate
- int * [secRowPosition_](#)
position of row after permutation during LUupdate
- int * [EtaPosition_](#)
position of Eta vector
- int * [EtaStarts_](#)
Starts of eta vectors.
- int * [EtaLengths_](#)
Lengths of eta vectors.
- int * [EtaInd_](#)
columns of eta vectors
- double * [Eta_](#)
elements of eta vectors
- int [EtaSize_](#)
number of elements in Eta_
- int [lastEtaRow_](#)

- last eta row*
- int [maxEtaRows_](#)
maximum number of eta vectors
- int [EtaMaxCap_](#)
Capacity of Eta_.
- int [minIncrease_](#)
minimum storage increase
- double [updateTol_](#)
maximum size for the diagonal of U after update
- bool [doSuhlHeuristic_](#)
do Suhl heuristic
- double [maxU_](#)
maximum of U
- double [maxGrowth_](#)
bound on the growth rate
- double [maxA_](#)
maximum of A
- int [pivotCandLimit_](#)
maximum number of candidates for pivot
- int [numberSlacks_](#)
number of slacks in basis
- int [firstNumberSlacks_](#)
number of slacks in irst basis

Friends

- [void CoinSimpFactorizationUnitTest](#) (const std::string &mpsDir)

9.77.1 Detailed Description

Definition at line 38 of file CoinSimpFactorization.hpp.

9.77.2 Constructor & Destructor Documentation

9.77.2.1 CoinSimpFactorization::CoinSimpFactorization ()

Default constructor.

9.77.2.2 CoinSimpFactorization::CoinSimpFactorization (const CoinSimpFactorization & other)

Copy constructor.

9.77.2.3 virtual CoinSimpFactorization::~CoinSimpFactorization () [virtual]

Destructor.

9.77.3 Member Function Documentation

9.77.3.1 CoinSimpFactorization& CoinSimpFactorization::operator= (const CoinSimpFactorization & other)

= copy

9.77.3.2 `virtual CoinOtherFactorization* CoinSimpFactorization::clone () const` [virtual]

Clone.

Implements [CoinOtherFactorization](#).

9.77.3.3 `virtual void CoinSimpFactorization::getAreas (int numberRows, int numberColumns, CoinBigIndex maximumL, CoinBigIndex maximumU)` [virtual]

Gets space for a factorization.

Implements [CoinOtherFactorization](#).

9.77.3.4 `virtual void CoinSimpFactorization::preProcess ()` [virtual]

PreProcesses column ordered copy of basis.

Implements [CoinOtherFactorization](#).

9.77.3.5 `virtual int CoinSimpFactorization::factor ()` [virtual]

Does most of factorization returning status 0 - OK -99 - needs more memory -1 - singular - use numberGoodColumns and redo.

Implements [CoinOtherFactorization](#).

9.77.3.6 `virtual void CoinSimpFactorization::postProcess (const int * sequence, int * pivotVariable)` [virtual]

Does post processing on valid factorization - putting variables on correct rows.

Implements [CoinOtherFactorization](#).

9.77.3.7 `virtual void CoinSimpFactorization::makeNonSingular (int * sequence, int numberColumns)` [virtual]

Makes a non-singular basis by replacing variables.

Implements [CoinOtherFactorization](#).

9.77.3.8 `virtual int CoinSimpFactorization::numberElements () const` [inline],[virtual]

Total number of elements in factorization.

Implements [CoinOtherFactorization](#).

Definition at line 83 of file CoinSimpFactorization.hpp.

9.77.3.9 `double CoinSimpFactorization::maximumCoefficient () const`

Returns maximum absolute value in factorization.

9.77.3.10 `virtual int CoinSimpFactorization::replaceColumn (CoinIndexedVector * regionSparse, int pivotRow, double pivotCheck, bool checkBeforeModifying = false, double acceptablePivot = 1.0e-8)` [virtual]

Replaces one Column to basis, returns 0=OK, 1=Probably OK, 2=singular, 3=no room If checkBeforeModifying is true will do all accuracy checks before modifying factorization.

Whether to set this depends on speed considerations. You could just do this on first iteration after factorization and thereafter re-factorize partial update already in U

Implements [CoinOtherFactorization](#).

9.77.3.11 `virtual int CoinSimpFactorization::updateColumnFT (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool noPermute = false) [virtual]`

Updates one column (FTRAN) from regionSparse2 Tries to do FT update number returned is negative if no room region-Sparse starts as zero and is zero at end.

Note - if regionSparse2 packed on input - will be packed on output

Implements [CoinOtherFactorization](#).

9.77.3.12 `virtual int CoinSimpFactorization::updateColumn (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool noPermute = false) const [virtual]`

This version has same effect as above with FTUpdate==false so number returned is always >=0.

Implements [CoinOtherFactorization](#).

9.77.3.13 `virtual int CoinSimpFactorization::updateTwoColumnsFT (CoinIndexedVector * regionSparse1, CoinIndexedVector * regionSparse2, CoinIndexedVector * regionSparse3, bool noPermute = false) [virtual]`

does FTRAN on two columns

Implements [CoinOtherFactorization](#).

9.77.3.14 `int CoinSimpFactorization::upColumn (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2, bool noPermute = false, bool save = false) const`

does updatecolumn if save==true keeps column for replace column

9.77.3.15 `virtual int CoinSimpFactorization::updateColumnTranspose (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2) const [virtual]`

Updates one column (BTRAN) from regionSparse2 regionSparse starts as zero and is zero at end Note - if region-Sparse2 packed on input - will be packed on output.

Implements [CoinOtherFactorization](#).

9.77.3.16 `int CoinSimpFactorization::upColumnTranspose (CoinIndexedVector * regionSparse, CoinIndexedVector * regionSparse2) const`

does updateColumnTranspose, the other is a wrapper

9.77.3.17 `void CoinSimpFactorization::clearArrays () [inline],[virtual]`

Get rid of all memory.

Reimplemented from [CoinOtherFactorization](#).

Definition at line 151 of file CoinSimpFactorization.hpp.

9.77.3.18 `int* CoinSimpFactorization::indices () const [inline],[virtual]`

Returns array to put basis indices in.

Implements [CoinOtherFactorization](#).

Definition at line 154 of file CoinSimpFactorization.hpp.

9.77.3.19 `virtual int* CoinSimpFactorization::permute () const` `[inline],[virtual]`

Returns permute in.

Implements [CoinOtherFactorization](#).

Definition at line 157 of file CoinSimpFactorization.hpp.

9.77.3.20 `void CoinSimpFactorization::gutsOfDestructor ()`

The real work of destructor.

9.77.3.21 `void CoinSimpFactorization::gutsOfInitialize ()`

The real work of constructor.

9.77.3.22 `void CoinSimpFactorization::gutsOfCopy (const CoinSimpFactorization & other)`

The real work of copy.

9.77.3.23 `void CoinSimpFactorization::factorize (int numberOfRows, int numberOfColumns, const int colStarts[], const int indicesRow[], const double elements[])`

calls factorization

9.77.3.24 `int CoinSimpFactorization::mainLoopFactor (FactorPointers & pointers)`

main loop of factorization

9.77.3.25 `void CoinSimpFactorization::copyLbyRows ()`

copies L by rows

9.77.3.26 `void CoinSimpFactorization::copyUbyColumns ()`

copies U by columns

9.77.3.27 `int CoinSimpFactorization::findPivot (FactorPointers & pointers, int & r, int & s, bool & ifSlack)`

finds a pivot element using Markowitz count

9.77.3.28 `int CoinSimpFactorization::findPivotShCol (FactorPointers & pointers, int & r, int & s)`

finds a pivot in a shortest column

9.77.3.29 `int CoinSimpFactorization::findPivotSimp (FactorPointers & pointers, int & r, int & s)`

finds a pivot in the first column available

9.77.3.30 `void CoinSimpFactorization::GaussEliminate (FactorPointers & pointers, int & r, int & s)`

does Gauss elimination

9.77.3.31 `int CoinSimpFactorization::findShortRow (const int column, const int length, int & minRow, int & minRowLength, FactorPointers & pointers)`

finds short row that intersects a given column

9.77.3.32 **int** CoinSimpFactorization::findShortColumn (**const** **int** *row*, **const** **int** *length*, **int** & *minCol*, **int** & *minColLength*, **FactorPointers** & *pointers*)

finds short column that intersects a given row

9.77.3.33 **double** CoinSimpFactorization::findMaxInRow (**const** **int** *row*, **FactorPointers** & *pointers*)

finds maximum absolute value in a row

9.77.3.34 **void** CoinSimpFactorization::pivoting (**const** **int** *pivotRow*, **const** **int** *pivotColumn*, **const** **double** *invPivot*, **FactorPointers** & *pointers*)

does pivoting

9.77.3.35 **void** CoinSimpFactorization::updateCurrentRow (**const** **int** *pivotRow*, **const** **int** *row*, **const** **double** *multiplier*, **FactorPointers** & *pointers*, **int** & *newNonZeros*)

part of pivoting

9.77.3.36 **void** CoinSimpFactorization::increaseLsize ()

allocates more space for L

9.77.3.37 **void** CoinSimpFactorization::increaseRowSize (**const** **int** *row*, **const** **int** *newSize*)

allocates more space for a row of U

9.77.3.38 **void** CoinSimpFactorization::increaseColSize (**const** **int** *column*, **const** **int** *newSize*, **const** **bool** *b*)

allocates more space for a column of U

9.77.3.39 **void** CoinSimpFactorization::enlargeUrow (**const** **int** *numNewElements*)

allocates more space for rows of U

9.77.3.40 **void** CoinSimpFactorization::enlargeUcol (**const** **int** *numNewElements*, **const** **bool** *b*)

allocates more space for columns of U

9.77.3.41 **int** CoinSimpFactorization::findInRow (**const** **int** *row*, **const** **int** *column*)

finds a given row in a column

9.77.3.42 **int** CoinSimpFactorization::findInColumn (**const** **int** *column*, **const** **int** *row*)

finds a given column in a row

9.77.3.43 **void** CoinSimpFactorization::removeRowFromActSet (**const** **int** *row*, **FactorPointers** & *pointers*)

declares a row inactive

9.77.3.44 **void** CoinSimpFactorization::removeColumnFromActSet (**const** **int** *column*, **FactorPointers** & *pointers*)

declares a column inactive

9.77.3.45 **void** CoinSimpFactorization::allocateSpaceForU ()

allocates space for U

9.77.3.46 **void** CoinSimpFactorization::allocateSomeArrays ()

allocates several working arrays

9.77.3.47 **void** CoinSimpFactorization::initialSomeNumbers ()

initializes some numbers

9.77.3.48 **void** CoinSimpFactorization::Lxeqb (*double * b*) **const**

solves $Lx = b$

9.77.3.49 **void** CoinSimpFactorization::Lxeqb2 (*double * b1*, *double * b2*) **const**

same as above but with two rhs

9.77.3.50 **void** CoinSimpFactorization::Uxeqb (*double * b*, *double * sol*) **const**

solves $Ux = b$

9.77.3.51 **void** CoinSimpFactorization::Uxeqb2 (*double * b1*, *double * sol1*, *double * sol2*, *double * b2*) **const**

same as above but with two rhs

9.77.3.52 **void** CoinSimpFactorization::xLeqb (*double * b*) **const**

solves $xL = b$

9.77.3.53 **void** CoinSimpFactorization::xUeqb (*double * b*, *double * sol*) **const**

solves $xU = b$

9.77.3.54 **int** CoinSimpFactorization::LUupdate (*int newBasicCol*)

updates factorization after a Simplex iteration

9.77.3.55 **void** CoinSimpFactorization::newEta (*int row*, *int numNewElements*)

creates a new eta vector

9.77.3.56 **void** CoinSimpFactorization::copyRowPermutations ()

makes a copy of row permutations

9.77.3.57 **void** CoinSimpFactorization::Hxeqb (*double * b*) **const**

solves $Hx = b$, where H is a product of eta matrices

9.77.3.58 **void** CoinSimpFactorization::Hxeqb2 (*double * b1*, *double * b2*) **const**

same as above but with two rhs

9.77.3.59 **void** CoinSimpFactorization::xHeqb (*double * b*) **const**

solves $xH = b$

9.77.3.60 `void CoinSimpFactorization::ftran (double * b, double * sol, bool save) const`

does FTRAN

9.77.3.61 `void CoinSimpFactorization::ftran2 (double * b1, double * sol1, double * b2, double * sol2) const`

same as above but with two columns

9.77.3.62 `void CoinSimpFactorization::btran (double * b, double * sol) const`

does BTRAN

9.77.3.63 `int CoinSimpFactorization::checkPivot (double saveFromU, double oldPivot) const` [protected]

Returns accuracy status of replaceColumn returns 0=OK, 1=Probably OK, 2=singular

9.77.4 Friends And Related Function Documentation

9.77.4.1 `void CoinSimpFactorizationUnitTest (const std::string & mpsDir)` [friend]

9.77.5 Member Data Documentation

9.77.5.1 `double* CoinSimpFactorization::denseVector_` [protected]

work array (should be initialized to zero)

Definition at line 273 of file CoinSimpFactorization.hpp.

9.77.5.2 `double* CoinSimpFactorization::workArea2_` [protected]

work array

Definition at line 275 of file CoinSimpFactorization.hpp.

9.77.5.3 `double* CoinSimpFactorization::workArea3_` [protected]

work array

Definition at line 277 of file CoinSimpFactorization.hpp.

9.77.5.4 `int* CoinSimpFactorization::vecLabels_` [protected]

array of labels (should be initialized to zero)

Definition at line 279 of file CoinSimpFactorization.hpp.

9.77.5.5 `int* CoinSimpFactorization::indVector_` [protected]

array of indices

Definition at line 281 of file CoinSimpFactorization.hpp.

9.77.5.6 `double* CoinSimpFactorization::auxVector_` [protected]

auxiliary vector

Definition at line 284 of file CoinSimpFactorization.hpp.

9.77.5.7 `int* CoinSimpFactorization::auxInd_` `[protected]`

auxiliary vector

Definition at line 286 of file CoinSimpFactorization.hpp.

9.77.5.8 `double* CoinSimpFactorization::vecKeep_` `[protected]`

vector to keep for LUupdate

Definition at line 289 of file CoinSimpFactorization.hpp.

9.77.5.9 `int* CoinSimpFactorization::indKeep_` `[protected]`

indices of this vector

Definition at line 291 of file CoinSimpFactorization.hpp.

9.77.5.10 `int CoinSimpFactorization::keepSize_` `[mutable], [protected]`

number of nonzeros

Definition at line 293 of file CoinSimpFactorization.hpp.

9.77.5.11 `int* CoinSimpFactorization::LrowStarts_` `[protected]`

Starts of the rows of L.

Definition at line 298 of file CoinSimpFactorization.hpp.

9.77.5.12 `int* CoinSimpFactorization::LrowLengths_` `[protected]`

Lengths of the rows of L.

Definition at line 300 of file CoinSimpFactorization.hpp.

9.77.5.13 `double* CoinSimpFactorization::Lrows_` `[protected]`

L by rows.

Definition at line 302 of file CoinSimpFactorization.hpp.

9.77.5.14 `int* CoinSimpFactorization::LrowInd_` `[protected]`

indices in the rows of L

Definition at line 304 of file CoinSimpFactorization.hpp.

9.77.5.15 `int CoinSimpFactorization::LrowSize_` `[protected]`

Size of Lrows_.

Definition at line 306 of file CoinSimpFactorization.hpp.

9.77.5.16 `int CoinSimpFactorization::LrowCap_` `[protected]`

Capacity of Lrows_.

Definition at line 308 of file CoinSimpFactorization.hpp.

9.77.5.17 int* CoinSimpFactorization::LcolStarts_ [protected]

Starts of the columns of L.

Definition at line 311 of file CoinSimpFactorization.hpp.

9.77.5.18 int* CoinSimpFactorization::LcolLengths_ [protected]

Lengths of the columns of L.

Definition at line 313 of file CoinSimpFactorization.hpp.

9.77.5.19 double* CoinSimpFactorization::Lcolumns_ [protected]

L by columns.

Definition at line 315 of file CoinSimpFactorization.hpp.

9.77.5.20 int* CoinSimpFactorization::LcolInd_ [protected]

indices in the columns of L

Definition at line 317 of file CoinSimpFactorization.hpp.

9.77.5.21 int CoinSimpFactorization::LcolSize_ [protected]

numbers of elements in L

Definition at line 319 of file CoinSimpFactorization.hpp.

9.77.5.22 int CoinSimpFactorization::LcolCap_ [protected]

maximum capacity of L

Definition at line 321 of file CoinSimpFactorization.hpp.

9.77.5.23 int* CoinSimpFactorization::UrowStarts_ [protected]

Starts of the rows of U.

Definition at line 325 of file CoinSimpFactorization.hpp.

9.77.5.24 int* CoinSimpFactorization::UrowLengths_ [protected]

Lengths of the rows of U.

Definition at line 327 of file CoinSimpFactorization.hpp.

9.77.5.25 double* CoinSimpFactorization::Urows_ [protected]

U by rows.

Definition at line 333 of file CoinSimpFactorization.hpp.

9.77.5.26 int* CoinSimpFactorization::UrowInd_ [protected]

Indices in the rows of U.

Definition at line 335 of file CoinSimpFactorization.hpp.

9.77.5.27 `int CoinSimpFactorization::UrowMaxCap_` [protected]

maximum capacity of Urows

Definition at line 337 of file CoinSimpFactorization.hpp.

9.77.5.28 `int CoinSimpFactorization::UrowEnd_` [protected]

number of used places in Urows

Definition at line 339 of file CoinSimpFactorization.hpp.

9.77.5.29 `int CoinSimpFactorization::firstRowInU_` [protected]

first row in U

Definition at line 341 of file CoinSimpFactorization.hpp.

9.77.5.30 `int CoinSimpFactorization::lastRowInU_` [protected]

last row in U

Definition at line 343 of file CoinSimpFactorization.hpp.

9.77.5.31 `int* CoinSimpFactorization::prevRowInU_` [protected]

previous row in U

Definition at line 345 of file CoinSimpFactorization.hpp.

9.77.5.32 `int* CoinSimpFactorization::nextRowInU_` [protected]

next row in U

Definition at line 347 of file CoinSimpFactorization.hpp.

9.77.5.33 `int* CoinSimpFactorization::UcolStarts_` [protected]

Starts of the columns of U.

Definition at line 350 of file CoinSimpFactorization.hpp.

9.77.5.34 `int* CoinSimpFactorization::UcolLengths_` [protected]

Lengths of the columns of U.

Definition at line 352 of file CoinSimpFactorization.hpp.

9.77.5.35 `double* CoinSimpFactorization::Ucolumns_` [protected]

U by columns.

Definition at line 358 of file CoinSimpFactorization.hpp.

9.77.5.36 `int* CoinSimpFactorization::UcolInd_` [protected]

Indices in the columns of U.

Definition at line 360 of file CoinSimpFactorization.hpp.

9.77.5.37 `int* CoinSimpFactorization::prevCollnU_` [protected]

previous column in U

Definition at line 362 of file CoinSimpFactorization.hpp.

9.77.5.38 `int* CoinSimpFactorization::nextCollnU_` [protected]

next column in U

Definition at line 364 of file CoinSimpFactorization.hpp.

9.77.5.39 `int CoinSimpFactorization::firstCollnU_` [protected]

first column in U

Definition at line 366 of file CoinSimpFactorization.hpp.

9.77.5.40 `int CoinSimpFactorization::lastCollnU_` [protected]

last column in U

Definition at line 368 of file CoinSimpFactorization.hpp.

9.77.5.41 `int CoinSimpFactorization::UcolMaxCap_` [protected]

maximum capacity of Ucolumns_

Definition at line 370 of file CoinSimpFactorization.hpp.

9.77.5.42 `int CoinSimpFactorization::UcolEnd_` [protected]

last used position in Ucolumns_

Definition at line 372 of file CoinSimpFactorization.hpp.

9.77.5.43 `int* CoinSimpFactorization::colSlack_` [protected]

indicator of slack variables

Definition at line 374 of file CoinSimpFactorization.hpp.

9.77.5.44 `double* CoinSimpFactorization::invOfPivots_` [protected]

inverse values of the elements of diagonal of U

Definition at line 377 of file CoinSimpFactorization.hpp.

9.77.5.45 `int* CoinSimpFactorization::colOfU_` [protected]

permutation of columns

Definition at line 380 of file CoinSimpFactorization.hpp.

9.77.5.46 `int* CoinSimpFactorization::colPosition_` [protected]

position of column after permutation

Definition at line 382 of file CoinSimpFactorization.hpp.

9.77.5.47 `int* CoinSimpFactorization::rowOfU_` [protected]

permutations of rows

Definition at line 384 of file CoinSimpFactorization.hpp.

9.77.5.48 `int* CoinSimpFactorization::rowPosition_` [protected]

position of row after permutation

Definition at line 386 of file CoinSimpFactorization.hpp.

9.77.5.49 `int* CoinSimpFactorization::secRowOfU_` [protected]

permutations of rows during LUupdate

Definition at line 388 of file CoinSimpFactorization.hpp.

9.77.5.50 `int* CoinSimpFactorization::secRowPosition_` [protected]

position of row after permutation during LUupdate

Definition at line 390 of file CoinSimpFactorization.hpp.

9.77.5.51 `int* CoinSimpFactorization::EtaPosition_` [protected]

position of Eta vector

Definition at line 393 of file CoinSimpFactorization.hpp.

9.77.5.52 `int* CoinSimpFactorization::EtaStarts_` [protected]

Starts of eta vectors.

Definition at line 395 of file CoinSimpFactorization.hpp.

9.77.5.53 `int* CoinSimpFactorization::EtaLengths_` [protected]

Lengths of eta vectors.

Definition at line 397 of file CoinSimpFactorization.hpp.

9.77.5.54 `int* CoinSimpFactorization::EtaInd_` [protected]

columns of eta vectors

Definition at line 399 of file CoinSimpFactorization.hpp.

9.77.5.55 `double* CoinSimpFactorization::Eta_` [protected]

elements of eta vectors

Definition at line 401 of file CoinSimpFactorization.hpp.

9.77.5.56 `int CoinSimpFactorization::EtaSize_` [protected]

number of elements in Eta_

Definition at line 403 of file CoinSimpFactorization.hpp.

9.77.5.57 `int CoinSimpFactorization::lastEtaRow_` `[protected]`

last eta row

Definition at line 405 of file CoinSimpFactorization.hpp.

9.77.5.58 `int CoinSimpFactorization::maxEtaRows_` `[protected]`

maximum number of eta vectors

Definition at line 407 of file CoinSimpFactorization.hpp.

9.77.5.59 `int CoinSimpFactorization::EtaMaxCap_` `[protected]`

Capacity of Eta_.

Definition at line 409 of file CoinSimpFactorization.hpp.

9.77.5.60 `int CoinSimpFactorization::minIncrease_` `[protected]`

minimum storage increase

Definition at line 412 of file CoinSimpFactorization.hpp.

9.77.5.61 `double CoinSimpFactorization::updateTol_` `[protected]`

maximum size for the diagonal of U after update

Definition at line 414 of file CoinSimpFactorization.hpp.

9.77.5.62 `bool CoinSimpFactorization::doSuhlHeuristic_` `[protected]`

do Shul heuristic

Definition at line 416 of file CoinSimpFactorization.hpp.

9.77.5.63 `double CoinSimpFactorization::maxU_` `[protected]`

maximum of U

Definition at line 418 of file CoinSimpFactorization.hpp.

9.77.5.64 `double CoinSimpFactorization::maxGrowth_` `[protected]`

bound on the growth rate

Definition at line 420 of file CoinSimpFactorization.hpp.

9.77.5.65 `double CoinSimpFactorization::maxA_` `[protected]`

maximum of A

Definition at line 422 of file CoinSimpFactorization.hpp.

9.77.5.66 `int CoinSimpFactorization::pivotCandLimit_` `[protected]`

maximum number of candidates for pivot

Definition at line 424 of file CoinSimpFactorization.hpp.

9.77.5.67 int CoinSimpFactorization::numberSlacks_ [protected]

number of slacks in basis

Definition at line 426 of file CoinSimpFactorization.hpp.

9.77.5.68 int CoinSimpFactorization::firstNumberSlacks_ [protected]

number of slacks in first basis

Definition at line 428 of file CoinSimpFactorization.hpp.

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

- /home/ted/COIN/trunk/CoinUtils/src/CoinSimpFactorization.hpp

9.78 CoinSnapshot Class Reference

NON Abstract Base Class for interfacing with cut generators or branching code or .

```
#include <CoinSnapshot.hpp>
```

Public Member Functions

Problem query methods

The Matrix pointers may be NULL

- int [getNumCols](#) () const
Get number of columns.
- int [getNumRows](#) () const
Get number of rows.
- int [getNumElements](#) () const
Get number of nonzero elements.
- int [getNumIntegers](#) () const
Get number of integer variables.
- const double * [getColLower](#) () const
Get pointer to array[getNumCols()] of column lower bounds.
- const double * [getColUpper](#) () const
Get pointer to array[getNumCols()] of column upper bounds.
- const double * [getRowLower](#) () const
Get pointer to array[getNumRows()] of row lower bounds.
- const double * [getRowUpper](#) () const
Get pointer to array[getNumRows()] of row upper bounds.
- const double * [getRightHandSide](#) () const
Get pointer to array[getNumRows()] of row right-hand sides This gives same results as OsiSolverInterface for useful cases If [getRowUpper](#)()[i] != infinity then [getRightHandSide](#)()[i] == [getRowUpper](#)()[i] else [getRightHandSide](#)()[i] == [getRowLower](#)()[i].
- const double * [getObjCoefficients](#) () const
Get pointer to array[getNumCols()] of objective function coefficients.
- double [getObjSense](#) () const
Get objective function sense (1 for min (default), -1 for max)
- bool [isContinuous](#) (int colIndex) const
Return true if variable is continuous.
- bool [isBinary](#) (int colIndex) const
Return true if variable is binary.

- bool `isInteger` (int collIndex) const
Return true if column is integer.
- bool `isIntegerNonBinary` (int collIndex) const
Return true if variable is general integer.
- bool `isFreeBinary` (int collIndex) const
Return true if variable is binary and not fixed at either bound.
- const char * `getColType` () const
Get colType array ('B', 'I', or 'C' for Binary, Integer and Continuous)
- const `CoinPackedMatrix` * `getMatrixByRow` () const
Get pointer to row-wise copy of current matrix.
- const `CoinPackedMatrix` * `getMatrixByCol` () const
Get pointer to column-wise copy of current matrix.
- const `CoinPackedMatrix` * `getOriginalMatrixByRow` () const
Get pointer to row-wise copy of "original" matrix.
- const `CoinPackedMatrix` * `getOriginalMatrixByCol` () const
Get pointer to column-wise copy of "original" matrix.

Solution query methods

- const double * `getColSolution` () const
Get pointer to array[getNumCols()] of primal variable values.
- const double * `getRowPrice` () const
Get pointer to array[getNumRows()] of dual variable values.
- const double * `getReducedCost` () const
Get a pointer to array[getNumCols()] of reduced costs.
- const double * `getRowActivity` () const
Get pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).
- const double * `getDoNotSeparateThis` () const
Get pointer to array[getNumCols()] of primal variable values which should not be separated (for debug)

Other scalar get methods

- double `getInfinity` () const
Get solver's value for infinity.
- double `getObjValue` () const
Get objective function value - including any offset i.e.
- double `getObjOffset` () const
Get objective offset i.e. $\sum c_j x_j - \text{objValue} = \text{objOffset}$.
- double `getDualTolerance` () const
Get dual tolerance.
- double `getPrimalTolerance` () const
Get primal tolerance.
- double `getIntegerTolerance` () const
Get integer tolerance.
- double `getIntegerUpperBound` () const
*Get integer upper bound i.e. best solution * getObjSense.*
- double `getIntegerLowerBound` () const
*Get integer lower bound i.e. best possible solution * getObjSense.*

Method to input a problem

- void `loadProblem` (const `CoinPackedMatrix` &matrix, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub, bool makeRowCopy=false)
Load in a problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).

Methods to set data

- **void setNumCols** (int value)
Set number of columns.
- **void setNumRows** (int value)
Set number of rows.
- **void setNumElements** (int value)
Set number of nonzero elements.
- **void setNumIntegers** (int value)
Set number of integer variables.
- **void setColLower** (const double *array, bool copyIn=true)
Set pointer to array[getNumCols()] of column lower bounds.
- **void setColUpper** (const double *array, bool copyIn=true)
Set pointer to array[getNumCols()] of column upper bounds.
- **void setRowLower** (const double *array, bool copyIn=true)
Set pointer to array[getNumRows()] of row lower bounds.
- **void setRowUpper** (const double *array, bool copyIn=true)
Set pointer to array[getNumRows()] of row upper bounds.
- **void setRightHandSide** (const double *array, bool copyIn=true)
Set pointer to array[getNumRows()] of row right-hand sides This gives same results as OsiSolverInterface for useful cases If `getRowUpper()[i] != infinity` then `getRightHandSide()[i] == getRowUpper()[i]` else `getRightHandSide()[i] == getRowLower()[i]`.
- **void createRightHandSide** ()
Create array[getNumRows()] of row right-hand sides using existing information This gives same results as OsiSolverInterface for useful cases If `getRowUpper()[i] != infinity` then `getRightHandSide()[i] == getRowUpper()[i]` else `getRightHandSide()[i] == getRowLower()[i]`.
- **void setObjCoefficients** (const double *array, bool copyIn=true)
Set pointer to array[getNumCols()] of objective function coefficients.
- **void setObjSense** (double value)
Set objective function sense (1 for min (default), -1 for max)
- **void setColType** (const char *array, bool copyIn=true)
Set colType array ('B', 'I', or 'C' for Binary, Integer and Continuous)
- **void setMatrixByRow** (const CoinPackedMatrix *matrix, bool copyIn=true)
Set pointer to row-wise copy of current matrix.
- **void createMatrixByRow** ()
Create row-wise copy from MatrixByCol.
- **void setMatrixByCol** (const CoinPackedMatrix *matrix, bool copyIn=true)
Set pointer to column-wise copy of current matrix.
- **void setOriginalMatrixByRow** (const CoinPackedMatrix *matrix, bool copyIn=true)
Set pointer to row-wise copy of "original" matrix.
- **void setOriginalMatrixByCol** (const CoinPackedMatrix *matrix, bool copyIn=true)
Set pointer to column-wise copy of "original" matrix.
- **void setColSolution** (const double *array, bool copyIn=true)
Set pointer to array[getNumCols()] of primal variable values.
- **void setRowPrice** (const double *array, bool copyIn=true)
Set pointer to array[getNumRows()] of dual variable values.
- **void setReducedCost** (const double *array, bool copyIn=true)
Set a pointer to array[getNumCols()] of reduced costs.
- **void setRowActivity** (const double *array, bool copyIn=true)
Set pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).
- **void setDoNotSeparateThis** (const double *array, bool copyIn=true)
Set pointer to array[getNumCols()] of primal variable values which should not be separated (for debug)
- **void setInfinity** (double value)
Set solver's value for infinity.
- **void setObjValue** (double value)

- *Set objective function value (including any rhs offset)*
- `void setObjOffset` (double value)
*Set objective offset i.e. $\sum c_{sub\ j} * x_{sub\ j} - objValue = objOffset$.*
- `void setDualTolerance` (double value)
Set dual tolerance.
- `void setPrimalTolerance` (double value)
Set primal tolerance.
- `void setIntegerTolerance` (double value)
Set integer tolerance.
- `void setIntegerUpperBound` (double value)
*Set integer upper bound i.e. $best\ solution * getObjSense$.*
- `void setIntegerLowerBound` (double value)
*Set integer lower bound i.e. $best\ possible\ solution * getObjSense$.*

Constructors and destructors

- `CoinSnapshot` ()
Default Constructor.
- `CoinSnapshot` (const `CoinSnapshot` &)
Copy constructor.
- `CoinSnapshot` & `operator=` (const `CoinSnapshot` &rhs)
Assignment operator.
- `virtual ~CoinSnapshot` ()
Destructor.

9.78.1 Detailed Description

NON Abstract Base Class for interfacing with cut generators or branching code or .

It is designed to be snapshot of a problem at a node in tree

The class may or may not own the arrays - see `owned_`

Querying a problem that has no data associated with it will result in zeros for the number of rows and columns, and NULL pointers from the methods that return arrays.

Definition at line 25 of file `CoinSnapshot.hpp`.

9.78.2 Constructor & Destructor Documentation

9.78.2.1 `CoinSnapshot::CoinSnapshot ()`

Default Constructor.

9.78.2.2 `CoinSnapshot::CoinSnapshot (const CoinSnapshot &)`

Copy constructor.

9.78.2.3 `virtual CoinSnapshot::~CoinSnapshot ()` `[virtual]`

Destructor.

9.78.3 Member Function Documentation

9.78.3.1 `int CoinSnapshot::getNumCols () const [inline]`

Get number of columns.

Definition at line 36 of file CoinSnapshot.hpp.

9.78.3.2 `int CoinSnapshot::getNumRows () const [inline]`

Get number of rows.

Definition at line 40 of file CoinSnapshot.hpp.

9.78.3.3 `int CoinSnapshot::getNumElements () const [inline]`

Get number of nonzero elements.

Definition at line 44 of file CoinSnapshot.hpp.

9.78.3.4 `int CoinSnapshot::getNumIntegers () const [inline]`

Get number of integer variables.

Definition at line 48 of file CoinSnapshot.hpp.

9.78.3.5 `const double* CoinSnapshot::getColLower () const [inline]`

Get pointer to array[getNumCols()] of column lower bounds.

Definition at line 52 of file CoinSnapshot.hpp.

9.78.3.6 `const double* CoinSnapshot::getColUpper () const [inline]`

Get pointer to array[getNumCols()] of column upper bounds.

Definition at line 56 of file CoinSnapshot.hpp.

9.78.3.7 `const double* CoinSnapshot::getRowLower () const [inline]`

Get pointer to array[getNumRows()] of row lower bounds.

Definition at line 60 of file CoinSnapshot.hpp.

9.78.3.8 `const double* CoinSnapshot::getRowUpper () const [inline]`

Get pointer to array[getNumRows()] of row upper bounds.

Definition at line 64 of file CoinSnapshot.hpp.

9.78.3.9 `const double* CoinSnapshot::getRightHandSide () const [inline]`

Get pointer to array[getNumRows()] of row right-hand sides This gives same results as OsiSolverInterface for useful cases If `getRowUpper()[i]` != infinity then `getRightHandSide()[i] == getRowUpper()[i]` else `getRightHandSide()[i] == getRowLower()[i]`.

Definition at line 74 of file CoinSnapshot.hpp.

9.78.3.10 `const double* CoinSnapshot::getObjCoefficients () const [inline]`

Get pointer to array[getNumCols()] of objective function coefficients.

Definition at line 78 of file CoinSnapshot.hpp.

9.78.3.11 `double CoinSnapshot::getObjSense () const [inline]`

Get objective function sense (1 for min (default), -1 for max)

Definition at line 82 of file CoinSnapshot.hpp.

9.78.3.12 `bool CoinSnapshot::isContinuous (int colIndex) const [inline]`

Return true if variable is continuous.

Definition at line 86 of file CoinSnapshot.hpp.

9.78.3.13 `bool CoinSnapshot::isBinary (int colIndex) const [inline]`

Return true if variable is binary.

Definition at line 90 of file CoinSnapshot.hpp.

9.78.3.14 `bool CoinSnapshot::isInteger (int colIndex) const [inline]`

Return true if column is integer.

Definition at line 94 of file CoinSnapshot.hpp.

9.78.3.15 `bool CoinSnapshot::isIntegerNonBinary (int colIndex) const [inline]`

Return true if variable is general integer.

Definition at line 98 of file CoinSnapshot.hpp.

9.78.3.16 `bool CoinSnapshot::isFreeBinary (int colIndex) const [inline]`

Return true if variable is binary and not fixed at either bound.

Definition at line 102 of file CoinSnapshot.hpp.

9.78.3.17 `const char* CoinSnapshot::getColType () const [inline]`

Get colType array ('B', 'I', or 'C' for Binary, Integer and Continuous)

Definition at line 106 of file CoinSnapshot.hpp.

9.78.3.18 `const CoinPackedMatrix* CoinSnapshot::getMatrixByRow () const [inline]`

Get pointer to row-wise copy of current matrix.

Definition at line 110 of file CoinSnapshot.hpp.

9.78.3.19 `const CoinPackedMatrix* CoinSnapshot::getMatrixByCol () const [inline]`

Get pointer to column-wise copy of current matrix.

Definition at line 114 of file CoinSnapshot.hpp.

9.78.3.20 `const CoinPackedMatrix* CoinSnapshot::getOriginalMatrixByRow () const [inline]`

Get pointer to row-wise copy of "original" matrix.

Definition at line 118 of file CoinSnapshot.hpp.

9.78.3.21 `const CoinPackedMatrix* CoinSnapshot::getOriginalMatrixByCol () const` `[inline]`

Get pointer to column-wise copy of "original" matrix.

Definition at line 122 of file CoinSnapshot.hpp.

9.78.3.22 `const double* CoinSnapshot::getColSolution () const` `[inline]`

Get pointer to array[[getNumCols\(\)](#)] of primal variable values.

Definition at line 129 of file CoinSnapshot.hpp.

9.78.3.23 `const double* CoinSnapshot::getRowPrice () const` `[inline]`

Get pointer to array[[getNumRows\(\)](#)] of dual variable values.

Definition at line 133 of file CoinSnapshot.hpp.

9.78.3.24 `const double* CoinSnapshot::getReducedCost () const` `[inline]`

Get a pointer to array[[getNumCols\(\)](#)] of reduced costs.

Definition at line 137 of file CoinSnapshot.hpp.

9.78.3.25 `const double* CoinSnapshot::getRowActivity () const` `[inline]`

Get pointer to array[[getNumRows\(\)](#)] of row activity levels (constraint matrix times the solution vector).

Definition at line 141 of file CoinSnapshot.hpp.

9.78.3.26 `const double* CoinSnapshot::getDoNotSeparateThis () const` `[inline]`

Get pointer to array[[getNumCols\(\)](#)] of primal variable values which should not be separated (for debug)

Definition at line 145 of file CoinSnapshot.hpp.

9.78.3.27 `double CoinSnapshot::getInfinity () const` `[inline]`

Get solver's value for infinity.

Definition at line 152 of file CoinSnapshot.hpp.

9.78.3.28 `double CoinSnapshot::getObjValue () const` `[inline]`

Get objective function value - including any offset i.e.

$\sum c_{\text{sub } j} * x_{\text{sub } j} - \text{objValue} = \text{objOffset}$

Definition at line 157 of file CoinSnapshot.hpp.

9.78.3.29 `double CoinSnapshot::getObjOffset () const` `[inline]`

Get objective offset i.e. $\sum c_{\text{sub } j} * x_{\text{sub } j} - \text{objValue} = \text{objOffset}$.

Definition at line 161 of file CoinSnapshot.hpp.

9.78.3.30 `double CoinSnapshot::getDualTolerance () const` `[inline]`

Get dual tolerance.

Definition at line 165 of file CoinSnapshot.hpp.

9.78.3.31 `double CoinSnapshot::getPrimalTolerance () const [inline]`

Get primal tolerance.

Definition at line 169 of file CoinSnapshot.hpp.

9.78.3.32 `double CoinSnapshot::getIntegerTolerance () const [inline]`

Get integer tolerance.

Definition at line 173 of file CoinSnapshot.hpp.

9.78.3.33 `double CoinSnapshot::getIntegerUpperBound () const [inline]`

Get integer upper bound i.e. best solution * getObjSense.

Definition at line 177 of file CoinSnapshot.hpp.

9.78.3.34 `double CoinSnapshot::getIntegerLowerBound () const [inline]`

Get integer lower bound i.e. best possible solution * getObjSense.

Definition at line 181 of file CoinSnapshot.hpp.

9.78.3.35 `void CoinSnapshot::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub, bool makeRowCopy = false)`

Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).

If a pointer is NULL then the following values are the default:

- `colub`: all columns have upper bound infinity
- `collb`: all columns have lower bound 0
- `rowub`: all rows have upper bound infinity
- `rowlb`: all rows have lower bound -infinity
- `obj`: all variables have 0 objective coefficient

All solution type arrays will be deleted

9.78.3.36 `void CoinSnapshot::setNumCols (int value) [inline]`

Set number of columns.

Definition at line 214 of file CoinSnapshot.hpp.

9.78.3.37 `void CoinSnapshot::setNumRows (int value) [inline]`

Set number of rows.

Definition at line 218 of file CoinSnapshot.hpp.

9.78.3.38 `void CoinSnapshot::setNumElements (int value) [inline]`

Set number of nonzero elements.

Definition at line 222 of file CoinSnapshot.hpp.

9.78.3.39 `void CoinSnapshot::setNumIntegers (int value) [inline]`

Set number of integer variables.

Definition at line 226 of file CoinSnapshot.hpp.

9.78.3.40 `void CoinSnapshot::setColLower (const double * array, bool copyIn = true)`

Set pointer to array[getNumCols()] of column lower bounds.

9.78.3.41 `void CoinSnapshot::setColUpper (const double * array, bool copyIn = true)`

Set pointer to array[getNumCols()] of column upper bounds.

9.78.3.42 `void CoinSnapshot::setRowLower (const double * array, bool copyIn = true)`

Set pointer to array[getNumRows()] of row lower bounds.

9.78.3.43 `void CoinSnapshot::setRowUpper (const double * array, bool copyIn = true)`

Set pointer to array[getNumRows()] of row upper bounds.

9.78.3.44 `void CoinSnapshot::setRightHandSide (const double * array, bool copyIn = true)`

Set pointer to array[getNumRows()] of row right-hand sides This gives same results as OsiSolverInterface for useful cases If `getRowUpper()[i] != infinity` then `getRightHandSide()[i] == getRowUpper()[i]` else `getRightHandSide()[i] == getRowLower()[i]`.

9.78.3.45 `void CoinSnapshot::createRightHandSide ()`

Create array[getNumRows()] of row right-hand sides using existing information This gives same results as OsiSolverInterface for useful cases If `getRowUpper()[i] != infinity` then `getRightHandSide()[i] == getRowUpper()[i]` else `getRightHandSide()[i] == getRowLower()[i]`.

9.78.3.46 `void CoinSnapshot::setObjCoefficients (const double * array, bool copyIn = true)`

Set pointer to array[getNumCols()] of objective function coefficients.

9.78.3.47 `void CoinSnapshot::setObjSense (double value) [inline]`

Set objective function sense (1 for min (default), -1 for max)

Definition at line 264 of file CoinSnapshot.hpp.

9.78.3.48 `void CoinSnapshot::setColType (const char * array, bool copyIn = true)`

Set colType array ('B', 'I', or 'C' for Binary, Integer and Continuous)

9.78.3.49 `void CoinSnapshot::setMatrixByRow (const CoinPackedMatrix * matrix, bool copyIn = true)`

Set pointer to row-wise copy of current matrix.

9.78.3.50 `void CoinSnapshot::createMatrixByRow ()`

Create row-wise copy from MatrixByCol.

9.78.3.51 **void** CoinSnapshot::setMatrixByCol (**const** CoinPackedMatrix * *matrix*, **bool** *copyIn* = **true**)

Set pointer to column-wise copy of current matrix.

9.78.3.52 **void** CoinSnapshot::setOriginalMatrixByRow (**const** CoinPackedMatrix * *matrix*, **bool** *copyIn* = **true**)

Set pointer to row-wise copy of "original" matrix.

9.78.3.53 **void** CoinSnapshot::setOriginalMatrixByCol (**const** CoinPackedMatrix * *matrix*, **bool** *copyIn* = **true**)

Set pointer to column-wise copy of "original" matrix.

9.78.3.54 **void** CoinSnapshot::setColSolution (**const** double * *array*, **bool** *copyIn* = **true**)

Set pointer to array[getNumCols()] of primal variable values.

9.78.3.55 **void** CoinSnapshot::setRowPrice (**const** double * *array*, **bool** *copyIn* = **true**)

Set pointer to array[getNumRows()] of dual variable values.

9.78.3.56 **void** CoinSnapshot::setReducedCost (**const** double * *array*, **bool** *copyIn* = **true**)

Set a pointer to array[getNumCols()] of reduced costs.

9.78.3.57 **void** CoinSnapshot::setRowActivity (**const** double * *array*, **bool** *copyIn* = **true**)

Set pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).

9.78.3.58 **void** CoinSnapshot::setDoNotSeparateThis (**const** double * *array*, **bool** *copyIn* = **true**)

Set pointer to array[getNumCols()] of primal variable values which should not be separated (for debug)

9.78.3.59 **void** CoinSnapshot::setInfinity (double *value*) [inline]

Set solver's value for infinity.

Definition at line 301 of file CoinSnapshot.hpp.

9.78.3.60 **void** CoinSnapshot::setObjValue (double *value*) [inline]

Set objective function value (including any rhs offset)

Definition at line 305 of file CoinSnapshot.hpp.

9.78.3.61 **void** CoinSnapshot::setObjOffset (double *value*) [inline]

Set objective offset i.e. $\sum c_{sub\ j} * x_{sub\ j} - objValue = objOffset$.

Definition at line 309 of file CoinSnapshot.hpp.

9.78.3.62 **void** CoinSnapshot::setDualTolerance (double *value*) [inline]

Set dual tolerance.

Definition at line 313 of file CoinSnapshot.hpp.

9.78.3.63 **void** CoinSnapshot::setPrimalTolerance (double *value*) [inline]

Set primal tolerance.

Definition at line 317 of file CoinSnapshot.hpp.

9.78.3.64 `void CoinSnapshot::setIntegerTolerance (double value) [inline]`

Set integer tolerance.

Definition at line 321 of file CoinSnapshot.hpp.

9.78.3.65 `void CoinSnapshot::setIntegerUpperBound (double value) [inline]`

Set integer upper bound i.e. best solution * getObjSense.

Definition at line 325 of file CoinSnapshot.hpp.

9.78.3.66 `void CoinSnapshot::setIntegerLowerBound (double value) [inline]`

Set integer lower bound i.e. best possible solution * getObjSense.

Definition at line 329 of file CoinSnapshot.hpp.

9.78.3.67 `CoinSnapshot& CoinSnapshot::operator= (const CoinSnapshot & rhs)`

Assignment operator.

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

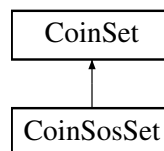
- </home/ted/COIN/trunk/CoinUtils/src/CoinSnapshot.hpp>

9.79 CoinSosSet Class Reference

Very simple class for containing SOS set.

```
#include <CoinMpsIO.hpp>
```

Inheritance diagram for CoinSosSet:



Public Member Functions

Constructor and destructor

- `CoinSosSet` (int [numberEntries](#), const int *[which](#), const double *[weights](#), int type)
Constructor.
- virtual `~CoinSosSet` ()
Destructor.

Additional Inherited Members

9.79.1 Detailed Description

Very simple class for containing SOS set.

Definition at line 286 of file CoinMpsIO.hpp.

9.79.2 Constructor & Destructor Documentation

9.79.2.1 CoinSosSet::CoinSosSet (int *numberEntries*, const int * *which*, const double * *weights*, int *type*)

Constructor.

9.79.2.2 virtual CoinSosSet::~~CoinSosSet () [virtual]

Destructor.

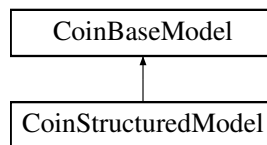
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/CoinUtils/src/CoinMpsIO.hpp

9.80 CoinStructuredModel Class Reference

```
#include <CoinStructuredModel.hpp>
```

Inheritance diagram for CoinStructuredModel:



Public Member Functions

Useful methods for building model

- int [addBlock](#) (const std::string &[rowBlock](#), const std::string &[columnBlock](#), const [CoinBaseModel](#) &[block](#))
add a block from a [CoinModel](#) using names in model returns number of errors (e.g.
- int [addBlock](#) (const [CoinBaseModel](#) &[block](#))
add a block from a [CoinModel](#) with names in model returns number of errors (e.g.
- int [addBlock](#) (const std::string &[rowBlock](#), const std::string &[columnBlock](#), [CoinBaseModel](#) *[block](#))
add a block from a [CoinModel](#) using names given as parameters returns number of errors (e.g.
- int [addBlock](#) (const std::string &[rowBlock](#), const std::string &[columnBlock](#), const [CoinPackedMatrix](#) &[matrix](#), const double *[rowLower](#), const double *[rowUpper](#), const double *[columnLower](#), const double *[columnUpper](#), const double *[objective](#))
add a block using names
- int [writeMps](#) (const char *[filename](#), int [compression](#)=0, int [formatType](#)=0, int [numberAcross](#)=2, bool [keepStrings](#)=false)
Write the problem in MPS format to a file with the given filename.
- int [decompose](#) (const [CoinModel](#) &[model](#), int [type](#), int [maxBlocks](#)=50)
Decompose a [CoinModel](#) 1 - try D-W 2 - try Benders 3 - try Staircase Returns number of blocks or zero if no structure.
- int [decompose](#) (const [CoinPackedMatrix](#) &[matrix](#), const double *[rowLower](#), const double *[rowUpper](#), const double *[columnLower](#), const double *[columnUpper](#), const double *[objective](#), int [type](#), int [maxBlocks](#)=50, double [objectiveOffset](#)=0.0)
Decompose a model specified as arrays + [CoinPackedMatrix](#) 1 - try D-W 2 - try Benders 3 - try Staircase Returns number of blocks or zero if no structure.

For getting information

- int [numberRowBlocks](#) () const
Return number of row blocks.
- int [numberColumnBlocks](#) () const
Return number of column blocks.
- [CoinBigIndex](#) [numberElementBlocks](#) () const
Return number of elementBlocks.
- [CoinBigIndex](#) [numberElements](#) () const
Return number of elements.
- const std::string & [getRowBlock](#) (int i) const
Return the i'th row block name.
- void [setRowBlock](#) (int i, const std::string &name)
Set i'th row block name.
- int [addRowBlock](#) (int [numberRows](#), const std::string &name)
Add or check a row block name and number of rows.
- int [rowBlock](#) (const std::string &name) const
Return a row block index given a row block name.
- const std::string & [getColumnBlock](#) (int i) const
Return i'th the column block name.
- void [setColumnBlock](#) (int i, const std::string &name)
Set i'th column block name.
- int [addColumnBlock](#) (int [numberColumns](#), const std::string &name)
Add or check a column block name and number of columns.
- int [columnBlock](#) (const std::string &name) const
Return a column block index given a column block name.
- const [CoinModelBlockInfo](#) & [blockType](#) (int i) const
Return i'th block type.
- [CoinBaseModel](#) * [block](#) (int i) const
Return i'th block.
- const [CoinBaseModel](#) * [block](#) (int row, int column) const
Return block corresponding to row and column.
- [CoinModel](#) * [coinBlock](#) (int i) const
Return i'th block as [CoinModel](#) (or NULL)
- const [CoinBaseModel](#) * [coinBlock](#) (int row, int column) const
Return block corresponding to row and column as [CoinModel](#).
- int [blockIndex](#) (int row, int column) const
Return block number corresponding to row and column.
- [CoinModel](#) * [coinModelBlock](#) ([CoinModelBlockInfo](#) &info)
Return model as a [CoinModel](#) block and fill in info structure and update counts.
- void [setCoinModel](#) ([CoinModel](#) *[block](#), int iBlock)
Sets given block into coinModelBlocks_.
- void [refresh](#) (int iBlock)
Refresh info in blockType_.
- [CoinModelBlockInfo](#) [block](#) (int row, int column, const double *&rowLower, const double *&rowUpper, const double *&columnLower, const double *&columnUpper, const double *&objective) const
Fill pointers corresponding to row and column.
- double [optimizationDirection](#) () const
Direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.
- void [setOptimizationDirection](#) (double value)
Set direction of optimization (1 - minimize, -1 - maximize, 0 - ignore.

Constructors, destructor

- [CoinStructuredModel](#) ()

Default constructor.

- [CoinStructuredModel](#) (const char *fileName, int [decompose](#)=0, int maxBlocks=50)
Read a problem in MPS format from the given filename.
- virtual [~CoinStructuredModel](#) ()
Destructor.

Copy method

- [CoinStructuredModel](#) (const [CoinStructuredModel](#) &)
The copy constructor.
- [CoinStructuredModel](#) & [operator=](#) (const [CoinStructuredModel](#) &)
=
• virtual [CoinBaseModel](#) * [clone](#) () const
Clone.

Additional Inherited Members

9.80.1 Detailed Description

Definition at line 36 of file [CoinStructuredModel.hpp](#).

9.80.2 Constructor & Destructor Documentation

9.80.2.1 [CoinStructuredModel::CoinStructuredModel](#) ()

Default constructor.

9.80.2.2 [CoinStructuredModel::CoinStructuredModel](#) (const char * *fileName*, int *decompose* = 0, int *maxBlocks* = 50)

Read a problem in MPS format from the given filename.

May try and decompose

9.80.2.3 virtual [CoinStructuredModel::~CoinStructuredModel](#) () [virtual]

Destructor.

9.80.2.4 [CoinStructuredModel::CoinStructuredModel](#) (const [CoinStructuredModel](#) &)

The copy constructor.

9.80.3 Member Function Documentation

9.80.3.1 int [CoinStructuredModel::addBlock](#) (const std::string & *rowBlock*, const std::string & *columnBlock*, const [CoinBaseModel](#) & *block*)

add a block from a [CoinModel](#) using names given as parameters returns number of errors (e.g. both have objectives but not same)

9.80.3.2 int [CoinStructuredModel::addBlock](#) (const [CoinBaseModel](#) & *block*)

add a block from a [CoinModel](#) with names in model returns number of errors (e.g. both have objectives but not same)

9.80.3.3 `int CoinStructuredModel::addBlock (const std::string & rowBlock, const std::string & columnBlock, CoinBaseModel * block)`

add a block from a [CoinModel](#) using names given as parameters returns number of errors (e.g.

both have objectives but not same) This passes in block - structured model takes ownership

9.80.3.4 `int CoinStructuredModel::addBlock (const std::string & rowBlock, const std::string & columnBlock, const CoinPackedMatrix & matrix, const double * rowLower, const double * rowUpper, const double * columnLower, const double * columnUpper, const double * objective)`

add a block using names

9.80.3.5 `int CoinStructuredModel::writeMps (const char * filename, int compression = 0, int formatType = 0, int numberAcross = 2, bool keepStrings = false)`

Write the problem in MPS format to a file with the given filename.

Parameters

<i>compression</i>	can be set to three values to indicate what kind of file should be written <ul style="list-style-type: none"> • 0: plain text (default) • 1: gzip compressed (.gz is appended to <i>filename</i>) • 2: bzip2 compressed (.bz2 is appended to <i>filename</i>) (TODO) <p>If the library was not compiled with the requested compression then writeMps falls back to writing a plain text file.</p>
<i>formatType</i>	specifies the precision to used for values in the MPS file <ul style="list-style-type: none"> • 0: normal precision (default) • 1: extra accuracy • 2: IEEE hex
<i>numberAcross</i>	specifies whether 1 or 2 (default) values should be specified on every data line in the MPS file.

not const as may change model e.g. fill in default bounds

9.80.3.6 `int CoinStructuredModel::decompose (const CoinModel & model, int type, int maxBlocks = 50)`

Decompose a [CoinModel](#) 1 - try D-W 2 - try Benders 3 - try Staircase Returns number of blocks or zero if no structure.

9.80.3.7 `int CoinStructuredModel::decompose (const CoinPackedMatrix & matrix, const double * rowLower, const double * rowUpper, const double * columnLower, const double * columnUpper, const double * objective, int type, int maxBlocks = 50, double objectiveOffset = 0.0)`

Decompose a model specified as arrays + [CoinPackedMatrix](#) 1 - try D-W 2 - try Benders 3 - try Staircase Returns number of blocks or zero if no structure.

9.80.3.8 `int CoinStructuredModel::numberRowBlocks () const [inline]`

Return number of row blocks.

Definition at line 120 of file CoinStructuredModel.hpp.

9.80.3.9 `int CoinStructuredModel::numberColumnBlocks () const [inline]`

Return number of column blocks.

Definition at line 123 of file CoinStructuredModel.hpp.

9.80.3.10 `CoinBigIndex CoinStructuredModel::numberElementBlocks () const [inline]`

Return number of elementBlocks.

Definition at line 126 of file CoinStructuredModel.hpp.

9.80.3.11 `CoinBigIndex CoinStructuredModel::numberElements () const [virtual]`

Return number of elements.

Implements [CoinBaseModel](#).

9.80.3.12 `const std::string& CoinStructuredModel::getRowBlock (int i) const [inline]`

Return the i'th row block name.

Definition at line 131 of file CoinStructuredModel.hpp.

9.80.3.13 `void CoinStructuredModel::setRowBlock (int i, const std::string & name) [inline]`

Set i'th row block name.

Definition at line 134 of file CoinStructuredModel.hpp.

9.80.3.14 `int CoinStructuredModel::addRowBlock (int numberOfRows, const std::string & name)`

Add or check a row block name and number of rows.

9.80.3.15 `int CoinStructuredModel::rowBlock (const std::string & name) const`

Return a row block index given a row block name.

9.80.3.16 `const std::string& CoinStructuredModel::getColumnBlock (int i) const [inline]`

Return i'th the column block name.

Definition at line 141 of file CoinStructuredModel.hpp.

9.80.3.17 `void CoinStructuredModel::setColumnBlock (int i, const std::string & name) [inline]`

Set i'th column block name.

Definition at line 144 of file CoinStructuredModel.hpp.

9.80.3.18 `int CoinStructuredModel::addColumnBlock (int numberColumns, const std::string & name)`

Add or check a column block name and number of columns.

9.80.3.19 `int CoinStructuredModel::columnBlock (const std::string & name) const`

Return a column block index given a column block name.

9.80.3.20 `const CoinModelBlockInfo& CoinStructuredModel::blockType (int i) const [inline]`

Return i'th block type.

Definition at line 151 of file CoinStructuredModel.hpp.

9.80.3.21 `CoinBaseModel* CoinStructuredModel::block (int i) const` `[inline]`

Return *i*'th block.

Definition at line 154 of file CoinStructuredModel.hpp.

9.80.3.22 `const CoinBaseModel* CoinStructuredModel::block (int row, int column) const`

Return block corresponding to row and column.

9.80.3.23 `CoinModel* CoinStructuredModel::coinBlock (int i) const`

Return *i*'th block as [CoinModel](#) (or NULL)

9.80.3.24 `const CoinBaseModel* CoinStructuredModel::coinBlock (int row, int column) const`

Return block corresponding to row and column as [CoinModel](#).

9.80.3.25 `int CoinStructuredModel::blockIndex (int row, int column) const`

Return block number corresponding to row and column.

9.80.3.26 `CoinModel* CoinStructuredModel::coinModelBlock (CoinModelBlockInfo & info)`

Return model as a [CoinModel](#) block and fill in info structure and update counts.

9.80.3.27 `void CoinStructuredModel::setCoinModel (CoinModel * block, int iBlock)`

Sets given block into coinModelBlocks_.

9.80.3.28 `void CoinStructuredModel::refresh (int iBlock)`

Refresh info in blockType_.

9.80.3.29 `CoinModelBlockInfo CoinStructuredModel::block (int row, int column, const double *& rowLower, const double *& rowUpper, const double *& columnLower, const double *& columnUpper, const double *& objective) const`

Fill pointers corresponding to row and column.

9.80.3.30 `double CoinStructuredModel::optimizationDirection () const` `[inline]`

Direction of optimization (1 - minimize, -1 - maximize, 0 - ignore).

Definition at line 179 of file CoinStructuredModel.hpp.

9.80.3.31 `void CoinStructuredModel::setOptimizationDirection (double value)` `[inline]`

Set direction of optimization (1 - minimize, -1 - maximize, 0 - ignore).

Definition at line 183 of file CoinStructuredModel.hpp.

9.80.3.32 `CoinStructuredModel& CoinStructuredModel::operator= (const CoinStructuredModel &)`

=

9.80.3.33 `virtual CoinBaseModel* CoinStructuredModel::clone () const` [virtual]

Clone.

Implements [CoinBaseModel](#).

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinStructuredModel.hpp](#)

9.81 CoinThreadRandom Class Reference

Class for thread specific random numbers.

```
#include <CoinHelperFunctions.hpp>
```

Public Member Functions

Constructors, destructor

- [CoinThreadRandom](#) ()
Default constructor.
- [CoinThreadRandom](#) (int seed)
Constructor with seed.
- [~CoinThreadRandom](#) ()
Destructor.
- [CoinThreadRandom](#) (const [CoinThreadRandom](#) &rhs)
- [CoinThreadRandom](#) & [operator=](#) (const [CoinThreadRandom](#) &rhs)

Sets/gets

- [void setSeed](#) (int seed)
Set seed.
- unsigned int [getSeed](#) () const
Get seed.
- double [randomDouble](#) () const
return a random number
- [void randomize](#) (int n=0)
make more random (i.e. for startup)

Protected Attributes

Data members

The data members are protected to allow access for derived classes.

- unsigned int [seed_](#)
Current seed.

9.81.1 Detailed Description

Class for thread specific random numbers.

Definition at line 951 of file [CoinHelperFunctions.hpp](#).

9.81.2 Constructor & Destructor Documentation

9.81.2.1 CoinThreadRandom::CoinThreadRandom () [inline]

Default constructor.

Definition at line 957 of file CoinHelperFunctions.hpp.

9.81.2.2 CoinThreadRandom::CoinThreadRandom (int *seed*) [inline]

Constructor with seed.

Definition at line 960 of file CoinHelperFunctions.hpp.

9.81.2.3 CoinThreadRandom::~~CoinThreadRandom () [inline]

Destructor.

Definition at line 965 of file CoinHelperFunctions.hpp.

9.81.2.4 CoinThreadRandom::CoinThreadRandom (const CoinThreadRandom & *rhs*) [inline]

Definition at line 967 of file CoinHelperFunctions.hpp.

9.81.3 Member Function Documentation

9.81.3.1 CoinThreadRandom& CoinThreadRandom::operator= (const CoinThreadRandom & *rhs*) [inline]

Definition at line 970 of file CoinHelperFunctions.hpp.

9.81.3.2 void CoinThreadRandom::setSeed (int *seed*) [inline]

Set seed.

Definition at line 984 of file CoinHelperFunctions.hpp.

9.81.3.3 unsigned int CoinThreadRandom::getSeed () const [inline]

Get seed.

Definition at line 989 of file CoinHelperFunctions.hpp.

9.81.3.4 double CoinThreadRandom::randomDouble () const [inline]

return a random number

Definition at line 994 of file CoinHelperFunctions.hpp.

9.81.3.5 void CoinThreadRandom::randomize (int *n* = 0) [inline]

make more random (i.e. for startup)

Definition at line 1002 of file CoinHelperFunctions.hpp.

9.81.4 Member Data Documentation

9.81.4.1 unsigned int CoinThreadRandom::seed_ [mutable],[protected]

Current seed.

Definition at line 1017 of file CoinHelperFunctions.hpp.

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

- /home/ted/COIN/trunk/CoinUtils/src/CoinHelperFunctions.hpp

9.82 CoinTimer Class Reference

This class implements a timer that also implements a tracing functionality.

```
#include <CoinTime.hpp>
```

Public Member Functions

- [CoinTimer](#) ()
Default constructor creates a timer with no time limit and no tracing.
- [CoinTimer](#) (double lim)
Create a timer with the given time limit and with no tracing.
- [void restart](#) ()
Restart the timer (keeping the same time limit)
- [void reset](#) ()
An alternate name for [restart](#) ()
- [void reset](#) (double lim)
Reset (and restart) the timer and change its time limit.
- [bool isPastPercent](#) (double pct) const
Return whether the given percentage of the time limit has elapsed since the timer was started.
- [bool isPast](#) (double lim) const
Return whether the given amount of time has elapsed since the timer was started.
- [bool isExpired](#) () const
Return whether the originally specified time limit has passed since the timer was started.
- [double timeLeft](#) () const
Return how much time is left on the timer.
- [double timeElapsed](#) () const
Return how much time has elapsed.
- [void setLimit](#) (double l)

9.82.1 Detailed Description

This class implements a timer that also implements a tracing functionality.

The timer stores the start time of the timer, for how much time it was set to and when does it expire (start + limit = end). Queries can be made that tell whether the timer is expired, is past an absolute time, is past a percentage of the length of the timer. All times are given in seconds, but as double numbers, so there can be fractional values.

The timer can also be initialized with a stream and a specification whether to write to or read from the stream. In the former case the result of every query is written into the stream, in the latter case timing is not tested at all, rather the supposed result is read out from the stream. This makes it possible to exactly retrace time sensitive program execution.

Definition at line 197 of file CoinTime.hpp.

9.82.2 Constructor & Destructor Documentation

9.82.2.1 CoinTimer::CoinTimer () [inline]

Default constructor creates a timer with no time limit and no tracing.

Definition at line 242 of file CoinTime.hpp.

9.82.2.2 CoinTimer::CoinTimer (double *lim*) [inline]

Create a timer with the given time limit and with no tracing.

Definition at line 250 of file CoinTime.hpp.

9.82.3 Member Function Documentation

9.82.3.1 void CoinTimer::restart () [inline]

Restart the timer (keeping the same time limit)

Definition at line 272 of file CoinTime.hpp.

9.82.3.2 void CoinTimer::reset () [inline]

An alternate name for [restart\(\)](#)

Definition at line 274 of file CoinTime.hpp.

9.82.3.3 void CoinTimer::reset (double *lim*) [inline]

Reset (and restart) the timer and change its time limit.

Definition at line 276 of file CoinTime.hpp.

9.82.3.4 bool CoinTimer::isPastPercent (double *pct*) const [inline]

Return whether the given percentage of the time limit has elapsed since the timer was started.

Definition at line 280 of file CoinTime.hpp.

9.82.3.5 bool CoinTimer::isPast (double *lim*) const [inline]

Return whether the given amount of time has elapsed since the timer was started.

Definition at line 285 of file CoinTime.hpp.

9.82.3.6 bool CoinTimer::isExpired () const [inline]

Return whether the originally specified time limit has passed since the timer was started.

Definition at line 290 of file CoinTime.hpp.

9.82.3.7 double CoinTimer::timeLeft () const [inline]

Return how much time is left on the timer.

Definition at line 295 of file CoinTime.hpp.

9.82.3.8 `double CoinTimer::timeElapsed () const [inline]`

Return how much time has elapsed.

Definition at line 300 of file `CoinTime.hpp`.

9.82.3.9 `void CoinTimer::setLimit (double l) [inline]`

Definition at line 304 of file `CoinTime.hpp`.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinTime.hpp](#)

9.83 CoinTreeNode Class Reference

A class from which the real tree nodes should be derived from.

```
#include <CoinSearchTree.hpp>
```

Public Member Functions

- virtual `~CoinTreeNode ()`
- int `getDepth () const`
- int `getFractionality () const`
- double `getQuality () const`
- double `getTrueLB () const`
- `BitVector128` `getPreferred () const`
- void `setDepth (int d)`
- void `setFractionality (int f)`
- void `setQuality (double q)`
- void `setTrueLB (double tlb)`
- void `setPreferred (BitVector128 p)`

Protected Member Functions

- `CoinTreeNode ()`
- `CoinTreeNode (int d, int f=-1, double q=-COIN_DBL_MAX, double tlb=-COIN_DBL_MAX, BitVector128 p=BitVector128())`
- `CoinTreeNode (const CoinTreeNode &x)`
- `CoinTreeNode & operator= (const CoinTreeNode &x)`

9.83.1 Detailed Description

A class from which the real tree nodes should be derived from.

Some of the data that undoubtedly exist in the real tree node is replicated here for fast access. This class is used in the various comparison functions.

Definition at line 42 of file `CoinSearchTree.hpp`.

9.83.2 Constructor & Destructor Documentation

9.83.2.1 CoinTreeNode::CoinTreeNode () [inline], [protected]

Definition at line 44 of file CoinSearchTree.hpp.

9.83.2.2 CoinTreeNode::CoinTreeNode (int *d*, int *f* = -1, double *q* = -COIN_DBL_MAX, double *t/b* = -COIN_DBL_MAX, BitVector128 *p* = BitVector128 ()) [inline], [protected]

Definition at line 50 of file CoinSearchTree.hpp.

9.83.2.3 CoinTreeNode::CoinTreeNode (const CoinTreeNode & *x*) [inline], [protected]

Definition at line 60 of file CoinSearchTree.hpp.

9.83.2.4 virtual CoinTreeNode::~CoinTreeNode () [inline], [virtual]

Definition at line 93 of file CoinSearchTree.hpp.

9.83.3 Member Function Documentation

9.83.3.1 CoinTreeNode& CoinTreeNode::operator= (const CoinTreeNode & *x*) [inline], [protected]

Definition at line 66 of file CoinSearchTree.hpp.

9.83.3.2 int CoinTreeNode::getDepth () const [inline]

Definition at line 95 of file CoinSearchTree.hpp.

9.83.3.3 int CoinTreeNode::getFractionality () const [inline]

Definition at line 96 of file CoinSearchTree.hpp.

9.83.3.4 double CoinTreeNode::getQuality () const [inline]

Definition at line 97 of file CoinSearchTree.hpp.

9.83.3.5 double CoinTreeNode::getTrueLB () const [inline]

Definition at line 98 of file CoinSearchTree.hpp.

9.83.3.6 BitVector128 CoinTreeNode::getPreferred () const [inline]

Definition at line 99 of file CoinSearchTree.hpp.

9.83.3.7 void CoinTreeNode::setDepth (int *d*) [inline]

Definition at line 101 of file CoinSearchTree.hpp.

9.83.3.8 void CoinTreeNode::setFractionality (int *f*) [inline]

Definition at line 102 of file CoinSearchTree.hpp.

9.83.3.9 void CoinTreeNode::setQuality (double *q*) [inline]

Definition at line 103 of file CoinSearchTree.hpp.

9.83.3.10 `void CoinTreeNode::setTrueLB (double tlb) [inline]`

Definition at line 104 of file `CoinSearchTree.hpp`.

9.83.3.11 `void CoinTreeNode::setPreferred (BitVector128 p) [inline]`

Definition at line 105 of file `CoinSearchTree.hpp`.

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

- `/home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp`

9.84 CoinTreeSiblings Class Reference

```
#include <CoinSearchTree.hpp>
```

Public Member Functions

- `CoinTreeSiblings` (const int *n*, `CoinTreeNode` ***nodes*)
- `CoinTreeSiblings` (const `CoinTreeSiblings` &*s*)
- `~CoinTreeSiblings` ()
- `CoinTreeNode * currentNode` () const
- `bool advanceNode` ()
returns false if cannot be advanced
- `int toProcess` () const
- `int size` () const
- `void printPref` () const

9.84.1 Detailed Description

Definition at line 110 of file `CoinSearchTree.hpp`.

9.84.2 Constructor & Destructor Documentation

9.84.2.1 `CoinTreeSiblings::CoinTreeSiblings (const int n, CoinTreeNode ** nodes) [inline]`

Definition at line 119 of file `CoinSearchTree.hpp`.

9.84.2.2 `CoinTreeSiblings::CoinTreeSiblings (const CoinTreeSiblings & s) [inline]`

Definition at line 124 of file `CoinSearchTree.hpp`.

9.84.2.3 `CoinTreeSiblings::~~CoinTreeSiblings () [inline]`

Definition at line 131 of file `CoinSearchTree.hpp`.

9.84.3 Member Function Documentation

9.84.3.1 `CoinTreeNode* CoinTreeSiblings::currentNode () const [inline]`

Definition at line 132 of file `CoinSearchTree.hpp`.

9.84.3.2 `bool CoinTreeSiblings::advanceNode () [inline]`

returns false if cannot be advanced

Definition at line 134 of file CoinSearchTree.hpp.

9.84.3.3 `int CoinTreeSiblings::toProcess () const [inline]`

Definition at line 135 of file CoinSearchTree.hpp.

9.84.3.4 `int CoinTreeSiblings::size () const [inline]`

Definition at line 136 of file CoinSearchTree.hpp.

9.84.3.5 `void CoinTreeSiblings::printPref () const [inline]`

Definition at line 137 of file CoinSearchTree.hpp.

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp>

9.85 CoinTriple< S, T, U > Class Template Reference

```
#include <CoinSort.hpp>
```

Public Member Functions

- [CoinTriple](#) (const S &s, const T &t, const U &u)
Construct from ordered triple.

Public Attributes

- S [first](#)
First member of triple.
- T [second](#)
Second member of triple.
- U [third](#)
Third member of triple.

9.85.1 Detailed Description

```
template<class S, class T, class U>class CoinTriple< S, T, U >
```

Definition at line 459 of file CoinSort.hpp.

9.85.2 Constructor & Destructor Documentation

9.85.2.1 `template<class S, class T, class U> CoinTriple< S, T, U >::CoinTriple (const S & s, const T & t, const U & u) [inline]`

Construct from ordered triple.

Definition at line 469 of file CoinSort.hpp.

9.85.3 Member Data Documentation

9.85.3.1 `template<class S, class T, class U> S CoinTriple< S, T, U >::first`

First member of triple.

Definition at line 462 of file CoinSort.hpp.

9.85.3.2 `template<class S, class T, class U> T CoinTriple< S, T, U >::second`

Second member of triple.

Definition at line 464 of file CoinSort.hpp.

9.85.3.3 `template<class S, class T, class U> U CoinTriple< S, T, U >::third`

Third member of triple.

Definition at line 466 of file CoinSort.hpp.

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

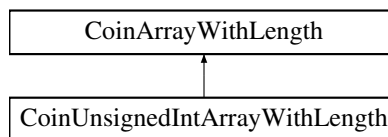
- [/home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp](#)

9.86 CoinUnsignedIntArrayWithLength Class Reference

unsigned int * version

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for CoinUnsignedIntArrayWithLength:



Public Member Functions

Get methods.

- `int getSize () const`
Get the size.
- `unsigned int * array () const`
Get Array.

Set methods

- `void setSize (int value)`
Set the size.

Condition methods

- unsigned int * [conditionalNew](#) (int sizeWanted)
Conditionally gets new array.

Constructors and destructors

- [CoinUnsignedIntArrayWithLength](#) ()
Default constructor - NULL.
- [CoinUnsignedIntArrayWithLength](#) (int size)
Alternate Constructor - length in bytes - size_ - 1.
- [CoinUnsignedIntArrayWithLength](#) (int size, int mode)
Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.
- [CoinUnsignedIntArrayWithLength](#) (const [CoinUnsignedIntArrayWithLength](#) &rhs)
Copy constructor.
- [CoinUnsignedIntArrayWithLength](#) (const [CoinUnsignedIntArrayWithLength](#) *rhs)
Copy constructor.2.
- [CoinUnsignedIntArrayWithLength](#) & [operator=](#) (const [CoinUnsignedIntArrayWithLength](#) &rhs)
Assignment operator.

Additional Inherited Members

9.86.1 Detailed Description

unsigned int * version

Definition at line 887 of file CoinIndexedVector.hpp.

9.86.2 Constructor & Destructor Documentation

9.86.2.1 CoinUnsignedIntArrayWithLength::CoinUnsignedIntArrayWithLength () [inline]

Default constructor - NULL.

Definition at line 917 of file CoinIndexedVector.hpp.

9.86.2.2 CoinUnsignedIntArrayWithLength::CoinUnsignedIntArrayWithLength (int size) [inline]

Alternate Constructor - length in bytes - size_ - 1.

Definition at line 920 of file CoinIndexedVector.hpp.

9.86.2.3 CoinUnsignedIntArrayWithLength::CoinUnsignedIntArrayWithLength (int size, int mode) [inline]

Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.

Definition at line 926 of file CoinIndexedVector.hpp.

9.86.2.4 CoinUnsignedIntArrayWithLength::CoinUnsignedIntArrayWithLength (const CoinUnsignedIntArrayWithLength & rhs) [inline]

Copy constructor.

Definition at line 929 of file CoinIndexedVector.hpp.

9.86.2.5 CoinUnsignedIntArrayWithLength::CoinUnsignedIntArrayWithLength (const CoinUnsignedIntArrayWithLength * rhs) [inline]

Copy constructor.2.

Definition at line 932 of file CoinIndexedVector.hpp.

9.86.3 Member Function Documentation

9.86.3.1 `int CoinUnsignedIntArrayWithLength::getSize () const` `[inline]`

Get the size.

Definition at line 893 of file CoinIndexedVector.hpp.

9.86.3.2 `unsigned int* CoinUnsignedIntArrayWithLength::array () const` `[inline]`

Get Array.

Definition at line 896 of file CoinIndexedVector.hpp.

9.86.3.3 `void CoinUnsignedIntArrayWithLength::setSize (int value)` `[inline]`

Set the size.

Definition at line 903 of file CoinIndexedVector.hpp.

9.86.3.4 `unsigned int* CoinUnsignedIntArrayWithLength::conditionalNew (int sizeWanted)` `[inline]`

Conditionally gets new array.

Definition at line 910 of file CoinIndexedVector.hpp.

9.86.3.5 `CoinUnsignedIntArrayWithLength& CoinUnsignedIntArrayWithLength::operator= (const CoinUnsignedIntArrayWithLength & rhs)` `[inline]`

Assignment operator.

Definition at line 935 of file CoinIndexedVector.hpp.

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

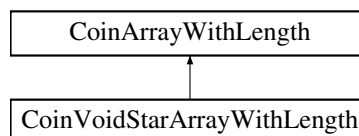
- [/home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp](#)

9.87 CoinVoidStarArrayWithLength Class Reference

void * version

```
#include <CoinIndexedVector.hpp>
```

Inheritance diagram for CoinVoidStarArrayWithLength:



Public Member Functions

Get methods.

- `int getSize () const`
Get the size.
- `void ** array () const`
Get Array.

Set methods

- `void setSize (int value)`
Set the size.

Condition methods

- `void ** conditionalNew (int sizeWanted)`
Conditionally gets new array.

Constructors and destructors

- `CoinVoidStarArrayWithLength ()`
Default constructor - NULL.
- `CoinVoidStarArrayWithLength (int size)`
Alternate Constructor - length in bytes - size_ -1.
- `CoinVoidStarArrayWithLength (int size, int mode)`
Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.
- `CoinVoidStarArrayWithLength (const CoinVoidStarArrayWithLength &rhs)`
Copy constructor.
- `CoinVoidStarArrayWithLength (const CoinVoidStarArrayWithLength *rhs)`
Copy constructor.2.
- `CoinVoidStarArrayWithLength & operator= (const CoinVoidStarArrayWithLength &rhs)`
Assignment operator.

Additional Inherited Members

9.87.1 Detailed Description

`void * version`

Definition at line 941 of file `CoinIndexedVector.hpp`.

9.87.2 Constructor & Destructor Documentation

9.87.2.1 `CoinVoidStarArrayWithLength::CoinVoidStarArrayWithLength ()` `[inline]`

Default constructor - NULL.

Definition at line 971 of file `CoinIndexedVector.hpp`.

9.87.2.2 `CoinVoidStarArrayWithLength::CoinVoidStarArrayWithLength (int size)` `[inline]`

Alternate Constructor - length in bytes - size_ -1.

Definition at line 974 of file `CoinIndexedVector.hpp`.

9.87.2.3 `CoinVoidStarArrayWithLength::CoinVoidStarArrayWithLength (int size, int mode)` `[inline]`

Alternate Constructor - length in bytes mode - 0 size_ set to size 1 size_ set to size and zeroed.

Definition at line 980 of file `CoinIndexedVector.hpp`.

9.87.2.4 `CoinVoidStarArrayWithLength::CoinVoidStarArrayWithLength (const CoinVoidStarArrayWithLength & rhs)`
`[inline]`

Copy constructor.

Definition at line 983 of file `CoinIndexedVector.hpp`.

9.87.2.5 `CoinVoidStarArrayWithLength::CoinVoidStarArrayWithLength (const CoinVoidStarArrayWithLength * rhs)`
`[inline]`

Copy constructor.2.

Definition at line 986 of file `CoinIndexedVector.hpp`.

9.87.3 Member Function Documentation

9.87.3.1 `int CoinVoidStarArrayWithLength::getSize () const` `[inline]`

Get the size.

Definition at line 947 of file `CoinIndexedVector.hpp`.

9.87.3.2 `void** CoinVoidStarArrayWithLength::array () const` `[inline]`

Get Array.

Definition at line 950 of file `CoinIndexedVector.hpp`.

9.87.3.3 `void CoinVoidStarArrayWithLength::setSize (int value)` `[inline]`

Set the size.

Definition at line 957 of file `CoinIndexedVector.hpp`.

9.87.3.4 `void** CoinVoidStarArrayWithLength::conditionalNew (int sizeWanted)` `[inline]`

Conditionally gets new array.

Definition at line 964 of file `CoinIndexedVector.hpp`.

9.87.3.5 `CoinVoidStarArrayWithLength& CoinVoidStarArrayWithLength::operator= (const CoinVoidStarArrayWithLength & rhs)` `[inline]`

Assignment operator.

Definition at line 989 of file `CoinIndexedVector.hpp`.

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

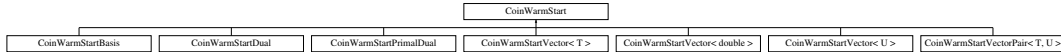
- </home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp>

9.88 CoinWarmStart Class Reference

Abstract base class for warm start information.

`#include <CoinWarmStart.hpp>`

Inheritance diagram for `CoinWarmStart`:



Public Member Functions

- virtual `~CoinWarmStart ()`
Abstract destructor.
- virtual `CoinWarmStart * clone () const =0`
‘Virtual constructor’
- virtual `CoinWarmStartDiff * generateDiff (const CoinWarmStart *const) const`
- virtual `void applyDiff (const CoinWarmStartDiff *const)`

9.88.1 Detailed Description

Abstract base class for warm start information.

Really nothing can be generalized for warm start information — all we know is that it exists. Hence the abstract base class contains only a virtual destructor and a virtual clone function (a virtual constructor), so that derived classes can provide these functions.

Definition at line 21 of file `CoinWarmStart.hpp`.

9.88.2 Constructor & Destructor Documentation

9.88.2.1 virtual `CoinWarmStart::~~CoinWarmStart ()` `[inline]`, `[virtual]`

Abstract destructor.

Definition at line 25 of file `CoinWarmStart.hpp`.

9.88.3 Member Function Documentation

9.88.3.1 virtual `CoinWarmStart* CoinWarmStart::clone () const` `[pure virtual]`

‘Virtual constructor’

Implemented in `CoinWarmStartBasis`, `CoinWarmStartVectorPair< T, U >`, `CoinWarmStartVector< T >`, `CoinWarmStartVector< double >`, `CoinWarmStartVector< U >`, `CoinWarmStartPrimalDual`, and `CoinWarmStartDual`.

9.88.3.2 virtual `CoinWarmStartDiff* CoinWarmStart::generateDiff (const CoinWarmStart *const) const` `[inline]`, `[virtual]`

Reimplemented in `CoinWarmStartVectorPair< T, U >`, `CoinWarmStartBasis`, `CoinWarmStartVector< T >`, `CoinWarmStartVector< double >`, `CoinWarmStartVector< U >`, `CoinWarmStartPrimalDual`, and `CoinWarmStartDual`.

Definition at line 31 of file `CoinWarmStart.hpp`.

9.88.3.3 virtual `void CoinWarmStart::applyDiff (const CoinWarmStartDiff *const)` `[inline]`, `[virtual]`

Reimplemented in `CoinWarmStartVectorPair< T, U >`, `CoinWarmStartBasis`, `CoinWarmStartVector< T >`, `CoinWarmStartVector< double >`, `CoinWarmStartVector< U >`, `CoinWarmStartPrimalDual`, and `CoinWarmStartDual`.

Definition at line 35 of file `CoinWarmStart.hpp`.

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

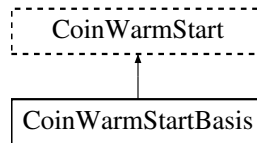
- [/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart.hpp](#)

9.89 CoinWarmStartBasis Class Reference

The default COIN simplex (basis-oriented) warm start class.

```
#include <CoinWarmStartBasis.hpp>
```

Inheritance diagram for CoinWarmStartBasis:



Public Types

- enum [Status](#) { [isFree](#) = 0x00, [basic](#) = 0x01, [atUpperBound](#) = 0x02, [atLowerBound](#) = 0x03 }
- *Enum for status of variables.*
- typedef [CoinTriple](#)< int, int, int > [XferEntry](#)
- *Transfer vector entry for [mergeBasis\(const CoinWarmStartBasis*,const XferVec*,const XferVec*\)](#)*
- typedef std::vector< [XferEntry](#) > [XferVec](#)
- *Transfer vector for [mergeBasis\(const CoinWarmStartBasis*,const XferVec*,const XferVec*\)](#)*

Public Member Functions

Methods to get and set basis information.

The status of variables is kept in a pair of arrays, one for structural variables, and one for artificials (aka logicals and slacks).

The status is coded using the values of the [Status](#) enum.

See Also

[CoinWarmStartBasis::Status](#) for a description of the packing used in the status arrays.

- int [getNumStructural](#) () const
- *Return the number of structural variables.*
- int [getNumArtificial](#) () const
- *Return the number of artificial variables.*
- int [numberBasicStructurals](#) () const
- *Return the number of basic structurals.*
- [Status](#) [getStructStatus](#) (int i) const
- *Return the status of the specified structural variable.*
- void [setStructStatus](#) (int i, [Status](#) st)
- *Set the status of the specified structural variable.*
- char * [getStructuralStatus](#) ()
- *Return the status array for the structural variables.*
- const char * [getStructuralStatus](#) () const
- *const overload for [getStructuralStatus\(\)](#)*
- char * [getArtificialStatus](#) ()
- *As for [getStructuralStatus](#) , but returns the status array for the artificial variables.*

- [Status getArtifStatus](#) (int i) const
Return the status of the specified artificial variable.
- [void setArtifStatus](#) (int i, [Status](#) st)
Set the status of the specified artificial variable.
- const char * [getArtificialStatus](#) () const
const overload for [getArtificialStatus\(\)](#)

Basis 'diff' methods

- virtual [CoinWarmStartDiff](#) * [generateDiff](#) (const [CoinWarmStart](#) *const oldCWS) const
Generate a 'diff' that can convert the warm start basis passed as a parameter to the warm start basis specified by `this`.
- virtual [void applyDiff](#) (const [CoinWarmStartDiff](#) *const cwsdDiff)
Apply `diff` to this basis.

Methods to modify the warm start object

- virtual [void setSize](#) (int ns, int na)
Set basis capacity; existing basis is discarded.
- virtual [void resize](#) (int newNumberRows, int newNumberColumns)
Set basis capacity; existing basis is maintained.
- virtual [void compressRows](#) (int tgtCnt, const int *tgts)
Delete a set of rows from the basis.
- virtual [void deleteRows](#) (int rawTgtCnt, const int *rawTgts)
Delete a set of rows from the basis.
- virtual [void deleteColumns](#) (int number, const int *which)
Delete a set of columns from the basis.
- virtual [void mergeBasis](#) (const [CoinWarmStartBasis](#) *src, const [XferVec](#) *xferRows, const [XferVec](#) *xferCols)
Merge entries from a source basis into this basis.

Constructors, destructors, and related functions

- [CoinWarmStartBasis](#) ()
Default constructor.
- [CoinWarmStartBasis](#) (int ns, int na, const char *sStat, const char *aStat)
Constructs a warm start object with the specified status vectors.
- [CoinWarmStartBasis](#) (const [CoinWarmStartBasis](#) &ws)
Copy constructor.
- virtual [CoinWarmStart](#) * [clone](#) () const
'Virtual constructor'
- virtual [~CoinWarmStartBasis](#) ()
Destructor.
- virtual [CoinWarmStartBasis](#) & [operator=](#) (const [CoinWarmStartBasis](#) &rhs)
Assignment.
- virtual [void assignBasisStatus](#) (int ns, int na, char *&sStat, char *&aStat)
Assign the status vectors to be the warm start information.

Miscellaneous methods

- virtual [void print](#) () const
Prints in readable format (for debug)
- bool [fullBasis](#) () const
Returns true if full basis (for debug)
- bool [fixFullBasis](#) ()
Returns true if full basis and fixes up (for debug)

Protected Attributes

Protected data members

See Also

[CoinWarmStartBasis::Status](#) for a description of the packing used in the status arrays.

- int [numStructural_](#)
The number of structural variables.
- int [numArtificial_](#)
The number of artificial variables.
- int [maxSize_](#)
The maximum size (in ints - actually 4*char) (so resize does not need to do new)
- char * [structuralStatus_](#)
The status of the structural variables.
- char * [artificialStatus_](#)
The status of the artificial variables.

Related Functions

(Note that these are not member functions.)

- [CoinWarmStartBasis::Status getStatus](#) (const char *array, int i)
Get the status of the specified variable in the given status array.
- [void setStatus](#) (char *array, int i, [CoinWarmStartBasis::Status st](#))
Set the status of the specified variable in the given status array.
- const char * [statusName](#) ([CoinWarmStartBasis::Status status](#))
Generate a print string for a status code.

9.89.1 Detailed Description

The default COIN simplex (basis-oriented) warm start class.

[CoinWarmStartBasis](#) provides for a warm start object which contains the status of each variable (structural and artificial).

Todo Modify this class so that the number of status entries per byte and bytes per status vector allocation unit are not hardcoded. At the least, collect this into a couple of macros.

Todo Consider separate fields for allocated capacity and actual basis size. We could avoid some reallocation, at the price of retaining more space than we need. Perhaps more important, we could do much better sanity checks.

Definition at line 40 of file [CoinWarmStartBasis.hpp](#).

9.89.2 Member Typedef Documentation

9.89.2.1 `typedef CoinTriple<int,int,int> CoinWarmStartBasis::XferEntry`

Transfer vector entry for [mergeBasis\(const CoinWarmStartBasis*,const XferVec*,const XferVec*\)](#)

Definition at line 67 of file [CoinWarmStartBasis.hpp](#).

9.89.2.2 `typedef std::vector<XferEntry> CoinWarmStartBasis::XferVec`

Transfer vector for `mergeBasis(const CoinWarmStartBasis*,const XferVec*,const XferVec*)`

Definition at line 72 of file `CoinWarmStartBasis.hpp`.

9.89.3 Member Enumeration Documentation

9.89.3.1 `enum CoinWarmStartBasis::Status`

Enum for status of variables.

Matches `CoinPrePostsolveMatrix::Status`, without `superBasic`. Most code that converts between `CoinPrePostsolveMatrix::Status` and `CoinWarmStartBasis::Status` will break if this correspondence is broken.

The status vectors are currently packed using two bits per status code, four codes per byte. The location of the status information for variable `i` is in byte `i>>2` and occupies bits 0:1 if `i%4 == 0`, bits 2:3 if `i%4 == 1`, etc. The non-member functions `getStatus(const char*,int)` and `setStatus(char*,int,CoinWarmStartBasis::Status)` are provided to hide details of the packing.

Enumerator

isFree Nonbasic free variable.

basic Basic variable.

atUpperBound Nonbasic at upper bound.

atLowerBound Nonbasic at lower bound.

Definition at line 57 of file `CoinWarmStartBasis.hpp`.

9.89.4 Constructor & Destructor Documentation

9.89.4.1 `CoinWarmStartBasis::CoinWarmStartBasis ()`

Default constructor.

Creates a warm start object representing an empty basis (0 rows, 0 columns).

9.89.4.2 `CoinWarmStartBasis::CoinWarmStartBasis (int ns, int na, const char * sStat, const char * aStat)`

Constructs a warm start object with the specified status vectors.

The parameters are copied. Consider `assignBasisStatus(int,int,char*&,char*&)` if the object should assume ownership.

See Also

[CoinWarmStartBasis::Status](#) for a description of the packing used in the status arrays.

9.89.4.3 `CoinWarmStartBasis::CoinWarmStartBasis (const CoinWarmStartBasis & ws)`

Copy constructor.

9.89.4.4 `virtual CoinWarmStartBasis::~~CoinWarmStartBasis () [virtual]`

Destructor.

9.89.5 Member Function Documentation

9.89.5.1 `int CoinWarmStartBasis::getNumStructural () const [inline]`

Return the number of structural variables.

Definition at line 87 of file `CoinWarmStartBasis.hpp`.

9.89.5.2 `int CoinWarmStartBasis::getNumArtificial () const [inline]`

Return the number of artificial variables.

Definition at line 90 of file `CoinWarmStartBasis.hpp`.

9.89.5.3 `int CoinWarmStartBasis::numberBasicStructurals () const`

Return the number of basic structurals.

A fast test for an all-slack basis.

9.89.5.4 `Status CoinWarmStartBasis::getStructStatus (int i) const [inline]`

Return the status of the specified structural variable.

Definition at line 99 of file `CoinWarmStartBasis.hpp`.

9.89.5.5 `void CoinWarmStartBasis::setStructStatus (int i, Status st) [inline]`

Set the status of the specified structural variable.

Definition at line 105 of file `CoinWarmStartBasis.hpp`.

9.89.5.6 `char* CoinWarmStartBasis::getStructuralStatus () [inline]`

Return the status array for the structural variables.

The status information is stored using the codes defined in the `Status` enum, 2 bits per variable, packed 4 variables per byte.

Definition at line 116 of file `CoinWarmStartBasis.hpp`.

9.89.5.7 `const char* CoinWarmStartBasis::getStructuralStatus () const [inline]`

`const` overload for [getStructuralStatus\(\)](#)

Definition at line 123 of file `CoinWarmStartBasis.hpp`.

9.89.5.8 `char* CoinWarmStartBasis::getArtificialStatus () [inline]`

As for [getStructuralStatus](#) , but returns the status array for the artificial variables.

Definition at line 128 of file `CoinWarmStartBasis.hpp`.

9.89.5.9 `Status CoinWarmStartBasis::getArtifStatus (int i) const [inline]`

Return the status of the specified artificial variable.

Definition at line 131 of file `CoinWarmStartBasis.hpp`.

9.89.5.10 `void CoinWarmStartBasis::setArtifStatus (int i, Status st) [inline]`

Set the status of the specified artificial variable.

Definition at line 137 of file CoinWarmStartBasis.hpp.

9.89.5.11 `const char* CoinWarmStartBasis::getArtificialStatus () const [inline]`

const overload for [getArtificialStatus\(\)](#)

Definition at line 148 of file CoinWarmStartBasis.hpp.

9.89.5.12 `virtual CoinWarmStartDiff* CoinWarmStartBasis::generateDiff (const CoinWarmStart *const oldCWS) const [virtual]`

Generate a 'diff' that can convert the warm start basis passed as a parameter to the warm start basis specified by [this](#).

The capabilities are limited: the basis passed as a parameter can be no larger than the basis pointed to by [this](#).

Reimplemented from [CoinWarmStart](#).

9.89.5.13 `virtual void CoinWarmStartBasis::applyDiff (const CoinWarmStartDiff *const cwsdDiff) [virtual]`

Apply [diff](#) to this basis.

Update this basis by applying [diff](#). It's assumed that the allocated capacity of the basis is sufficiently large.

Reimplemented from [CoinWarmStart](#).

9.89.5.14 `virtual void CoinWarmStartBasis::setSize (int ns, int na) [virtual]`

Set basis capacity; existing basis is discarded.

After execution of this routine, the warm start object does not describe a valid basis: all structural and artificial variables have status `isFree`.

9.89.5.15 `virtual void CoinWarmStartBasis::resize (int newNumberRows, int newNumberColumns) [virtual]`

Set basis capacity; existing basis is maintained.

After execution of this routine, the warm start object describes a valid basis: the status of new structural variables (added columns) is set to nonbasic at lower bound, and the status of new artificial variables (added rows) is set to basic. (The basis can be invalid if new structural variables do not have a finite lower bound.)

9.89.5.16 `virtual void CoinWarmStartBasis::compressRows (int tgtCnt, const int * tgts) [virtual]`

Delete a set of rows from the basis.

Warning

This routine assumes that the set of indices to be deleted is sorted in ascending order and contains no duplicates. Use [deleteRows\(\)](#) if this is not the case.

The resulting basis is guaranteed valid only if all deleted constraints are slack (hence the associated logicals are basic).

Removal of a tight constraint with a nonbasic logical implies that some basic variable must be made nonbasic. This correction is left to the client.

9.89.5.17 `virtual void CoinWarmStartBasis::deleteRows (int rawTgtCnt, const int * rawTgts) [virtual]`

Delete a set of rows from the basis.

Warning

The resulting basis is guaranteed valid only if all deleted constraints are slack (hence the associated logicals are basic).

Removal of a tight constraint with a nonbasic logical implies that some basic variable must be made nonbasic. This correction is left to the client.

9.89.5.18 `virtual void CoinWarmStartBasis::deleteColumns (int number, const int * which) [virtual]`

Delete a set of columns from the basis.

Warning

The resulting basis is guaranteed valid only if all deleted variables are nonbasic.

Removal of a basic variable implies that some nonbasic variable must be made basic. This correction is left to the client.

9.89.5.19 `virtual void CoinWarmStartBasis::mergeBasis (const CoinWarmStartBasis * src, const XferVec * xferRows, const XferVec * xferCols) [virtual]`

Merge entries from a source basis into this basis.

Warning

It's the client's responsibility to ensure validity of the merged basis, if that's important to the application.

The vector *xferCols* (*xferRows*) specifies runs of entries to be taken from the source basis and placed in this basis. Each entry is a [CoinTriple](#), with first specifying the starting source index of a run, second specifying the starting destination index, and third specifying the run length.

9.89.5.20 `virtual CoinWarmStart* CoinWarmStartBasis::clone () const [inline],[virtual]`

'Virtual constructor'

Implements [CoinWarmStart](#).

Definition at line 284 of file `CoinWarmStartBasis.hpp`.

9.89.5.21 `virtual CoinWarmStartBasis& CoinWarmStartBasis::operator= (const CoinWarmStartBasis & rhs) [virtual]`

Assignment.

9.89.5.22 `virtual void CoinWarmStartBasis::assignBasisStatus (int ns, int na, char * & sStat, char * & aStat) [virtual]`

Assign the status vectors to be the warm start information.

In this method the [CoinWarmStartBasis](#) object assumes ownership of the pointers and upon return the argument pointers will be NULL. If copying is desirable, use the [array constructor](#) or the [assignment operator](#).

Note

The pointers passed to this method will be freed using `delete[]`, so they must be created using `new[]`.

9.89.5.23 `virtual void CoinWarmStartBasis::print () const [virtual]`

Prints in readable format (for debug)

9.89.5.24 bool CoinWarmStartBasis::fullBasis () const

Returns true if full basis (for debug)

9.89.5.25 bool CoinWarmStartBasis::fixFullBasis ()

Returns true if full basis and fixes up (for debug)

9.89.6 Friends And Related Function Documentation

9.89.6.1 CoinWarmStartBasis::Status getStatus (const char * array, int i) [related]

Get the status of the specified variable in the given status array.

Definition at line 351 of file CoinWarmStartBasis.hpp.

9.89.6.2 void setStatus (char * array, int i, CoinWarmStartBasis::Status st) [related]

Set the status of the specified variable in the given status array.

Definition at line 360 of file CoinWarmStartBasis.hpp.

9.89.6.3 const char * statusName (CoinWarmStartBasis::Status status) [related]

Generate a print string for a status code.

9.89.7 Member Data Documentation

9.89.7.1 int CoinWarmStartBasis::numStructural_ [protected]

The number of structural variables.

Definition at line 334 of file CoinWarmStartBasis.hpp.

9.89.7.2 int CoinWarmStartBasis::numArtificial_ [protected]

The number of artificial variables.

Definition at line 336 of file CoinWarmStartBasis.hpp.

9.89.7.3 int CoinWarmStartBasis::maxSize_ [protected]

The maximum size (in ints - actually 4*char) (so resize does not need to do new)

Definition at line 338 of file CoinWarmStartBasis.hpp.

9.89.7.4 char* CoinWarmStartBasis::structuralStatus_ [protected]

The status of the structural variables.

Definition at line 340 of file CoinWarmStartBasis.hpp.

9.89.7.5 char* CoinWarmStartBasis::artificialStatus_ [protected]

The status of the artificial variables.

Definition at line 342 of file CoinWarmStartBasis.hpp.

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

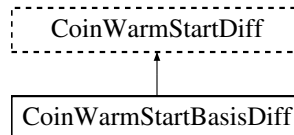
- [/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartBasis.hpp](#)

9.90 CoinWarmStartBasisDiff Class Reference

A 'diff' between two [CoinWarmStartBasis](#) objects.

```
#include <CoinWarmStartBasis.hpp>
```

Inheritance diagram for CoinWarmStartBasisDiff:



Public Member Functions

- virtual [CoinWarmStartDiff](#) * [clone](#) () const
'Virtual constructor'
- virtual [CoinWarmStartBasisDiff](#) & [operator=](#) (const [CoinWarmStartBasisDiff](#) &rhs)
Assignment.
- virtual [~CoinWarmStartBasisDiff](#) ()
Destructor.

Protected Member Functions

- [CoinWarmStartBasisDiff](#) ()
Default constructor.
- [CoinWarmStartBasisDiff](#) (const [CoinWarmStartBasisDiff](#) &cwsbd)
Copy constructor.
- [CoinWarmStartBasisDiff](#) (int size, const unsigned int *const diffNdxs, const unsigned int *const diffVals)
Standard constructor.
- [CoinWarmStartBasisDiff](#) (const [CoinWarmStartBasis](#) *rhs)
Constructor when full is smaller than diff!

Friends

- [CoinWarmStartDiff](#) * [CoinWarmStartBasis::generateDiff](#) (const [CoinWarmStart](#) *const oldCWS) const
- void [CoinWarmStartBasis::applyDiff](#) (const [CoinWarmStartDiff](#) *const diff)

9.90.1 Detailed Description

A 'diff' between two [CoinWarmStartBasis](#) objects.

This class exists in order to hide from the world the details of calculating and representing a 'diff' between two [CoinWarmStartBasis](#) objects. For convenience, assignment, cloning, and deletion are visible to the world, and default and copy constructors are made available to derived classes. Knowledge of the rest of this structure, and of generating and

applying diffs, is restricted to the friend functions [CoinWarmStartBasis::generateDiff\(\)](#) and [CoinWarmStartBasis::applyDiff\(\)](#).

The actual data structure is an unsigned int vector, `#difference_` which starts with indices of changed and then has values starting after `#size_`

Todo This is a pretty generic structure, and vector diff is a pretty generic activity. We should be able to convert this to a template.

Todo Using unsigned int as the data type for the diff vectors might help to contain the damage when this code is inevitably compiled for 64 bit architectures. But the notion of int as 4 bytes is hardwired into [CoinWarmStartBasis](#), so changes are definitely required.

Definition at line 395 of file `CoinWarmStartBasis.hpp`.

9.90.2 Constructor & Destructor Documentation

9.90.2.1 `virtual CoinWarmStartBasisDiff::~CoinWarmStartBasisDiff () [virtual]`

Destructor.

9.90.2.2 `CoinWarmStartBasisDiff::CoinWarmStartBasisDiff () [inline],[protected]`

Default constructor.

This is protected (rather than private) so that derived classes can see it when they make *their* default constructor protected or private.

Definition at line 418 of file `CoinWarmStartBasis.hpp`.

9.90.2.3 `CoinWarmStartBasisDiff::CoinWarmStartBasisDiff (const CoinWarmStartBasisDiff & cwsbd) [protected]`

Copy constructor.

For convenience when copying objects containing [CoinWarmStartBasisDiff](#) objects. But consider whether you should be using [clone\(\)](#) to retain polymorphism.

This is protected (rather than private) so that derived classes can see it when they make *their* copy constructor protected or private.

9.90.2.4 `CoinWarmStartBasisDiff::CoinWarmStartBasisDiff (int sze, const unsigned int *const diffNdxs, const unsigned int *const diffVals) [protected]`

Standard constructor.

9.90.2.5 `CoinWarmStartBasisDiff::CoinWarmStartBasisDiff (const CoinWarmStartBasis * rhs) [protected]`

Constructor when full is smaller than diff!

9.90.3 Member Function Documentation

9.90.3.1 `virtual CoinWarmStartDiff* CoinWarmStartBasisDiff::clone () const [inline],[virtual]`

‘Virtual constructor’

Implements [CoinWarmStartDiff](#).

Definition at line 399 of file `CoinWarmStartBasis.hpp`.

9.90.3.2 `virtual CoinWarmStartBasisDiff& CoinWarmStartBasisDiff::operator= (const CoinWarmStartBasisDiff & rhs)`
`[virtual]`

Assignment.

9.90.4 Friends And Related Function Documentation

9.90.4.1 `CoinWarmStartDiff* CoinWarmStartBasis::generateDiff (const CoinWarmStart *const oldCWS) const`
`[friend]`

9.90.4.2 `void CoinWarmStartBasis::applyDiff (const CoinWarmStartDiff *const diff)` `[friend]`

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

- </home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartBasis.hpp>

9.91 CoinWarmStartDiff Class Reference

Abstract base class for warm start 'diff' objects.

`#include <CoinWarmStart.hpp>`

Inheritance diagram for CoinWarmStartDiff:



Public Member Functions

- `virtual ~CoinWarmStartDiff ()`
Abstract destructor.
- `virtual CoinWarmStartDiff * clone () const =0`
'Virtual constructor'

9.91.1 Detailed Description

Abstract base class for warm start 'diff' objects.

For those types of warm start objects where the notion of a 'diff' makes sense, this virtual base class is provided. As with [CoinWarmStart](#), its sole reason for existence is to make it possible to write solver-independent code.

Definition at line 48 of file [CoinWarmStart.hpp](#).

9.91.2 Constructor & Destructor Documentation

9.91.2.1 `virtual CoinWarmStartDiff::~~CoinWarmStartDiff ()` `[inline],[virtual]`

Abstract destructor.

Definition at line 52 of file [CoinWarmStart.hpp](#).

9.91.3 Member Function Documentation

9.91.3.1 `virtual CoinWarmStartDiff* CoinWarmStartDiff::clone () const` [pure virtual]

‘Virtual constructor’

Implemented in [CoinWarmStartBasisDiff](#), [CoinWarmStartVectorPairDiff< T, U >](#), [CoinWarmStartVectorDiff< T >](#), [CoinWarmStartVectorDiff< double >](#), [CoinWarmStartVectorDiff< U >](#), [CoinWarmStartPrimalDualDiff](#), and [CoinWarmStartDualDiff](#).

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

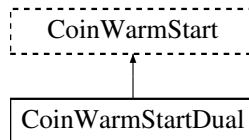
- [/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart.hpp](#)

9.92 CoinWarmStartDual Class Reference

WarmStart information that is only a dual vector.

```
#include <CoinWarmStartDual.hpp>
```

Inheritance diagram for CoinWarmStartDual:



Public Member Functions

- `int size () const`
return the size of the dual vector
- `const double * dual () const`
return a pointer to the array of duals
- `void assignDual (int size, double *&dual)`
Assign the dual vector to be the warmstart information.
- `CoinWarmStartDual ()`
- `CoinWarmStartDual (int size, const double *dual)`
- `CoinWarmStartDual (const CoinWarmStartDual &rhs)`
- `CoinWarmStartDual & operator= (const CoinWarmStartDual &rhs)`
- `virtual CoinWarmStart * clone () const`
‘Virtual constructor’
- `virtual ~CoinWarmStartDual ()`

Dual warm start ‘diff’ methods

- `virtual CoinWarmStartDiff * generateDiff (const CoinWarmStart *const oldCWS) const`
Generate a ‘diff’ that can convert the warm start passed as a parameter to the warm start specified by `this`.
- `virtual void applyDiff (const CoinWarmStartDiff *const cwsdDiff)`
Apply `diff` to this warm start.

9.92.1 Detailed Description

WarmStart information that is only a dual vector.

Definition at line 18 of file CoinWarmStartDual.hpp.

9.92.2 Constructor & Destructor Documentation

9.92.2.1 CoinWarmStartDual::CoinWarmStartDual () [inline]

Definition at line 31 of file CoinWarmStartDual.hpp.

9.92.2.2 CoinWarmStartDual::CoinWarmStartDual (int *size*, const double * *dual*) [inline]

Definition at line 33 of file CoinWarmStartDual.hpp.

9.92.2.3 CoinWarmStartDual::CoinWarmStartDual (const CoinWarmStartDual & *rhs*) [inline]

Definition at line 35 of file CoinWarmStartDual.hpp.

9.92.2.4 virtual CoinWarmStartDual::~CoinWarmStartDual () [inline],[virtual]

Definition at line 49 of file CoinWarmStartDual.hpp.

9.92.3 Member Function Documentation

9.92.3.1 int CoinWarmStartDual::size () const [inline]

return the size of the dual vector

Definition at line 21 of file CoinWarmStartDual.hpp.

9.92.3.2 const double* CoinWarmStartDual::dual () const [inline]

return a pointer to the array of duals

Definition at line 23 of file CoinWarmStartDual.hpp.

9.92.3.3 void CoinWarmStartDual::assignDual (int *size*, double *& *dual*) [inline]

Assign the dual vector to be the warmstart information.

In this method the object assumes ownership of the pointer and upon return "dual" will be a NULL pointer. If copying is desirable use the constructor.

Definition at line 28 of file CoinWarmStartDual.hpp.

9.92.3.4 CoinWarmStartDual& CoinWarmStartDual::operator= (const CoinWarmStartDual & *rhs*) [inline]

Definition at line 37 of file CoinWarmStartDual.hpp.

9.92.3.5 virtual CoinWarmStart* CoinWarmStartDual::clone () const [inline],[virtual]

‘Virtual constructor’

Implements [CoinWarmStart](#).

Definition at line 45 of file CoinWarmStartDual.hpp.

9.92.3.6 `virtual CoinWarmStartDiff* CoinWarmStartDual::generateDiff (const CoinWarmStart *const oldCWS) const`
`[virtual]`

Generate a 'diff' that can convert the warm start passed as a parameter to the warm start specified by `this`.

The capabilities are limited: the basis passed as a parameter can be no larger than the basis pointed to by `this`.

Reimplemented from [CoinWarmStart](#).

9.92.3.7 `virtual void CoinWarmStartDual::applyDiff (const CoinWarmStartDiff *const cwsdDiff)` `[virtual]`

Apply `diff` to this warm start.

Update this warm start by applying `diff`. It's assumed that the allocated capacity of the warm start is sufficiently large.

Reimplemented from [CoinWarmStart](#).

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

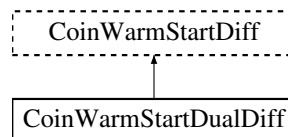
- `/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartDual.hpp`

9.93 CoinWarmStartDualDiff Class Reference

A 'diff' between two [CoinWarmStartDual](#) objects.

`#include <CoinWarmStartDual.hpp>`

Inheritance diagram for `CoinWarmStartDualDiff`:



Public Member Functions

- `virtual CoinWarmStartDiff * clone () const`
'Virtual constructor'
- `virtual CoinWarmStartDualDiff & operator= (const CoinWarmStartDualDiff &rhs)`
Assignment.
- `virtual ~CoinWarmStartDualDiff ()`
Destructor.

Protected Member Functions

- `CoinWarmStartDualDiff ()`
Default constructor.
- `CoinWarmStartDualDiff (const CoinWarmStartDualDiff &rhs)`
Copy constructor.

Friends

- [CoinWarmStartDiff](#) * [CoinWarmStartDual::generateDiff](#) (const [CoinWarmStart](#) *const oldCWS) const
- void [CoinWarmStartDual::applyDiff](#) (const [CoinWarmStartDiff](#) *const diff)

9.93.1 Detailed Description

A ‘diff’ between two [CoinWarmStartDual](#) objects.

This class exists in order to hide from the world the details of calculating and representing a ‘diff’ between two [CoinWarmStartDual](#) objects. For convenience, assignment, cloning, and deletion are visible to the world, and default and copy constructors are made available to derived classes. Knowledge of the rest of this structure, and of generating and applying diffs, is restricted to the friend functions [CoinWarmStartDual::generateDiff\(\)](#) and [CoinWarmStartDual::applyDiff\(\)](#).

The actual data structure is a pair of vectors, `#diffNdxs_` and `#diffVals_`.

Definition at line 101 of file `CoinWarmStartDual.hpp`.

9.93.2 Constructor & Destructor Documentation

9.93.2.1 virtual [CoinWarmStartDualDiff::~~CoinWarmStartDualDiff](#) () [inline], [virtual]

Destructor.

Definition at line 120 of file `CoinWarmStartDual.hpp`.

9.93.2.2 [CoinWarmStartDualDiff::CoinWarmStartDualDiff](#) () [inline], [protected]

Default constructor.

This is protected (rather than private) so that derived classes can see it when they make *their* default constructor protected or private.

Definition at line 130 of file `CoinWarmStartDual.hpp`.

9.93.2.3 [CoinWarmStartDualDiff::CoinWarmStartDualDiff](#) (const [CoinWarmStartDualDiff](#) & rhs) [inline], [protected]

Copy constructor.

For convenience when copying objects containing [CoinWarmStartDualDiff](#) objects. But consider whether you should be using [clone\(\)](#) to retain polymorphism.

This is protected (rather than private) so that derived classes can see it when the make *their* copy constructor protected or private.

Definition at line 142 of file `CoinWarmStartDual.hpp`.

9.93.3 Member Function Documentation

9.93.3.1 virtual [CoinWarmStartDiff](#)* [CoinWarmStartDualDiff::clone](#) () const [inline], [virtual]

‘Virtual constructor’

Implements [CoinWarmStartDiff](#).

Definition at line 105 of file `CoinWarmStartDual.hpp`.

9.93.3.2 `virtual CoinWarmStartDualDiff& CoinWarmStartDualDiff::operator= (const CoinWarmStartDualDiff & rhs)`
`[inline],[virtual]`

Assignment.

Definition at line 111 of file CoinWarmStartDual.hpp.

9.93.4 Friends And Related Function Documentation

9.93.4.1 `CoinWarmStartDiff* CoinWarmStartDual::generateDiff (const CoinWarmStart *const oldCWS) const`
`[friend]`

9.93.4.2 `void CoinWarmStartDual::applyDiff (const CoinWarmStartDiff *const diff)` `[friend]`

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

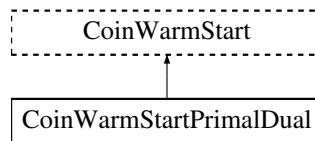
- </home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartDual.hpp>

9.94 CoinWarmStartPrimalDual Class Reference

WarmStart information that is only a dual vector.

```
#include <CoinWarmStartPrimalDual.hpp>
```

Inheritance diagram for CoinWarmStartPrimalDual:



Public Member Functions

- `int dualSize () const`
return the size of the dual vector
- `const double * dual () const`
return a pointer to the array of duals
- `int primalSize () const`
return the size of the primal vector
- `const double * primal () const`
return a pointer to the array of primals
- `void assign (int primalSize, int dualSize, double *&primal, double *&dual)`
Assign the primal/dual vectors to be the warmstart information.
- `CoinWarmStartPrimalDual ()`
- `CoinWarmStartPrimalDual (int primalSize, int dualSize, const double *primal, const double *dual)`
- `CoinWarmStartPrimalDual (const CoinWarmStartPrimalDual &rhs)`
- `CoinWarmStartPrimalDual & operator= (const CoinWarmStartPrimalDual &rhs)`
- `void clear ()`
Clear the data.
- `void swap (CoinWarmStartPrimalDual &rhs)`

- virtual `CoinWarmStart * clone ()` const
'Virtual constructor'
- virtual `~CoinWarmStartPrimalDual ()`

PrimalDual warm start 'diff' methods

- virtual `CoinWarmStartDiff * generateDiff (const CoinWarmStart *const oldCWS)` const
Generate a 'diff' that can convert the warm start passed as a parameter to the warm start specified by `this`.
- virtual `void applyDiff (const CoinWarmStartDiff *const cwsdDiff)`
Apply `diff` to this warm start.

9.94.1 Detailed Description

WarmStart information that is only a dual vector.

Definition at line 18 of file `CoinWarmStartPrimalDual.hpp`.

9.94.2 Constructor & Destructor Documentation

9.94.2.1 `CoinWarmStartPrimalDual::CoinWarmStartPrimalDual ()` [inline]

Definition at line 44 of file `CoinWarmStartPrimalDual.hpp`.

9.94.2.2 `CoinWarmStartPrimalDual::CoinWarmStartPrimalDual (int primalSize, int dualSize, const double * primal, const double * dual)` [inline]

Definition at line 46 of file `CoinWarmStartPrimalDual.hpp`.

9.94.2.3 `CoinWarmStartPrimalDual::CoinWarmStartPrimalDual (const CoinWarmStartPrimalDual & rhs)` [inline]

Definition at line 50 of file `CoinWarmStartPrimalDual.hpp`.

9.94.2.4 `virtual CoinWarmStartPrimalDual::~~CoinWarmStartPrimalDual ()` [inline],[virtual]

Definition at line 83 of file `CoinWarmStartPrimalDual.hpp`.

9.94.3 Member Function Documentation

9.94.3.1 `int CoinWarmStartPrimalDual::dualSize ()` const [inline]

return the size of the dual vector

Definition at line 21 of file `CoinWarmStartPrimalDual.hpp`.

9.94.3.2 `const double* CoinWarmStartPrimalDual::dual ()` const [inline]

return a pointer to the array of duals

Definition at line 23 of file `CoinWarmStartPrimalDual.hpp`.

9.94.3.3 `int CoinWarmStartPrimalDual::primalSize ()` const [inline]

return the size of the primal vector

Definition at line 26 of file `CoinWarmStartPrimalDual.hpp`.

9.94.3.4 `const double* CoinWarmStartPrimalDual::primal () const [inline]`

return a pointer to the array of primals

Definition at line 28 of file CoinWarmStartPrimalDual.hpp.

9.94.3.5 `void CoinWarmStartPrimalDual::assign (int primalSize, int dualSize, double *& primal, double *& dual) [inline]`

Assign the primal/dual vectors to be the warmstart information.

In this method the object assumes ownership of the pointers and upon return `primal` and `dual` will be a NULL pointers. If copying is desirable use the constructor.

NOTE: `primal` and `dual` must have been allocated by `new double[]`, because they will be freed by `delete[]` upon the destruction of this object...

Definition at line 39 of file CoinWarmStartPrimalDual.hpp.

9.94.3.6 `CoinWarmStartPrimalDual& CoinWarmStartPrimalDual::operator= (const CoinWarmStartPrimalDual & rhs) [inline]`

Definition at line 53 of file CoinWarmStartPrimalDual.hpp.

9.94.3.7 `void CoinWarmStartPrimalDual::clear () [inline]`

Clear the data.

Make it appear as if the warmstart was just created using the default constructor.

Definition at line 66 of file CoinWarmStartPrimalDual.hpp.

9.94.3.8 `void CoinWarmStartPrimalDual::swap (CoinWarmStartPrimalDual & rhs) [inline]`

Definition at line 71 of file CoinWarmStartPrimalDual.hpp.

9.94.3.9 `virtual CoinWarmStart* CoinWarmStartPrimalDual::clone () const [inline],[virtual]`

‘Virtual constructor’

Implements [CoinWarmStart](#).

Definition at line 79 of file CoinWarmStartPrimalDual.hpp.

9.94.3.10 `virtual CoinWarmStartDiff* CoinWarmStartPrimalDual::generateDiff (const CoinWarmStart *const oldCWS) const [virtual]`

Generate a ‘diff’ that can convert the warm start passed as a parameter to the warm start specified by `this`.

The capabilities are limited: the basis passed as a parameter can be no larger than the basis pointed to by `this`.

Reimplemented from [CoinWarmStart](#).

9.94.3.11 `virtual void CoinWarmStartPrimalDual::applyDiff (const CoinWarmStartDiff *const cwsdDiff) [virtual]`

Apply `diff` to this warm start.

Update this warm start by applying `diff`. It’s assumed that the allocated capacity of the warm start is sufficiently large.

Reimplemented from [CoinWarmStart](#).

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

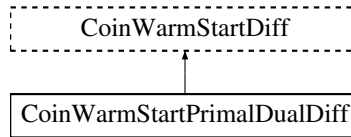
- [/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartPrimalDual.hpp](#)

9.95 CoinWarmStartPrimalDualDiff Class Reference

A 'diff' between two [CoinWarmStartPrimalDual](#) objects.

```
#include <CoinWarmStartPrimalDual.hpp>
```

Inheritance diagram for CoinWarmStartPrimalDualDiff:



Public Member Functions

- virtual [CoinWarmStartDiff](#) * [clone](#) () const
'Virtual constructor'.
- virtual [~CoinWarmStartPrimalDualDiff](#) ()
Destructor.

Protected Member Functions

- [CoinWarmStartPrimalDualDiff](#) ()
Default constructor.
- [CoinWarmStartPrimalDualDiff](#) (const [CoinWarmStartPrimalDualDiff](#) &rhs)
Copy constructor.
- void [clear](#) ()
Clear the data.
- void [swap](#) ([CoinWarmStartPrimalDualDiff](#) &rhs)

Friends

- [CoinWarmStartDiff](#) * [CoinWarmStartPrimalDual::generateDiff](#) (const [CoinWarmStart](#) *const oldCWS) const
- void [CoinWarmStartPrimalDual::applyDiff](#) (const [CoinWarmStartDiff](#) *const diff)

9.95.1 Detailed Description

A 'diff' between two [CoinWarmStartPrimalDual](#) objects.

This class exists in order to hide from the world the details of calculating and representing a 'diff' between two [CoinWarmStartPrimalDual](#) objects. For convenience, assignment, cloning, and deletion are visible to the world, and default and copy constructors are made available to derived classes. Knowledge of the rest of this structure, and of generating and applying diffs, is restricted to the friend functions [CoinWarmStartPrimalDual::generateDiff\(\)](#) and [CoinWarmStartPrimalDual::applyDiff\(\)](#).

The actual data structure is a pair of vectors, `#diffNdxs_` and `#diffVals_`.

Definition at line 142 of file [CoinWarmStartPrimalDual.hpp](#).

9.95.2 Constructor & Destructor Documentation

9.95.2.1 `virtual CoinWarmStartPrimalDualDiff::~~CoinWarmStartPrimalDualDiff () [inline], [virtual]`

Destructor.

Definition at line 159 of file `CoinWarmStartPrimalDual.hpp`.

9.95.2.2 `CoinWarmStartPrimalDualDiff::CoinWarmStartPrimalDualDiff () [inline], [protected]`

Default constructor.

This is protected (rather than private) so that derived classes can see it when they make *their* default constructor protected or private.

Definition at line 169 of file `CoinWarmStartPrimalDual.hpp`.

9.95.2.3 `CoinWarmStartPrimalDualDiff::CoinWarmStartPrimalDualDiff (const CoinWarmStartPrimalDualDiff & rhs) [inline], [protected]`

Copy constructor.

For convenience when copying objects containing `CoinWarmStartPrimalDualDiff` objects. But consider whether you should be using `clone()` to retain polymorphism.

This is protected (rather than private) so that derived classes can see it when they make *their* copy constructor protected or private.

Definition at line 181 of file `CoinWarmStartPrimalDual.hpp`.

9.95.3 Member Function Documentation

9.95.3.1 `virtual CoinWarmStartDiff* CoinWarmStartPrimalDualDiff::clone () const [inline], [virtual]`

‘Virtual constructor’.

To be used when retaining polymorphism is important

Implements `CoinWarmStartDiff`.

Definition at line 153 of file `CoinWarmStartPrimalDual.hpp`.

9.95.3.2 `void CoinWarmStartPrimalDualDiff::clear () [inline], [protected]`

Clear the data.

Make it appear as if the diff was just created using the default constructor.

Definition at line 189 of file `CoinWarmStartPrimalDual.hpp`.

9.95.3.3 `void CoinWarmStartPrimalDualDiff::swap (CoinWarmStartPrimalDualDiff & rhs) [inline], [protected]`

Definition at line 194 of file `CoinWarmStartPrimalDual.hpp`.

9.95.4 Friends And Related Function Documentation

9.95.4.1 `CoinWarmStartDiff* CoinWarmStartPrimalDual::generateDiff (const CoinWarmStart *const oldCWS) const [friend]`

9.95.4.2 `void CoinWarmStartPrimalDual::applyDiff (const CoinWarmStartDiff *const diff) [friend]`

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

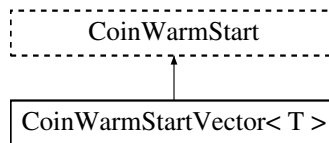
- </home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartPrimalDual.hpp>

9.96 CoinWarmStartVector< T > Class Template Reference

WarmStart information that is only a vector.

```
#include <CoinWarmStartVector.hpp>
```

Inheritance diagram for CoinWarmStartVector< T >:



Public Member Functions

- `int size () const`
return the size of the vector
- `const T * values () const`
return a pointer to the array of vectors
- `void assignVector (int size, T *&vec)`
Assign the vector to be the warmstart information.
- `CoinWarmStartVector ()`
- `CoinWarmStartVector (int size, const T *vec)`
- `CoinWarmStartVector (const CoinWarmStartVector &rhs)`
- `CoinWarmStartVector & operator= (const CoinWarmStartVector &rhs)`
- `void swap (CoinWarmStartVector &rhs)`
- `virtual CoinWarmStart * clone () const`
'Virtual constructor'
- `virtual ~CoinWarmStartVector ()`
- `void clear ()`
Clear the data.

Vector warm start 'diff' methods

- `virtual CoinWarmStartDiff * generateDiff (const CoinWarmStart *const oldCWS) const`
Generate a 'diff' that can convert the warm start passed as a parameter to the warm start specified by this.
- `virtual void applyDiff (const CoinWarmStartDiff *const cwsdDiff)`
Apply diff to this warm start.

Protected Member Functions

- `void gutsOfDestructor ()`
- `void gutsOfCopy (const CoinWarmStartVector< T > &rhs)`

9.96.1 Detailed Description

```
template<typename T> class CoinWarmStartVector< T >
```

WarmStart information that is only a vector.

Definition at line 26 of file CoinWarmStartVector.hpp.

9.96.2 Constructor & Destructor Documentation

9.96.2.1 `template<typename T> CoinWarmStartVector< T >::CoinWarmStartVector () [inline]`

Definition at line 54 of file CoinWarmStartVector.hpp.

9.96.2.2 `template<typename T> CoinWarmStartVector< T >::CoinWarmStartVector (int size, const T * vec) [inline]`

Definition at line 56 of file CoinWarmStartVector.hpp.

9.96.2.3 `template<typename T> CoinWarmStartVector< T >::CoinWarmStartVector (const CoinWarmStartVector< T > & rhs) [inline]`

Definition at line 61 of file CoinWarmStartVector.hpp.

9.96.2.4 `template<typename T> virtual CoinWarmStartVector< T >::~CoinWarmStartVector () [inline], [virtual]`

Definition at line 85 of file CoinWarmStartVector.hpp.

9.96.3 Member Function Documentation

9.96.3.1 `template<typename T> void CoinWarmStartVector< T >::gutsOfDestructor () [inline], [protected]`

Definition at line 29 of file CoinWarmStartVector.hpp.

9.96.3.2 `template<typename T> void CoinWarmStartVector< T >::gutsOfCopy (const CoinWarmStartVector< T > & rhs) [inline], [protected]`

Definition at line 32 of file CoinWarmStartVector.hpp.

9.96.3.3 `template<typename T> int CoinWarmStartVector< T >::size () const [inline]`

return the size of the vector

Definition at line 40 of file CoinWarmStartVector.hpp.

9.96.3.4 `template<typename T> const T* CoinWarmStartVector< T >::values () const [inline]`

return a pointer to the array of vectors

Definition at line 42 of file CoinWarmStartVector.hpp.

9.96.3.5 `template<typename T> void CoinWarmStartVector< T >::assignVector (int size, T *& vec) [inline]`

Assign the vector to be the warmstart information.

In this method the object assumes ownership of the pointer and upon return #vector will be a NULL pointer. If copying is desirable use the constructor.

Definition at line 47 of file CoinWarmStartVector.hpp.

```
9.96.3.6  template<typename T> CoinWarmStartVector& CoinWarmStartVector< T >::operator= ( const
        CoinWarmStartVector< T > & rhs ) [inline]
```

Definition at line 65 of file CoinWarmStartVector.hpp.

```
9.96.3.7  template<typename T> void CoinWarmStartVector< T >::swap ( CoinWarmStartVector< T > & rhs )
        [inline]
```

Definition at line 73 of file CoinWarmStartVector.hpp.

```
9.96.3.8  template<typename T> virtual CoinWarmStart* CoinWarmStartVector< T >::clone ( ) const [inline],
        [virtual]
```

‘Virtual constructor’

Implements [CoinWarmStart](#).

Definition at line 81 of file CoinWarmStartVector.hpp.

```
9.96.3.9  template<typename T> void CoinWarmStartVector< T >::clear ( ) [inline]
```

Clear the data.

Make it appear as if the warmstart was just created using the default constructor.

Definition at line 94 of file CoinWarmStartVector.hpp.

```
9.96.3.10 template<typename T > CoinWarmStartDiff * CoinWarmStartVector< T >::generateDiff ( const
        CoinWarmStart *const oldCWS ) const [virtual]
```

Generate a ‘diff’ that can convert the warm start passed as a parameter to the warm start specified by `this`.

The capabilities are limited: the basis passed as a parameter can be no larger than the basis pointed to by `this`.

Reimplemented from [CoinWarmStart](#).

Definition at line 336 of file CoinWarmStartVector.hpp.

```
9.96.3.11 template<typename T > void CoinWarmStartVector< T >::applyDiff ( const CoinWarmStartDiff *const
        cwsdDiff ) [virtual]
```

Apply `diff` to this warm start.

Update this warm start by applying `diff`. It’s assumed that the allocated capacity of the warm start is sufficiently large.

Reimplemented from [CoinWarmStart](#).

Definition at line 400 of file CoinWarmStartVector.hpp.

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

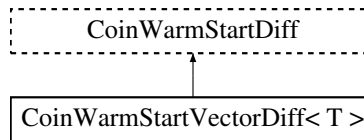
- [/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartVector.hpp](#)

9.97 CoinWarmStartVectorDiff< T > Class Template Reference

A ‘diff’ between two [CoinWarmStartVector](#) objects.

```
#include <CoinWarmStartVector.hpp>
```

Inheritance diagram for CoinWarmStartVectorDiff< T >:



Public Member Functions

- virtual [CoinWarmStartDiff](#) * [clone](#) () const
'Virtual constructor'
- virtual [CoinWarmStartVectorDiff](#) & [operator=](#) (const [CoinWarmStartVectorDiff](#)< T > &rhs)
Assignment.
- virtual [~CoinWarmStartVectorDiff](#) ()
Destructor.
- [void swap](#) ([CoinWarmStartVectorDiff](#) &rhs)
- [CoinWarmStartVectorDiff](#) ()
Default constructor.
- [CoinWarmStartVectorDiff](#) (const [CoinWarmStartVectorDiff](#)< T > &rhs)
Copy constructor.
- [CoinWarmStartVectorDiff](#) (int size, const unsigned int *const diffNdxs, const T *const diffVals)
Standard constructor.
- [void clear](#) ()
Clear the data.

Friends

- [CoinWarmStartDiff](#) * [CoinWarmStartVector](#) (const [CoinWarmStart](#) *const oldCWS) const
- [void CoinWarmStartVector](#) (const [CoinWarmStartDiff](#) *const diff)

9.97.1 Detailed Description

```
template<typename T>class CoinWarmStartVectorDiff< T >
```

A 'diff' between two [CoinWarmStartVector](#) objects.

This class exists in order to hide from the world the details of calculating and representing a 'diff' between two [CoinWarmStartVector](#) objects. For convenience, assignment, cloning, and deletion are visible to the world, and default and copy constructors are made available to derived classes. Knowledge of the rest of this structure, and of generating and applying diffs, is restricted to the friend functions [CoinWarmStartVector::generateDiff\(\)](#) and [CoinWarmStartVector::applyDiff\(\)](#).

The actual data structure is a pair of vectors, #diffNdxs_ and #diffVals_.

Definition at line 151 of file CoinWarmStartVector.hpp.

9.97.2 Constructor & Destructor Documentation

9.97.2.1 `template<typename T> virtual CoinWarmStartVectorDiff< T >::~CoinWarmStartVectorDiff ()`
`[inline], [virtual]`

Destructor.

Definition at line 170 of file `CoinWarmStartVector.hpp`.

9.97.2.2 `template<typename T> CoinWarmStartVectorDiff< T >::CoinWarmStartVectorDiff ()` `[inline]`

Default constructor.

Definition at line 185 of file `CoinWarmStartVector.hpp`.

9.97.2.3 `template<typename T> CoinWarmStartVectorDiff< T >::CoinWarmStartVectorDiff (const`
`CoinWarmStartVectorDiff< T > & rhs)`

Copy constructor.

For convenience when copying objects containing `CoinWarmStartVectorDiff` objects. But consider whether you should be using `clone()` to retain polymorphism.

Definition at line 458 of file `CoinWarmStartVector.hpp`.

9.97.2.4 `template<typename T> CoinWarmStartVectorDiff< T >::CoinWarmStartVectorDiff (int sze, const unsigned int`
`*const diffNdxs, const T *const diffVals)`

Standard constructor.

Definition at line 475 of file `CoinWarmStartVector.hpp`.

9.97.3 Member Function Documentation

9.97.3.1 `template<typename T> virtual CoinWarmStartDiff* CoinWarmStartVectorDiff< T >::clone () const`
`[inline], [virtual]`

‘Virtual constructor’

Implements `CoinWarmStartDiff`.

Definition at line 161 of file `CoinWarmStartVector.hpp`.

9.97.3.2 `template<typename T> CoinWarmStartVectorDiff< T > & CoinWarmStartVectorDiff< T >::operator= (const`
`CoinWarmStartVectorDiff< T > & rhs)` `[virtual]`

Assignment.

Definition at line 432 of file `CoinWarmStartVector.hpp`.

9.97.3.3 `template<typename T> void CoinWarmStartVectorDiff< T >::swap (CoinWarmStartVectorDiff< T > & rhs)`
`[inline]`

Definition at line 175 of file `CoinWarmStartVector.hpp`.

9.97.3.4 `template<typename T> void CoinWarmStartVectorDiff< T >::clear ()` `[inline]`

Clear the data.

Make it appear as if the diff was just created using the default constructor.

Definition at line 204 of file CoinWarmStartVector.hpp.

9.97.4 Friends And Related Function Documentation

9.97.4.1 `template<typename T> CoinWarmStartDiff* CoinWarmStartVector (const CoinWarmStart *const oldCWS) const [friend]`

9.97.4.2 `template<typename T> void CoinWarmStartVector (const CoinWarmStartDiff *const diff) [friend]`

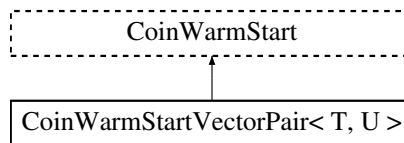
The documentation for this class was generated from the following file:

- </home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartVector.hpp>

9.98 CoinWarmStartVectorPair< T, U > Class Template Reference

```
#include <CoinWarmStartVector.hpp>
```

Inheritance diagram for CoinWarmStartVectorPair< T, U >:



Public Member Functions

- `int size0 () const`
- `int size1 () const`
- `const T * values0 () const`
- `const U * values1 () const`
- `void assignVector0 (int size, T *&vec)`
- `void assignVector1 (int size, U *&vec)`
- `CoinWarmStartVectorPair ()`
- `CoinWarmStartVectorPair (int s0, const T *v0, int s1, const U *v1)`
- `CoinWarmStartVectorPair (const CoinWarmStartVectorPair< T, U > &rhs)`
- `CoinWarmStartVectorPair & operator= (const CoinWarmStartVectorPair< T, U > &rhs)`
- `void swap (CoinWarmStartVectorPair< T, U > &rhs)`
- `virtual CoinWarmStart * clone () const`
'Virtual constructor'
- `virtual ~CoinWarmStartVectorPair ()`
- `void clear ()`
- `virtual CoinWarmStartDiff * generateDiff (const CoinWarmStart *const oldCWS) const`
- `virtual void applyDiff (const CoinWarmStartDiff *const cwsdDiff)`

9.98.1 Detailed Description

```
template<typename T, typename U>class CoinWarmStartVectorPair< T, U >
```

Definition at line 229 of file CoinWarmStartVector.hpp.

9.98.2 Constructor & Destructor Documentation

9.98.2.1 `template<typename T, typename U> CoinWarmStartVectorPair< T, U >::CoinWarmStartVectorPair ()`
`[inline]`

Definition at line 244 of file CoinWarmStartVector.hpp.

9.98.2.2 `template<typename T, typename U> CoinWarmStartVectorPair< T, U >::CoinWarmStartVectorPair (int s0,`
`const T * v0, int s1, const U * v1) [inline]`

Definition at line 245 of file CoinWarmStartVector.hpp.

9.98.2.3 `template<typename T, typename U> CoinWarmStartVectorPair< T, U >::CoinWarmStartVectorPair (const`
`CoinWarmStartVectorPair< T, U > & rhs) [inline]`

Definition at line 248 of file CoinWarmStartVector.hpp.

9.98.2.4 `template<typename T, typename U> virtual CoinWarmStartVectorPair< T, U >::~~CoinWarmStartVectorPair (`
`) [inline], [virtual]`

Definition at line 267 of file CoinWarmStartVector.hpp.

9.98.3 Member Function Documentation

9.98.3.1 `template<typename T, typename U> int CoinWarmStartVectorPair< T, U >::size0 () const [inline]`

Definition at line 236 of file CoinWarmStartVector.hpp.

9.98.3.2 `template<typename T, typename U> int CoinWarmStartVectorPair< T, U >::size1 () const [inline]`

Definition at line 237 of file CoinWarmStartVector.hpp.

9.98.3.3 `template<typename T, typename U> const T* CoinWarmStartVectorPair< T, U >::values0 () const [inline]`

Definition at line 238 of file CoinWarmStartVector.hpp.

9.98.3.4 `template<typename T, typename U> const U* CoinWarmStartVectorPair< T, U >::values1 () const [inline]`

Definition at line 239 of file CoinWarmStartVector.hpp.

9.98.3.5 `template<typename T, typename U> void CoinWarmStartVectorPair< T, U >::assignVector0 (int size, T *& vec)`
`[inline]`

Definition at line 241 of file CoinWarmStartVector.hpp.

9.98.3.6 `template<typename T, typename U> void CoinWarmStartVectorPair< T, U >::assignVector1 (int size, U *& vec)`
`[inline]`

Definition at line 242 of file CoinWarmStartVector.hpp.

9.98.3.7 `template<typename T, typename U> CoinWarmStartVectorPair& CoinWarmStartVectorPair< T, U >::operator=`
`(const CoinWarmStartVectorPair< T, U > & rhs) [inline]`

Definition at line 250 of file CoinWarmStartVector.hpp.

9.98.3.8 `template<typename T, typename U> void CoinWarmStartVectorPair< T, U >::swap (CoinWarmStartVectorPair< T, U > & rhs) [inline]`

Definition at line 258 of file CoinWarmStartVector.hpp.

9.98.3.9 `template<typename T, typename U> virtual CoinWarmStart* CoinWarmStartVectorPair< T, U >::clone () const [inline],[virtual]`

‘Virtual constructor’

Implements [CoinWarmStart](#).

Definition at line 263 of file CoinWarmStartVector.hpp.

9.98.3.10 `template<typename T, typename U> void CoinWarmStartVectorPair< T, U >::clear () [inline]`

Definition at line 269 of file CoinWarmStartVector.hpp.

9.98.3.11 `template<typename T, typename U> virtual CoinWarmStartDiff* CoinWarmStartVectorPair< T, U >::generateDiff (const CoinWarmStart *const oldCWS) const [virtual]`

Reimplemented from [CoinWarmStart](#).

9.98.3.12 `template<typename T, typename U> virtual void CoinWarmStartVectorPair< T, U >::applyDiff (const CoinWarmStartDiff *const cwsdDiff) [virtual]`

Reimplemented from [CoinWarmStart](#).

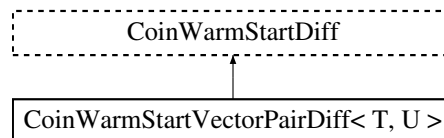
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartVector.hpp](#)

9.99 CoinWarmStartVectorPairDiff< T, U > Class Template Reference

```
#include <CoinWarmStartVector.hpp>
```

Inheritance diagram for CoinWarmStartVectorPairDiff< T, U >:



Public Member Functions

- [CoinWarmStartVectorPairDiff \(\)](#)
- [CoinWarmStartVectorPairDiff \(const \[CoinWarmStartVectorPairDiff\]\(#\)< T, U > &rhs\)](#)
- [virtual ~CoinWarmStartVectorPairDiff \(\)](#)
- [virtual](#)
[CoinWarmStartVectorPairDiff & operator= \(const \[CoinWarmStartVectorPairDiff\]\(#\)< T, U > &rhs\)](#)
- [virtual \[CoinWarmStartDiff\]\(#\) * clone \(\) const](#)
‘Virtual constructor’
- [void swap \(\[CoinWarmStartVectorPairDiff\]\(#\)< T, U > &rhs\)](#)
- [void clear \(\)](#)

Friends

- [CoinWarmStartDiff](#) * [CoinWarmStartVectorPair](#) (const [CoinWarmStart](#) *const oldCWS) const
- void [CoinWarmStartVectorPair](#) (const [CoinWarmStartDiff](#) *const diff)

9.99.1 Detailed Description

```
template<typename T, typename U>class CoinWarmStartVectorPairDiff< T, U >
```

Definition at line 283 of file CoinWarmStartVector.hpp.

9.99.2 Constructor & Destructor Documentation

```
9.99.2.1  template<typename T, typename U> CoinWarmStartVectorPairDiff< T, U >::CoinWarmStartVectorPairDiff ( )
        [inline]
```

Definition at line 295 of file CoinWarmStartVector.hpp.

```
9.99.2.2  template<typename T, typename U> CoinWarmStartVectorPairDiff< T, U >::CoinWarmStartVectorPairDiff (
        const CoinWarmStartVectorPairDiff< T, U > & rhs ) [inline]
```

Definition at line 296 of file CoinWarmStartVector.hpp.

```
9.99.2.3  template<typename T, typename U> virtual CoinWarmStartVectorPairDiff< T, U
        >::~~CoinWarmStartVectorPairDiff ( ) [inline],[virtual]
```

Definition at line 298 of file CoinWarmStartVector.hpp.

9.99.3 Member Function Documentation

```
9.99.3.1  template<typename T, typename U> virtual CoinWarmStartVectorPairDiff& CoinWarmStartVectorPairDiff< T,
        U >::operator= ( const CoinWarmStartVectorPairDiff< T, U > & rhs ) [inline],[virtual]
```

Definition at line 301 of file CoinWarmStartVector.hpp.

```
9.99.3.2  template<typename T, typename U> virtual CoinWarmStartDiff* CoinWarmStartVectorPairDiff< T, U >::clone (
        ) const [inline],[virtual]
```

‘Virtual constructor’

Implements [CoinWarmStartDiff](#).

Definition at line 309 of file CoinWarmStartVector.hpp.

```
9.99.3.3  template<typename T, typename U> void CoinWarmStartVectorPairDiff< T, U >::swap (
        CoinWarmStartVectorPairDiff< T, U > & rhs ) [inline]
```

Definition at line 313 of file CoinWarmStartVector.hpp.

```
9.99.3.4  template<typename T, typename U> void CoinWarmStartVectorPairDiff< T, U >::clear ( ) [inline]
```

Definition at line 318 of file CoinWarmStartVector.hpp.

9.99.4 Friends And Related Function Documentation

9.99.4.1 `template<typename T, typename U> CoinWarmStartDiff* CoinWarmStartVectorPair (const CoinWarmStart *const oldCWS) const` [*friend*]

9.99.4.2 `template<typename T, typename U> void CoinWarmStartVectorPair (const CoinWarmStartDiff *const diff)` [*friend*]

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartVector.hpp](#)

9.100 CoinYacc Class Reference

```
#include <CoinModelUseful.hpp>
```

Public Member Functions

- [CoinYacc \(\)](#)
- [~CoinYacc \(\)](#)

Public Attributes

- [symrec * symtable](#)
- [char * symbuf](#)
- [int length](#)
- [double unsetValue](#)

9.100.1 Detailed Description

Definition at line 151 of file `CoinModelUseful.hpp`.

9.100.2 Constructor & Destructor Documentation

9.100.2.1 `CoinYacc::CoinYacc ()` [*inline*]

Definition at line 157 of file `CoinModelUseful.hpp`.

9.100.2.2 `CoinYacc::~~CoinYacc ()` [*inline*]

Definition at line 158 of file `CoinModelUseful.hpp`.

9.100.3 Member Data Documentation

9.100.3.1 `symrec* CoinYacc::symtable`

Definition at line 174 of file `CoinModelUseful.hpp`.

9.100.3.2 `char* CoinYacc::symbuf`

Definition at line 175 of file `CoinModelUseful.hpp`.

9.100.3.3 int CoinYacc::length

Definition at line 176 of file CoinModelUseful.hpp.

9.100.3.4 double CoinYacc::unsetValue

Definition at line 177 of file CoinModelUseful.hpp.

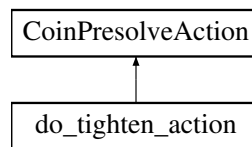
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/CoinUtils/src/[CoinModelUseful.hpp](#)

9.101 do_tighten_action Class Reference

```
#include <CoinPresolveTighten.hpp>
```

Inheritance diagram for do_tighten_action:



Public Member Functions

- const char * [name](#) () const
A name for debug printing.
- void [postsolve](#) ([CoinPostsolveMatrix](#) *prob) const
Apply the postsolve transformation for this particular presolve action.
- virtual [~do_tighten_action](#) ()

Static Public Member Functions

- static const [CoinPresolveAction](#) * [presolve](#) ([CoinPresolveMatrix](#) *prob, const [CoinPresolveAction](#) *next)

Additional Inherited Members

9.101.1 Detailed Description

Definition at line 19 of file CoinPresolveTighten.hpp.

9.101.2 Constructor & Destructor Documentation

9.101.2.1 virtual do_tighten_action::~do_tighten_action () [virtual]

9.101.3 Member Function Documentation

9.101.3.1 const char* do_tighten_action::name () const [virtual]

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

9.101.3.2 `static const CoinPresolveAction* do_tighten_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next) [static]`

9.101.3.3 `void do_tighten_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

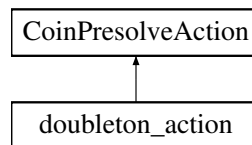
- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTighten.hpp`

9.102 doubleton_action Class Reference

Solve $ax+by=c$ for y and substitute y out of the problem.

```
#include <CoinPresolveDoubleton.hpp>
```

Inheritance diagram for doubleton_action:



Classes

- struct [action](#)

Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~doubleton_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *, const CoinPresolveAction *next)`

Public Attributes

- `const int nactions_`
- `const action *const actions_`

9.102.1 Detailed Description

Solve $ax+by=c$ for y and substitute y out of the problem.

This moves the bounds information for y onto x , making y free and allowing us to substitute it away.

```

a x + b y = c
l1 <= x <= u1
l2 <= y <= u2 ==>

l2 <= (c - a x) / b <= u2
b/-a > 0 ==> (b l2 - c) / -a <= x <= (b u2 - c) / -a
b/-a < 0 ==> (b u2 - c) / -a <= x <= (b l2 - c) / -a

```

Definition at line 26 of file `CoinPresolveDoubleton.hpp`.

9.102.2 Constructor & Destructor Documentation

9.102.2.1 `virtual doubleton_action::~doubleton_action () [virtual]`

9.102.3 Member Function Documentation

9.102.3.1 `const char* doubleton_action::name () const [inline],[virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

Definition at line 62 of file `CoinPresolveDoubleton.hpp`.

9.102.3.2 `static const CoinPresolveAction* doubleton_action::presolve (CoinPresolveMatrix *, const CoinPresolveAction * next) [static]`

9.102.3.3 `void doubleton_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

9.102.4 Member Data Documentation

9.102.4.1 `const int doubleton_action::nactions_`

Definition at line 50 of file `CoinPresolveDoubleton.hpp`.

9.102.4.2 `const action* const doubleton_action::actions_`

Definition at line 51 of file `CoinPresolveDoubleton.hpp`.

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

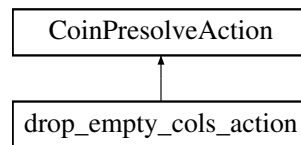
- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDoubleton.hpp`

9.103 drop_empty_cols_action Class Reference

Physically removes empty columns in presolve, and reinserts empty columns in postsolve.

```
#include <CoinPresolveEmpty.hpp>
```

Inheritance diagram for drop_empty_cols_action:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~drop_empty_cols_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *, const int *ecols, int necols, const CoinPresolveAction *)`
- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`

Additional Inherited Members

9.103.1 Detailed Description

Physically removes empty columns in presolve, and reinserts empty columns in postsolve.

Physical removal of rows and columns should be the last activities performed during presolve. Do them exactly once. The row-major matrix is **not** maintained by this transform.

To physically drop the columns, `CoinPrePostsolveMatrix::mcstrt_` and `CoinPrePostsolveMatrix::hincol_` are compressed, along with column bounds, objective, and (if present) the column portions of the solution. This renumbers the columns. `drop_empty_cols_action::presolve` will reconstruct `CoinPresolveMatrix::clink_`.

Todo Confirm correct behaviour with solution in presolve.

Definition at line 34 of file `CoinPresolveEmpty.hpp`.

9.103.2 Constructor & Destructor Documentation

9.103.2.1 `virtual drop_empty_cols_action::~drop_empty_cols_action () [inline], [virtual]`

Definition at line 68 of file `CoinPresolveEmpty.hpp`.

9.103.3 Member Function Documentation

9.103.3.1 `const char* drop_empty_cols_action::name () const` `[inline],[virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

Definition at line 56 of file `CoinPresolveEmpty.hpp`.

9.103.3.2 `static const CoinPresolveAction* drop_empty_cols_action::presolve (CoinPresolveMatrix *, const int * ecol, int necol, const CoinPresolveAction *)` `[static]`

9.103.3.3 `static const CoinPresolveAction* drop_empty_cols_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next)` `[static]`

9.103.3.4 `void drop_empty_cols_action::postsolve (CoinPostsolveMatrix * prob) const` `[virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

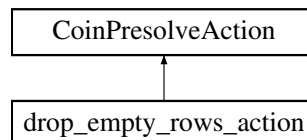
- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveEmpty.hpp`

9.104 drop_empty_rows_action Class Reference

Physically removes empty rows in presolve, and reinserts empty rows in postsolve.

```
#include <CoinPresolveEmpty.hpp>
```

Inheritance diagram for `drop_empty_rows_action`:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~drop_empty_rows_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`

Additional Inherited Members

9.104.1 Detailed Description

Physically removes empty rows in presolve, and reinserts empty rows in postsolve.

Physical removal of rows and columns should be the last activities performed during presolve. Do them exactly once. The row-major matrix is **not** maintained by this transform.

To physically drop the rows, the rows are renumbered, excluding empty rows. This involves rewriting [CoinPrePostsolveMatrix::hrow_](#) and compressing the row bounds and (if present) the row portions of the solution.

Todo Confirm behaviour when a solution is present in presolve.

Definition at line 86 of file `CoinPresolveEmpty.hpp`.

9.104.2 Constructor & Destructor Documentation

9.104.2.1 `virtual drop_empty_rows_action::~drop_empty_rows_action () [inline],[virtual]`

Definition at line 113 of file `CoinPresolveEmpty.hpp`.

9.104.3 Member Function Documentation

9.104.3.1 `const char* drop_empty_rows_action::name () const [inline],[virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

Definition at line 106 of file `CoinPresolveEmpty.hpp`.

9.104.3.2 `static const CoinPresolveAction* drop_empty_rows_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next) [static]`

9.104.3.3 `void drop_empty_rows_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

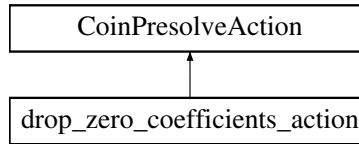
- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveEmpty.hpp`

9.105 drop_zero_coefficients_action Class Reference

Removal of explicit zeros.

```
#include <CoinPresolveZeros.hpp>
```

Inheritance diagram for `drop_zero_coefficients_action`:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~drop_zero_coefficients_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, int *checkcols, int ncheckcols, const CoinPresolveAction *next)`

Additional Inherited Members

9.105.1 Detailed Description

Removal of explicit zeros.

The presolve action for this class removes explicit zeros from the constraint matrix. The postsolve action puts them back.

Definition at line 32 of file `CoinPresolveZeros.hpp`.

9.105.2 Constructor & Destructor Documentation

9.105.2.1 `virtual drop_zero_coefficients_action::~~drop_zero_coefficients_action () [inline],[virtual]`

Definition at line 54 of file `CoinPresolveZeros.hpp`.

9.105.3 Member Function Documentation

9.105.3.1 `const char* drop_zero_coefficients_action::name () const [inline],[virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

Definition at line 45 of file `CoinPresolveZeros.hpp`.

9.105.3.2 `static const CoinPresolveAction* drop_zero_coefficients_action::presolve (CoinPresolveMatrix * prob, int * checkcols, int ncheckcols, const CoinPresolveAction * next) [static]`

9.105.3.3 `void drop_zero_coefficients_action::postsolve (CoinPostsolveMatrix * prob) const` [virtual]

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveZeros.hpp](#)

9.106 dropped_zero Struct Reference

Tracking information for an explicit zero coefficient.

```
#include <CoinPresolveZeros.hpp>
```

Public Attributes

- `int row`
- `int col`

9.106.1 Detailed Description

Tracking information for an explicit zero coefficient.

Todo Why isn't this a nested class in [drop_zero_coefficients_action](#)? That would match the structure of other presolve classes.

Definition at line 22 of file [CoinPresolveZeros.hpp](#).

9.106.2 Member Data Documentation

9.106.2.1 `int dropped_zero::row`

Definition at line 23 of file [CoinPresolveZeros.hpp](#).

9.106.2.2 `int dropped_zero::col`

Definition at line 24 of file [CoinPresolveZeros.hpp](#).

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

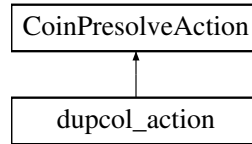
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveZeros.hpp](#)

9.107 dupcol_action Class Reference

Detect and remove duplicate columns.

```
#include <CoinPresolveDupcol.hpp>
```

Inheritance diagram for `dupcol_action`:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~dupcol_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`

Additional Inherited Members

9.107.1 Detailed Description

Detect and remove duplicate columns.

The general technique is to sum the coefficients $a_{*,j}$ of each column. Columns with identical sums are duplicates. The obvious problem is that, e.g., $[1\ 0\ 1\ 0]$ and $[0\ 1\ 0\ 1]$ both add to 2. To minimize the chances of false positives, the coefficients of each row are multiplied by a random number r_i , so that we sum $r_i * a_{ij}$.

Candidate columns are checked to confirm they are identical. Where the columns have the same objective coefficient, the two are combined. If the columns have different objective coefficients, complications ensue. In order to remove the duplicate, it must be possible to fix the variable at a bound.

Definition at line 32 of file `CoinPresolveDupcol.hpp`.

9.107.2 Constructor & Destructor Documentation

9.107.2.1 `virtual dupcol_action::~~dupcol_action () [virtual]`

9.107.3 Member Function Documentation

9.107.3.1 `const char* dupcol_action::name () const [virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements `CoinPresolveAction`.

9.107.3.2 `static const CoinPresolveAction* dupcol_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next) [static]`

9.107.3.3 `void dupcol_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

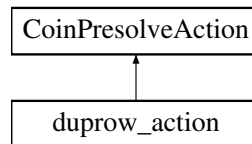
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDupcol.hpp](#)

9.108 duprow_action Class Reference

Detect and remove duplicate rows.

```
#include <CoinPresolveDupcol.hpp>
```

Inheritance diagram for duprow_action:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`

Additional Inherited Members

9.108.1 Detailed Description

Detect and remove duplicate rows.

The algorithm to detect duplicate rows is as outlined for [dupcol_action](#).

If the feasible interval for one constraint is strictly contained in the other, the tighter (contained) constraint is kept. If the feasible intervals are disjoint, the problem is infeasible. If the feasible intervals overlap, both constraints are kept.

[duprow_action](#) is definitely a work in progress; [postsolve](#) is unimplemented. This doesn't matter as it uses `useless_`-constraint.

Definition at line 87 of file `CoinPresolveDupcol.hpp`.

9.108.2 Member Function Documentation

9.108.2.1 `const char* duprow_action::name () const` [virtual]

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

9.108.2.2 `static const CoinPresolveAction* duprow_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next) [static]`

9.108.2.3 `void duprow_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDupcol.hpp](#)

9.109 EKKHlink Struct Reference

This deals with Factorization and Updates This is ripped off from OSL!!!!!!!!!!

```
#include <CoinOslFactorization.hpp>
```

Public Attributes

- int [suc](#)
- int [pre](#)

9.109.1 Detailed Description

This deals with Factorization and Updates This is ripped off from OSL!!!!!!!!!!

I am assuming that 32 bits is enough for number of rows or columns, but CoinBigIndex may be redefined to get 64 bits.

Definition at line 28 of file CoinOslFactorization.hpp.

9.109.2 Member Data Documentation

9.109.2.1 int EKKHlink::suc

Definition at line 28 of file CoinOslFactorization.hpp.

9.109.2.2 int EKKHlink::pre

Definition at line 28 of file CoinOslFactorization.hpp.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinOslFactorization.hpp](#)

9.110 FactorPointers Class Reference

pointers used during factorization

```
#include <CoinSimpFactorization.hpp>
```

Public Member Functions

- [FactorPointers](#) (int numRows, int numCols, int *UrowLengths_, int *UcolLengths_)
- [~FactorPointers](#) ()

Public Attributes

- double * [rowMax](#)
- int * [firstRowKnonzeros](#)
- int * [prevRow](#)
- int * [nextRow](#)
- int * [firstColKnonzeros](#)
- int * [prevColumn](#)
- int * [nextColumn](#)
- int * [newCols](#)

9.110.1 Detailed Description

pointers used during factorization

Definition at line 22 of file CoinSimpFactorization.hpp.

9.110.2 Constructor & Destructor Documentation

9.110.2.1 [FactorPointers::FactorPointers](#) (int *numRows*, int *numCols*, int * *UrowLengths_*, int * *UcolLengths_*)

9.110.2.2 [FactorPointers::~~FactorPointers](#) ()

9.110.3 Member Data Documentation

9.110.3.1 [double*](#) [FactorPointers::rowMax](#)

Definition at line 24 of file CoinSimpFactorization.hpp.

9.110.3.2 [int*](#) [FactorPointers::firstRowKnonzeros](#)

Definition at line 25 of file CoinSimpFactorization.hpp.

9.110.3.3 [int*](#) [FactorPointers::prevRow](#)

Definition at line 26 of file CoinSimpFactorization.hpp.

9.110.3.4 [int*](#) [FactorPointers::nextRow](#)

Definition at line 27 of file CoinSimpFactorization.hpp.

9.110.3.5 [int*](#) [FactorPointers::firstColKnonzeros](#)

Definition at line 28 of file CoinSimpFactorization.hpp.

9.110.3.6 [int*](#) [FactorPointers::prevColumn](#)

Definition at line 29 of file CoinSimpFactorization.hpp.

9.110.3.7 `int* FactorPointers::nextColumn`

Definition at line 30 of file `CoinSimpFactorization.hpp`.

9.110.3.8 `int* FactorPointers::newCols`

Definition at line 31 of file `CoinSimpFactorization.hpp`.

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

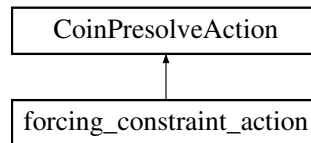
- `/home/ted/COIN/trunk/CoinUtils/src/CoinSimpFactorization.hpp`

9.111 `forcing_constraint_action` Class Reference

Detect and process forcing constraints and useless constraints.

```
#include <CoinPresolveForcing.hpp>
```

Inheritance diagram for `forcing_constraint_action`:



Classes

- struct [action](#)

Public Member Functions

- [forcing_constraint_action](#) (int nactions, const [action](#) *actions, const [CoinPresolveAction](#) *next)
- const char * [name](#) () const
A name for debug printing.
- void [postsolve](#) ([CoinPostsolveMatrix](#) *prob) const
Apply the postsolve transformation for this particular presolve action.
- virtual [~forcing_constraint_action](#) ()

Static Public Member Functions

- static const [CoinPresolveAction](#) * [presolve](#) ([CoinPresolveMatrix](#) *prob, const [CoinPresolveAction](#) *next)

Additional Inherited Members

9.111.1 Detailed Description

Detect and process forcing constraints and useless constraints.

A constraint is useless if the bounds on the variables prevent the constraint from ever being violated.

A constraint is a forcing constraint if the bounds on the constraint force the value of an involved variable to one of its bounds. A constraint can force more than one variable.

Definition at line 27 of file CoinPresolveForcing.hpp.

9.111.2 Constructor & Destructor Documentation

9.111.2.1 `forcing_constraint_action::forcing_constraint_action (int nactions, const action * actions, const CoinPresolveAction * next) [inline]`

Definition at line 45 of file CoinPresolveForcing.hpp.

9.111.2.2 `virtual forcing_constraint_action::~~forcing_constraint_action () [virtual]`

9.111.3 Member Function Documentation

9.111.3.1 `const char* forcing_constraint_action::name () const [virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

9.111.3.2 `static const CoinPresolveAction* forcing_constraint_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next) [static]`

9.111.3.3 `void forcing_constraint_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

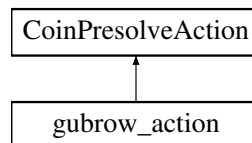
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveForcing.hpp](#)

9.112 gubrow_action Class Reference

Detect and remove entries whose sum is known.

```
#include <CoinPresolveDupcol.hpp>
```

Inheritance diagram for gubrow_action:



Public Member Functions

- `const char * name () const`
A name for debug printing.

- `void postsolve (CoinPostsolveMatrix *prob) const`

Apply the postsolve transformation for this particular presolve action.

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`

Additional Inherited Members

9.112.1 Detailed Description

Detect and remove entries whose sum is known.

If we have an equality row where all entries same then For other rows where all entries for that equality row are same then we can delete entries and modify rhs `gubrow_action` is definitely a work in progress; `postsolve` is unimplemented.

Definition at line 125 of file `CoinPresolveDupcol.hpp`.

9.112.2 Member Function Documentation

9.112.2.1 `const char* gubrow_action::name () const` [virtual]

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements `CoinPresolveAction`.

9.112.2.2 `static const CoinPresolveAction* gubrow_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next)` [static]

9.112.2.3 `void gubrow_action::postsolve (CoinPostsolveMatrix * prob) const` [virtual]

Apply the postsolve transformation for this particular presolve action.

Implements `CoinPresolveAction`.

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

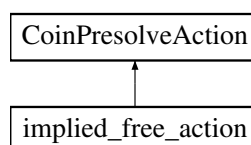
- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDupcol.hpp`

9.113 `implied_free_action` Class Reference

Detect and process implied free variables.

```
#include <CoinPresolveImpliedFree.hpp>
```

Inheritance diagram for `implied_free_action`:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~implied_free_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const CoinPresolveAction *next, int &fillLevel)`

Additional Inherited Members

9.113.1 Detailed Description

Detect and process implied free variables.

Consider a singleton variable x (i.e., a variable involved in only one constraint). Suppose that the bounds on that constraint, combined with the bounds on the other variables involved in the constraint, are such that even the worst case values of the other variables still imply bounds for x which are tighter than the variable's original bounds. Since x can never reach its upper or lower bounds, it is an implied free variable. Both x and the constraint can be deleted from the problem.

A similar transform for the case where the variable is not a natural column singleton is handled by [subst_constraint_action](#).

Definition at line 29 of file `CoinPresolveImpliedFree.hpp`.

9.113.2 Constructor & Destructor Documentation

9.113.2.1 `virtual implied_free_action::~implied_free_action () [virtual]`

9.113.3 Member Function Documentation

9.113.3.1 `const char* implied_free_action::name () const [virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

9.113.3.2 `static const CoinPresolveAction* implied_free_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next, int & fillLevel) [static]`

9.113.3.3 `void implied_free_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

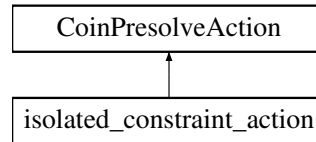
The documentation for this class was generated from the following file:

- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveImpliedFree.hpp`

9.114 isolated_constraint_action Class Reference

```
#include <CoinPresolveIsolated.hpp>
```

Inheritance diagram for `isolated_constraint_action`:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~isolated_constraint_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, int row, const CoinPresolveAction *next)`

Additional Inherited Members

9.114.1 Detailed Description

Definition at line 11 of file `CoinPresolveIsolated.hpp`.

9.114.2 Constructor & Destructor Documentation

9.114.2.1 `virtual isolated_constraint_action::~isolated_constraint_action () [virtual]`

9.114.3 Member Function Documentation

9.114.3.1 `const char* isolated_constraint_action::name () const [virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

9.114.3.2 `static const CoinPresolveAction* isolated_constraint_action::presolve (CoinPresolveMatrix * prob, int row, const CoinPresolveAction * next) [static]`

9.114.3.3 `void isolated_constraint_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

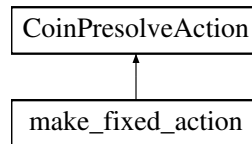
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveIsolated.hpp](#)

9.115 `make_fixed_action` Class Reference

Fix a variable at a specified bound.

```
#include <CoinPresolveFixed.hpp>
```

Inheritance diagram for `make_fixed_action`:



Public Member Functions

- `const char * name () const`
Returns string "make_fixed_action".
- `void postsolve (CoinPostsolveMatrix *prob) const`
Postsolve (unfix variables)
- `virtual ~make_fixed_action ()`
Destructor.

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, int *fcols, int nfc, bool fix_to_lower, const CoinPresolveAction *next)`
Perform actions to fix variables and return postsolve object.

Related Functions

(Note that these are not member functions.)

- `const CoinPresolveAction * make_fixed (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`
Scan variables and fix any with equal bounds.
- `void transferCosts (CoinPresolveMatrix *prob)`
Transfer costs from singleton variables.

Additional Inherited Members

9.115.1 Detailed Description

Fix a variable at a specified bound.

Implements the action of fixing a variable by forcing both bounds to the same value and forcing the value of the variable to match.

If the bounds are already equal, and the value of the variable is already correct, consider [remove_fixed_action](#).

Definition at line 95 of file `CoinPresolveFixed.hpp`.

9.115.2 Constructor & Destructor Documentation

9.115.2.1 `virtual make_fixed_action::~~make_fixed_action () [inline], [virtual]`

Destructor.

Definition at line 153 of file `CoinPresolveFixed.hpp`.

9.115.3 Member Function Documentation

9.115.3.1 `const char* make_fixed_action::name () const [virtual]`

Returns string "make_fixed_action".

Implements [CoinPresolveAction](#).

9.115.3.2 `static const CoinPresolveAction* make_fixed_action::presolve (CoinPresolveMatrix * prob, int * fcols, int nfcols, bool fix_to_lower, const CoinPresolveAction * next) [static]`

Perform actions to fix variables and return postsolve object.

For each specified variable (`nfcols`, `fcols`), fix the variable to the specified bound (`fix_to_lower`) by setting the variable's bounds to be equal in `prob`. Create a postsolve object, link it at the head of the list of postsolve objects (`next`), and return the object.

9.115.3.3 `void make_fixed_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Postsolve (unfix variables)

Back out the variables fixed by the presolve side of this object.

Implements [CoinPresolveAction](#).

9.115.4 Friends And Related Function Documentation

9.115.4.1 `const CoinPresolveAction * make_fixed (CoinPresolveMatrix * prob, const CoinPresolveAction * next) [related]`

Scan variables and fix any with equal bounds.

A front end to collect a list of columns with equal bounds and hand them to [make_fixed_action::presolve\(\)](#) for processing.

9.115.4.2 `void transferCosts (CoinPresolveMatrix * prob) [related]`

Transfer costs from singleton variables.

Transfers costs from singleton variables in equalities onto the other variables. Will also transfer costs from one integer variable to other integer variables with zero cost if there's a net gain in integer variables with non-zero cost.

The relation to [make_fixed_action](#) is tenuous, but this transform should be attempted before the initial round of variable fixing.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveFixed.hpp](#)

9.116 presolvehlink Class Reference

Links to aid in packed matrix modification.

```
#include <CoinPresolveMatrix.hpp>
```

Public Attributes

- int [pre](#)
- int [suc](#)

Related Functions

(Note that these are not member functions.)

- [void PRESOLVE_REMOVE_LINK](#) ([presolvehlink](#) *link, int i)
unlink vector i
- [void PRESOLVE_INSERT_LINK](#) ([presolvehlink](#) *link, int i, int j)
insert vector i after vector j
- [void PRESOLVE_MOVE_LINK](#) ([presolvehlink](#) *link, int i, int j)
relink vector j in place of vector i

9.116.1 Detailed Description

Links to aid in packed matrix modification.

Currently, the matrices held by the [CoinPrePostsolveMatrix](#) and [CoinPresolveMatrix](#) objects are represented in the same way as a [CoinPackedMatrix](#). In the course of presolve and postsolve transforms, it will happen that a major-dimension vector needs to increase in size. In order to check whether there is enough room to add another coefficient in place, it helps to know the next vector (in memory order) in the bulk storage area. To do that, a linked list of major-dimension vectors is maintained; the "pre" and "suc" fields give the previous and next vector, in memory order (that is, the vector whose mcstrt_ or mrstrt_ entry is next smaller or larger).

Consider a column-major matrix with ncols columns. By definition, presolvehlink[ncols].pre points to the column in the last occupied position of the bulk storage arrays. There is no easy way to find the column which occupies the first position (there is no presolvehlink[-1] to consult). If the column that initially occupies the first position is moved for expansion, there is no way to reclaim the space until the bulk storage is compacted. The same holds for the last and first rows of a row-major matrix, of course.

Definition at line 738 of file CoinPresolveMatrix.hpp.

9.116.2 Friends And Related Function Documentation

9.116.2.1 void PRESOLVE_REMOVE_LINK (presolvehlink * link, int i) [related]

unlink vector i

Remove vector i from the ordering.

Definition at line 750 of file CoinPresolveMatrix.hpp.

9.116.2.2 void PRESOLVE_INSERT_LINK (presolvehlink * link, int i, int j) [related]

insert vector i after vector j

Insert vector i between j and j.suc.

Definition at line 768 of file CoinPresolveMatrix.hpp.

9.116.2.3 void PRESOLVE_MOVE_LINK (presolvehlink * link, int i, int j) [related]

relink vector j in place of vector i

Replace vector i in the ordering with vector j. This is equivalent to

```
int pre = link[i].pre;
PRESOLVE_REMOVE_LINK(link, i);
PRESOLVE_INSERT_LINK(link, j, pre);
```

But, this routine will work even if i happens to be first in the order.

Definition at line 790 of file CoinPresolveMatrix.hpp.

9.116.3 Member Data Documentation

9.116.3.1 int presolvehlink::pre

Definition at line 740 of file CoinPresolveMatrix.hpp.

9.116.3.2 int presolvehlink::suc

Definition at line 740 of file CoinPresolveMatrix.hpp.

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

- /home/ted/COIN/trunk/CoinUtils/src/[CoinPresolveMatrix.hpp](#)

9.117 Coin::ReferencedObject Class Reference

[ReferencedObject](#) class.

```
#include <CoinSmartPtr.hpp>
```

Public Member Functions

- [ReferencedObject](#) ()
- virtual [~ReferencedObject](#) ()
- int [ReferenceCount](#) () const
- void [AddRef](#) () const
- void [ReleaseRef](#) () const

9.117.1 Detailed Description

[ReferencedObject](#) class.

This is part of the implementation of an intrusive smart pointer design. This class stores the reference count of all the smart pointers that currently reference it. See the documentation for the [SmartPtr](#) class for more details.

A [SmartPtr](#) behaves much like a raw pointer, but manages the lifetime of an object, deleting the object automatically. This class implements a reference-counting, intrusive smart pointer design, where all objects pointed to must inherit off of [ReferencedObject](#), which stores the reference count. Although this is intrusive (native types and externally authored classes require wrappers to be referenced by smart pointers), it is a safer design. A more detailed discussion of these issues follows after the usage information.

Usage Example: Note: to use the [SmartPtr](#), all objects to which you point MUST inherit off of [ReferencedObject](#).

```
*
* In MyClass.hpp...
*
* #include "CoinSmartPtr.hpp"
*
*
* class MyClass : public Coin::ReferencedObject // must derive from ReferencedObject
* {
*     ...
* }
*
* In my_usage.cpp...
*
* #include "CoinSmartPtr.hpp"
* #include "MyClass.hpp"
*
* void func(AnyObject& obj)
* {
*     Coin::SmartPtr<MyClass> ptr_to_myclass = new MyClass(...);
*     // ptr_to_myclass now points to a new MyClass,
*     // and the reference count is 1
*
*     ...
*
*     obj.SetMyClass(ptr_to_myclass);
*     // Here, let's assume that AnyObject uses a
*     // SmartPtr<MyClass> internally here.
*     // Now, both ptr_to_myclass and the internal
*     // SmartPtr in obj point to the same MyClass object
*     // and its reference count is 2.
*
*     ...
*
*     // No need to delete ptr_to_myclass, this
*     // will be done automatically when the
*     // reference count drops to zero.
*
* }
```

Other Notes: The [SmartPtr](#) implements both dereference operators `->` & `*`. The [SmartPtr](#) does NOT implement a conversion operator to the raw pointer. Use the `GetRawPtr()` method when this is necessary. Make sure that the raw pointer is NOT deleted. The [SmartPtr](#) implements the comparison operators `==` & `!=` for a variety of types. Use these instead of

```
*     if (GetRawPtr(smr_ptr) == ptr) // Don't use this
*
```

[SmartPtr](#)'s, as currently implemented, do NOT handle circular references. For example: consider a higher level object using [SmartPtr](#)s to point to A and B, but A and B also point to each other (i.e. A has a [SmartPtr](#) to B and B has a [SmartPtr](#) to A). In this scenario, when the higher level object is finished with A and B, their reference counts will never drop to zero (since they reference each other) and they will not be deleted. This can be detected by memory leak tools like valgrind. If the circular reference is necessary, the problem can be overcome by a number of techniques:

1) A and B can have a method that "releases" each other, that is they set their internal [SmartPtr](#)s to NULL.

```

*         void AClass::ReleaseCircularReferences()
*         {
*             smart_ptr_to_B = NULL;
*         }
*

```

Then, the higher level class can call these methods before it is done using A & B.

2) Raw pointers can be used in A and B to reference each other. Here, an implicit assumption is made that the lifetime is controlled by the higher level object and that A and B will both exist in a controlled manner. Although this seems dangerous, in many situations, this type of referencing is very controlled and this is reasonably safe.

3) This [SmartPtr](#) class could be redesigned with the Weak/Strong design concept. Here, the [SmartPtr](#) is identified as being Strong (controls lifetime of the object) or Weak (merely referencing the object). The Strong [SmartPtr](#) increments (and decrements) the reference count in [ReferencedObject](#) but the Weak [SmartPtr](#) does not. In the example above, the higher level object would have Strong SmartPtrs to A and B, but A and B would have Weak SmartPtrs to each other. Then, when the higher level object was done with A and B, they would be deleted. The Weak SmartPtrs in A and B would not decrement the reference count and would, of course, not delete the object. This idea is very similar to item (2), where it is implied that the sequence of events is controlled such that A and B will not call anything using their pointers following the higher level delete (i.e. in their destructors!). This is somehow safer, however, because code can be written (however expensive) to perform run-time detection of this situation. For example, the [ReferencedObject](#) could store pointers to all Weak SmartPtrs that are referencing it and, in its destructor, tell these pointers that it is dying. They could then set themselves to NULL, or set an internal flag to detect usage past this point.

Comments on Non-Intrusive Design: In a non-intrusive design, the reference count is stored somewhere other than the object being referenced. This means, unless the reference counting pointer is the first referencer, it must get a pointer to the referenced object from another smart pointer (so it has access to the reference count location). In this non-intrusive design, if we are pointing to an object with a smart pointer (or a number of smart pointers), and we then give another smart pointer the address through a RAW pointer, we will have two independent, AND INCORRECT, reference counts. To avoid this pitfall, we use an intrusive reference counting technique where the reference count is stored in the object being referenced.

Definition at line 157 of file CoinSmartPtr.hpp.

9.117.2 Constructor & Destructor Documentation

9.117.2.1 Coin::ReferencedObject::ReferencedObject () [inline]

Definition at line 159 of file CoinSmartPtr.hpp.

9.117.2.2 virtual Coin::ReferencedObject::~~ReferencedObject () [inline], [virtual]

Definition at line 160 of file CoinSmartPtr.hpp.

9.117.3 Member Function Documentation

9.117.3.1 int Coin::ReferencedObject::ReferenceCount () const [inline]

Definition at line 161 of file CoinSmartPtr.hpp.

9.117.3.2 void Coin::ReferencedObject::AddRef () const [inline]

Definition at line 162 of file CoinSmartPtr.hpp.

9.117.3.3 void Coin::ReferencedObject::ReleaseRef () const [inline]

Definition at line 163 of file CoinSmartPtr.hpp.

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

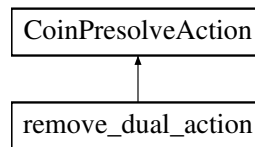
- /home/ted/COIN/trunk/CoinUtils/src/CoinSmartPtr.hpp

9.118 remove_dual_action Class Reference

Attempt to fix variables by bounding reduced costs.

```
#include <CoinPresolveDual.hpp>
```

Inheritance diagram for remove_dual_action:



Public Member Functions

- [~remove_dual_action](#) ()
Destructor.
- const char * [name](#) () const
Name.
- void [postsolve](#) ([CoinPostsolveMatrix](#) *prob) const
Postsolve.

Static Public Member Functions

- static const [CoinPresolveAction](#) * [presolve](#) ([CoinPresolveMatrix](#) *prob, const [CoinPresolveAction](#) *next)
Attempt to fix variables by bounding reduced costs.

Additional Inherited Members

9.118.1 Detailed Description

Attempt to fix variables by bounding reduced costs.

The reduced cost of x_j is $d_j = c_j - y \cdot a_j$ (1). Assume minimization, so that at optimality $d_j \geq 0$ for x_j nonbasic at lower bound, and $d_j \leq 0$ for x_j nonbasic at upper bound.

For a slack variable s_i , $c_{(n+i)} = 0$ and $a_{(n+i)}$ is a unit vector, hence $d_{(n+i)} = -y_i$. If s_i has a finite lower bound and no upper bound, we must have $y_i \leq 0$ at optimality. Similarly, if s_i has no lower bound and a finite upper bound, we must have $y_i \geq 0$.

For a singleton variable x_j , $d_j = c_j - y_i \cdot a_{ij}$. Given x_j with a single finite bound, we can bound d_j greater or less than 0 at optimality, and that allows us to calculate an upper or lower bound on y_i (depending on the bound on d_j and the sign of a_{ij}).

Now we have bounds on some subset of the y_i , and we can use these to calculate upper and lower bounds on the d_j , using bound propagation on (1). If we can manage to bound some d_j as strictly positive or strictly negative, then at

optimality the corresponding variable must be nonbasic at its lower or upper bound, respectively. If the required bound is lacking, the problem is unbounded.

Definition at line 35 of file CoinPresolveDual.hpp.

9.118.2 Constructor & Destructor Documentation

9.118.2.1 `remove_dual_action::~~remove_dual_action ()`

Destructor.

9.118.3 Member Function Documentation

9.118.3.1 `const char* remove_dual_action::name () const` `[inline],[virtual]`

Name.

Implements [CoinPresolveAction](#).

Definition at line 43 of file CoinPresolveDual.hpp.

9.118.3.2 `static const CoinPresolveAction* remove_dual_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next)` `[static]`

Attempt to fix variables by bounding reduced costs.

Always scans all variables. Propagates bounds on reduced costs until there's no change or until some set of variables can be fixed.

9.118.3.3 `void remove_dual_action::postsolve (CoinPostsolveMatrix * prob) const` `[virtual]`

Postsolve.

In addition to fixing variables (handled by [make_fixed_action](#)), we may need use our own postsolve to restore constraint bounds.

Implements [CoinPresolveAction](#).

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

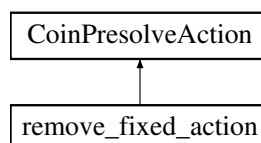
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDual.hpp](#)

9.119 `remove_fixed_action` Class Reference

Excise fixed variables from the model.

```
#include <CoinPresolveFixed.hpp>
```

Inheritance diagram for `remove_fixed_action`:



Classes

- struct `action`

Structure to hold information necessary to reintroduce a column into the problem representation.

Public Member Functions

- `const char * name () const`
Returns string "remove_fixed_action".
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~remove_fixed_action ()`
Destructor.

Static Public Member Functions

- `static const remove_fixed_action * presolve (CoinPresolveMatrix *prob, int *fcols, int nfc, const CoinPresolveAction *next)`
Excise the specified columns.

Public Attributes

- `int * colrows_`
Array of row indices for coefficients of excised columns.
- `double * coeels_`
Array of coefficients of excised columns.
- `int nactions_`
Number of entries in `actions_`.
- `action * actions_`
Vector specifying variable(s) affected by this object.

Related Functions

(Note that these are not member functions.)

- `const CoinPresolveAction * remove_fixed (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`
Scan the problem for fixed columns and remove them.

9.119.1 Detailed Description

Excise fixed variables from the model.

Implements the action of virtually removing one or more fixed variables x_j from the model by substituting the value sol_j in each constraint. Specifically, for each constraint i where $a_{ij} \neq 0$, rlo_i and rup_i are adjusted by $-a_{ij} \cdot sol_j$ and a_{ij} is set to 0.

There is an implicit assumption that the variable already has the correct value. If this isn't true, corrections to row activity may be incorrect. If you want to guard against this possibility, consider `make_fixed_action`.

Actual removal of the empty column from the matrix is handled by [drop_empty_cols_action](#). Correction of the objective function is done there.

Definition at line 25 of file CoinPresolveFixed.hpp.

9.119.2 Constructor & Destructor Documentation

9.119.2.1 `virtual remove_fixed_action::~~remove_fixed_action () [virtual]`

Destructor.

9.119.3 Member Function Documentation

9.119.3.1 `const char* remove_fixed_action::name () const [virtual]`

Returns string "remove_fixed_action".

Implements [CoinPresolveAction](#).

9.119.3.2 `static const remove_fixed_action* remove_fixed_action::presolve (CoinPresolveMatrix * prob, int * fcols, int nfcols, const CoinPresolveAction * next) [static]`

Excise the specified columns.

Remove the specified columns (*nfcols*, *fcols*) from the problem representation (*prob*), leaving the appropriate postsolve object linked as the head of the list of postsolve objects (currently headed by *next*).

9.119.3.3 `void remove_fixed_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

9.119.4 Friends And Related Function Documentation

9.119.4.1 `const CoinPresolveAction * remove_fixed (CoinPresolveMatrix * prob, const CoinPresolveAction * next) [related]`

Scan the problem for fixed columns and remove them.

A front end to collect a list of columns with equal bounds and hand them to [remove_fixed_action::presolve\(\)](#) for processing.

9.119.5 Member Data Documentation

9.119.5.1 `int* remove_fixed_action::colrows_`

Array of row indices for coefficients of excised columns.

Definition at line 36 of file CoinPresolveFixed.hpp.

9.119.5.2 `double* remove_fixed_action::colels_`

Array of coefficients of excised columns.

Definition at line 38 of file CoinPresolveFixed.hpp.

9.119.5.3 int remove_fixed_action::nactions_

Number of entries in [actions_](#).

Definition at line 40 of file CoinPresolveFixed.hpp.

9.119.5.4 action* remove_fixed_action::actions_

Vector specifying variable(s) affected by this object.

Definition at line 42 of file CoinPresolveFixed.hpp.

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

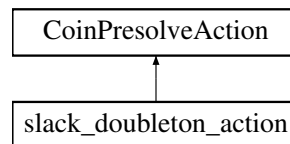
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveFixed.hpp](#)

9.120 slack_doubleton_action Class Reference

Convert an explicit bound constraint to a column bound.

```
#include <CoinPresolveSingleton.hpp>
```

Inheritance diagram for slack_doubleton_action:



Public Member Functions

- const char * [name](#) () const
A name for debug printing.
- void [postsolve](#) ([CoinPostsolveMatrix](#) *prob) const
Apply the postsolve transformation for this particular presolve action.
- virtual [~slack_doubleton_action](#) ()

Static Public Member Functions

- static const [CoinPresolveAction](#) * [presolve](#) ([CoinPresolveMatrix](#) *prob, const [CoinPresolveAction](#) *next, bool ¬Finished)
Convert explicit bound constraints to column bounds.

Additional Inherited Members

9.120.1 Detailed Description

Convert an explicit bound constraint to a column bound.

This transform looks for explicit bound constraints for a variable and transfers the bound to the appropriate column bound array. The constraint is removed from the constraint system.

Definition at line 24 of file CoinPresolveSingleton.hpp.

9.120.2 Constructor & Destructor Documentation

9.120.2.1 `virtual slack_doubleton_action::~slack_doubleton_action () [inline],[virtual]`

Definition at line 65 of file `CoinPresolveSingleton.hpp`.

9.120.3 Member Function Documentation

9.120.3.1 `const char* slack_doubleton_action::name () const [inline],[virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

Definition at line 50 of file `CoinPresolveSingleton.hpp`.

9.120.3.2 `static const CoinPresolveAction* slack_doubleton_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next, bool & notFinished) [static]`

Convert explicit bound constraints to column bounds.

Not now There is a hard limit (`#MAX_SLACK_DOUBLETONS`) on the number of constraints processed in a given call. `notFinished` is set to true if candidates remain.

9.120.3.3 `void slack_doubleton_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

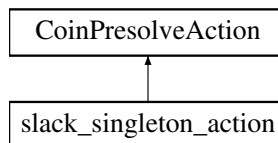
- `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSingleton.hpp`

9.121 slack_singleton_action Class Reference

For variables with one entry.

```
#include <CoinPresolveSingleton.hpp>
```

Inheritance diagram for `slack_singleton_action`:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`

Apply the postsolve transformation for this particular presolve action.

- virtual [~slack_singleton_action](#) ()

Static Public Member Functions

- static const [CoinPresolveAction](#) * [presolve](#) ([CoinPresolveMatrix](#) *prob, const [CoinPresolveAction](#) *next, double *rowObjective)

Additional Inherited Members

9.121.1 Detailed Description

For variables with one entry.

If we have a variable with one entry and no cost then we can transform the row from E to G etc. If there is a row objective region then we may be able to do this even with a cost.

Definition at line 75 of file [CoinPresolveSingleton.hpp](#).

9.121.2 Constructor & Destructor Documentation

9.121.2.1 virtual [slack_singleton_action::~slack_singleton_action](#) () [\[inline\]](#), [\[virtual\]](#)

Definition at line 110 of file [CoinPresolveSingleton.hpp](#).

9.121.3 Member Function Documentation

9.121.3.1 const char* [slack_singleton_action::name](#) () const [\[inline\]](#), [\[virtual\]](#)

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

Definition at line 101 of file [CoinPresolveSingleton.hpp](#).

9.121.3.2 static const [CoinPresolveAction](#)* [slack_singleton_action::presolve](#) ([CoinPresolveMatrix](#) * prob, const [CoinPresolveAction](#) * next, double * rowObjective) [\[static\]](#)

9.121.3.3 void [slack_singleton_action::postsolve](#) ([CoinPostsolveMatrix](#) * prob) const [\[virtual\]](#)

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSingleton.hpp](#)

9.122 Coin::SmartPtr< T > Class Template Reference

Template class for Smart Pointers.

```
#include <CoinSmartPtr.hpp>
```

Public Member Functions

- `T * GetRawPtr () const`
Returns the raw pointer contained.
- `bool IsValid () const`
Returns true if the `SmartPtr` is NOT NULL.
- `bool IsNull () const`
Returns true if the `SmartPtr` is NULL.

Constructors/Destructors

- `SmartPtr ()`
Default constructor, initialized to NULL.
- `SmartPtr (const SmartPtr< T > ©)`
Copy constructor, initialized from copy.
- `SmartPtr (T *ptr)`
Constructor, initialized from `T ptr`.*
- `~SmartPtr ()`
Destructor, automatically decrements the reference count, deletes the object if necessary.

Overloaded operators.

- `T * operator-> () const`
Overloaded arrow operator, allows the user to call methods using the contained pointer.
- `T & operator* () const`
Overloaded dereference operator, allows the user to dereference the contained pointer.
- `SmartPtr< T > & operator= (T *rhs)`
Overloaded equals operator, allows the user to set the value of the `SmartPtr` from a raw pointer.
- `SmartPtr< T > & operator= (const SmartPtr< T > &rhs)`
Overloaded equals operator, allows the user to set the value of the `SmartPtr` from another `SmartPtr`.
- `template<class U1 , class U2 >`
`bool operator== (const SmartPtr< U1 > &lhs, const SmartPtr< U2 > &rhs)`
Overloaded equality comparison operator, allows the user to compare the value of two `SmartPtrs`.
- `template<class U1 , class U2 >`
`bool operator== (const SmartPtr< U1 > &lhs, U2 *raw_rhs)`
Overloaded equality comparison operator, allows the user to compare the value of a `SmartPtr` with a raw pointer.
- `template<class U1 , class U2 >`
`bool operator== (U1 *lhs, const SmartPtr< U2 > &raw_rhs)`
Overloaded equality comparison operator, allows the user to compare the value of a raw pointer with a `SmartPtr`.
- `template<class U1 , class U2 >`
`bool operator!= (const SmartPtr< U1 > &lhs, const SmartPtr< U2 > &rhs)`
Overloaded in-equality comparison operator, allows the user to compare the value of two `SmartPtrs`.
- `template<class U1 , class U2 >`
`bool operator!= (const SmartPtr< U1 > &lhs, U2 *raw_rhs)`
Overloaded in-equality comparison operator, allows the user to compare the value of a `SmartPtr` with a raw pointer.
- `template<class U1 , class U2 >`
`bool operator!= (U1 *lhs, const SmartPtr< U2 > &raw_rhs)`
Overloaded in-equality comparison operator, allows the user to compare the value of a `SmartPtr` with a raw pointer.

9.122.1 Detailed Description

```
template<class T>class Coin::SmartPtr< T >
```

Template class for Smart Pointers.

A [SmartPtr](#) behaves much like a raw pointer, but manages the lifetime of an object, deleting the object automatically. This class implements a reference-counting, intrusive smart pointer design, where all objects pointed to must inherit off of [ReferencedObject](#), which stores the reference count. Although this is intrusive (native types and externally authored classes require wrappers to be referenced by smart pointers), it is a safer design. A more detailed discussion of these issues follows after the usage information.

Usage Example: Note: to use the [SmartPtr](#), all objects to which you point MUST inherit off of [ReferencedObject](#).

```
*
* In MyClass.hpp...
*
* #include "CoinSmartPtr.hpp"
*
* class MyClass : public Coin::ReferencedObject // must derive from ReferencedObject
* {
*     ...
* }
*
* In my_usage.cpp...
*
* #include "CoinSmartPtr.hpp"
* #include "MyClass.hpp"
*
* void func(AnyObject& obj)
* {
*     SmartPtr<MyClass> ptr_to_myclass = new MyClass(...);
*     // ptr_to_myclass now points to a new MyClass,
*     // and the reference count is 1
*     ...
*
*     obj.SetMyClass(ptr_to_myclass);
*     // Here, let's assume that AnyObject uses a
*     // SmartPtr<MyClass> internally here.
*     // Now, both ptr_to_myclass and the internal
*     // SmartPtr in obj point to the same MyClass object
*     // and its reference count is 2.
*     ...
*
*     // No need to delete ptr_to_myclass, this
*     // will be done automatically when the
*     // reference count drops to zero.
* }
*
*
*
```

It is not necessary to use [SmartPtr](#)'s in all cases where an object is used that has been allocated "into" a [SmartPtr](#). It is possible to just pass objects by reference or regular pointers, even if lower down in the stack a [SmartPtr](#) is to be held on to. Everything should work fine as long as a pointer created by "new" is immediately passed into a [SmartPtr](#), and if [SmartPtr](#)'s are used to hold on to objects.

Other Notes: The [SmartPtr](#) implements both dereference operators -> & *. The [SmartPtr](#) does NOT implement a conversion operator to the raw pointer. Use the [GetRawPtr\(\)](#) method when this is necessary. Make sure that the raw pointer is NOT deleted. The [SmartPtr](#) implements the comparison operators == & != for a variety of types. Use these instead of

```
*     if (GetRawPtr(smrt_ptr) == ptr) // Don't use this
```

*

SmartPtr's, as currently implemented, do NOT handle circular references. For example: consider a higher level object using **SmartPtr**s to point to A and B, but A and B also point to each other (i.e. A has a **SmartPtr** to B and B has a **SmartPtr** to A). In this scenario, when the higher level object is finished with A and B, their reference counts will never drop to zero (since they reference each other) and they will not be deleted. This can be detected by memory leak tools like valgrind. If the circular reference is necessary, the problem can be overcome by a number of techniques:

1) A and B can have a method that "releases" each other, that is they set their internal **SmartPtr**s to NULL.

```
*      void AClass::ReleaseCircularReferences()
*      {
*          smart_ptr_to_B = NULL;
*      }
*
```

Then, the higher level class can call these methods before it is done using A & B.

2) Raw pointers can be used in A and B to reference each other. Here, an implicit assumption is made that the lifetime is controlled by the higher level object and that A and B will both exist in a controlled manner. Although this seems dangerous, in many situations, this type of referencing is very controlled and this is reasonably safe.

3) This **SmartPtr** class could be redesigned with the Weak/Strong design concept. Here, the **SmartPtr** is identified as being Strong (controls lifetime of the object) or Weak (merely referencing the object). The Strong **SmartPtr** increments (and decrements) the reference count in **ReferencedObject** but the Weak **SmartPtr** does not. In the example above, the higher level object would have Strong **SmartPtr**s to A and B, but A and B would have Weak **SmartPtr**s to each other. Then, when the higher level object was done with A and B, they would be deleted. The Weak **SmartPtr**s in A and B would not decrement the reference count and would, of course, not delete the object. This idea is very similar to item (2), where it is implied that the sequence of events is controlled such that A and B will not call anything using their pointers following the higher level delete (i.e. in their destructors!). This is somehow safer, however, because code can be written (however expensive) to perform run-time detection of this situation. For example, the **ReferencedObject** could store pointers to all Weak **SmartPtr**s that are referencing it and, in its destructor, tell these pointers that it is dying. They could then set themselves to NULL, or set an internal flag to detect usage past this point.

Comments on Non-Intrusive Design: In a non-intrusive design, the reference count is stored somewhere other than the object being referenced. This means, unless the reference counting pointer is the first referencer, it must get a pointer to the referenced object from another smart pointer (so it has access to the reference count location). In this non-intrusive design, if we are pointing to an object with a smart pointer (or a number of smart pointers), and we then give another smart pointer the address through a RAW pointer, we will have two independent, AND INCORRECT, reference counts. To avoid this pitfall, we use an intrusive reference counting technique where the reference count is stored in the object being referenced.

Definition at line 319 of file CoinSmartPtr.hpp.

9.122.2 Constructor & Destructor Documentation

9.122.2.1 `template<class T> Coin::SmartPtr< T >::SmartPtr () [inline]`

Default constructor, initialized to NULL.

Definition at line 384 of file CoinSmartPtr.hpp.

9.122.2.2 `template<class T> Coin::SmartPtr< T >::SmartPtr (const SmartPtr< T > ©) [inline]`

Copy constructor, initialized from copy.

Definition at line 387 of file CoinSmartPtr.hpp.

9.122.2.3 `template<class T> Coin::SmartPtr< T >::SmartPtr (T * ptr) [inline]`

Constructor, initialized from T* ptr.

Definition at line 392 of file CoinSmartPtr.hpp.

9.122.2.4 `template<class T> Coin::SmartPtr< T >::~~SmartPtr () [inline]`

Destructor, automatically decrements the reference count, deletes the object if necessary.

Definition at line 398 of file CoinSmartPtr.hpp.

9.122.3 Member Function Documentation

9.122.3.1 `template<class T> T* Coin::SmartPtr< T >::GetRawPtr () const [inline]`

Returns the raw pointer contained.

Use to get the value of the raw ptr (i.e. to pass to other methods/functions, etc.) Note: This method does NOT copy, therefore, modifications using this value modify the underlying object contained by the [SmartPtr](#), NEVER delete this returned value.

Definition at line 327 of file CoinSmartPtr.hpp.

9.122.3.2 `template<class T> bool Coin::SmartPtr< T >::IsValid () const [inline]`

Returns true if the [SmartPtr](#) is NOT NULL.

Use this to check if the [SmartPtr](#) is not null This is preferred to `if(GetRawPtr(sp) != NULL)`

Definition at line 333 of file CoinSmartPtr.hpp.

9.122.3.3 `template<class T> bool Coin::SmartPtr< T >::IsNull () const [inline]`

Returns true if the [SmartPtr](#) is NULL.

Use this to check if the [SmartPtr](#) IsNull. This is preferred to `if(GetRawPtr(sp) == NULL)`

Definition at line 339 of file CoinSmartPtr.hpp.

9.122.3.4 `template<class T> T* Coin::SmartPtr< T >::operator-> () const [inline]`

Overloaded arrow operator, allows the user to call methods using the contained pointer.

Definition at line 407 of file CoinSmartPtr.hpp.

9.122.3.5 `template<class T> T& Coin::SmartPtr< T >::operator* () const [inline]`

Overloaded dereference operator, allows the user to dereference the contained pointer.

Definition at line 416 of file CoinSmartPtr.hpp.

9.122.3.6 `template<class T> SmartPtr<T>& Coin::SmartPtr< T >::operator= (T * rhs) [inline]`

Overloaded equals operator, allows the user to set the value of the [SmartPtr](#) from a raw pointer.

Definition at line 425 of file CoinSmartPtr.hpp.

9.122.3.7 `template<class T> SmartPtr<T>& Coin::SmartPtr< T >::operator= (const SmartPtr< T > & rhs)`
`[inline]`

Overloaded equals operator, allows the user to set the value of the [SmartPtr](#) from another [SmartPtr](#).

Definition at line 432 of file CoinSmartPtr.hpp.

9.122.4 Friends And Related Function Documentation

9.122.4.1 `template<class T> template<class U1 , class U2 > bool operator== (const SmartPtr< U1 > & lhs, const SmartPtr< U2 > & rhs)` `[friend]`

Overloaded equality comparison operator, allows the user to compare the value of two SmartPtrs.

Definition at line 494 of file CoinSmartPtr.hpp.

9.122.4.2 `template<class T> template<class U1 , class U2 > bool operator== (const SmartPtr< U1 > & lhs, U2 * raw_rhs)`
`[friend]`

Overloaded equality comparison operator, allows the user to compare the value of a [SmartPtr](#) with a raw pointer.

Definition at line 499 of file CoinSmartPtr.hpp.

9.122.4.3 `template<class T> template<class U1 , class U2 > bool operator== (U1 * lhs, const SmartPtr< U2 > & raw_rhs)`
`[friend]`

Overloaded equality comparison operator, allows the user to compare the value of a raw pointer with a [SmartPtr](#).

Definition at line 504 of file CoinSmartPtr.hpp.

9.122.4.4 `template<class T> template<class U1 , class U2 > bool operator!= (const SmartPtr< U1 > & lhs, const SmartPtr< U2 > & rhs)` `[friend]`

Overloaded in-equality comparison operator, allows the user to compare the value of two SmartPtrs.

Definition at line 509 of file CoinSmartPtr.hpp.

9.122.4.5 `template<class T> template<class U1 , class U2 > bool operator!= (const SmartPtr< U1 > & lhs, U2 * raw_rhs)`
`[friend]`

Overloaded in-equality comparison operator, allows the user to compare the value of a [SmartPtr](#) with a raw pointer.

Definition at line 514 of file CoinSmartPtr.hpp.

9.122.4.6 `template<class T> template<class U1 , class U2 > bool operator!= (U1 * lhs, const SmartPtr< U2 > & raw_rhs)`
`[friend]`

Overloaded in-equality comparison operator, allows the user to compare the value of a [SmartPtr](#) with a raw pointer.

Definition at line 519 of file CoinSmartPtr.hpp.

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

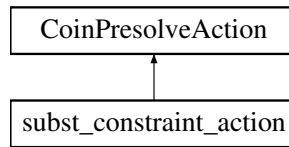
- [/home/ted/COIN/trunk/CoinUtils/src/CoinSmartPtr.hpp](#)

9.123 subst_constraint_action Class Reference

Detect and process implied free variables.

```
#include <CoinPresolveSubst.hpp>
```

Inheritance diagram for subst_constraint_action:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~subst_constraint_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const int *implied_free, const int *which, int numberFree, const CoinPresolveAction *next, int fill_level)`
- `static const CoinPresolveAction * presolveX (CoinPresolveMatrix *prob, const CoinPresolveAction *next, int fill_level)`

Additional Inherited Members

9.123.1 Detailed Description

Detect and process implied free variables.

Consider a variable x . Suppose that we can find an equality such that the bound on the equality, combined with the bounds on the other variables involved in the equality, are such that even the worst case values of the other variables still imply bounds for x which are tighter than the variable's original bounds. Since x can never reach its upper or lower bounds, it is an implied free variable. By solving the equality for x and substituting for x in every other constraint entangled with x , we can make x into a column singleton. Now x is an implied free column singleton and both x and the equality can be removed.

A similar transform for the case where the variable is a natural column singleton is handled by `implied_free_action`. In the current presolve architecture, `implied_free_action` is responsible for detecting implied free variables that are natural column singletons or can be reduced to column singletons. `implied_free_action` calls `subst_constraint_action` to process variables that must be reduced to column singletons.

Definition at line 37 of file `CoinPresolveSubst.hpp`.

9.123.2 Constructor & Destructor Documentation

9.123.2.1 `virtual subst_constraint_action::~~subst_constraint_action () [virtual]`

9.123.3 Member Function Documentation

9.123.3.1 `const char* subst_constraint_action::name () const` [virtual]

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

9.123.3.2 `static const CoinPresolveAction* subst_constraint_action::presolve (CoinPresolveMatrix * prob, const int * implied_free, const int * which, int numberFree, const CoinPresolveAction * next, int fill_level)` [static]

9.123.3.3 `static const CoinPresolveAction* subst_constraint_action::presolveX (CoinPresolveMatrix * prob, const CoinPresolveAction * next, int fillLevel)` [static]

9.123.3.4 `void subst_constraint_action::postsolve (CoinPostsolveMatrix * prob) const` [virtual]

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSubst.hpp](#)

9.124 symrec Struct Reference

For string evaluation.

```
#include <CoinModelUseful.hpp>
```

Public Attributes

- `char * name`
- `int type`
- `union {
 double var
 func_t fnctptr
} value`
- `struct symrec * next`

9.124.1 Detailed Description

For string evaluation.

Definition at line 137 of file `CoinModelUseful.hpp`.

9.124.2 Member Data Documentation

9.124.2.1 `char* symrec::name`

Definition at line 139 of file `CoinModelUseful.hpp`.

9.124.2.2 `int symrec::type`

Definition at line 140 of file `CoinModelUseful.hpp`.

9.124.2.3 double symrec::var

Definition at line 143 of file CoinModelUseful.hpp.

9.124.2.4 func_t symrec::fnctptr

Definition at line 144 of file CoinModelUseful.hpp.

9.124.2.5 union { ... } symrec::value

9.124.2.6 struct symrec* symrec::next

Definition at line 146 of file CoinModelUseful.hpp.

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

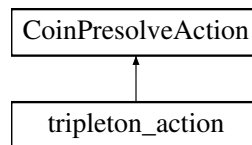
- /home/ted/COIN/trunk/CoinUtils/src/[CoinModelUseful.hpp](#)

9.125 tripleton_action Class Reference

We are only going to do this if it does not increase number of elements?.

```
#include <CoinPresolveTripletion.hpp>
```

Inheritance diagram for tripleton_action:



Classes

- struct [action](#)

Public Member Functions

- const char * [name](#) () const
A name for debug printing.
- void [postsolve](#) (CoinPostsolveMatrix *prob) const
Apply the postsolve transformation for this particular presolve action.
- virtual [~tripleton_action](#) ()

Static Public Member Functions

- static const [CoinPresolveAction](#) * [presolve](#) (CoinPresolveMatrix *, const [CoinPresolveAction](#) *next)

Public Attributes

- const int [nactions_](#)
- const [action](#) *const [actions_](#)

9.125.1 Detailed Description

We are only going to do this if it does not increase number of elements?.

It could be generalized to more than three but it seems unlikely it would help.

As it is adapted from doubleton icoly is one dropped.

Definition at line 15 of file CoinPresolveTripletion.hpp.

9.125.2 Constructor & Destructor Documentation

9.125.2.1 `virtual tripletion_action::~~tripletion_action () [virtual]`

9.125.3 Member Function Documentation

9.125.3.1 `const char* tripletion_action::name () const [inline],[virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

Definition at line 55 of file CoinPresolveTripletion.hpp.

9.125.3.2 `static const CoinPresolveAction* tripletion_action::presolve (CoinPresolveMatrix *, const CoinPresolveAction * next) [static]`

9.125.3.3 `void tripletion_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

9.125.4 Member Data Documentation

9.125.4.1 `const int tripletion_action::nactions_`

Definition at line 43 of file CoinPresolveTripletion.hpp.

9.125.4.2 `const action* const tripletion_action::actions_`

Definition at line 44 of file CoinPresolveTripletion.hpp.

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

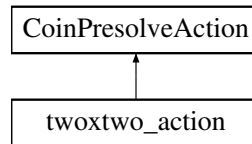
- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTripletion.hpp](#)

9.126 twotwo_action Class Reference

Detect interesting 2 by 2 blocks.

```
#include <CoinPresolveDupcol.hpp>
```

Inheritance diagram for twotwo_action:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `~twotwo_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`

Additional Inherited Members

9.126.1 Detailed Description

Detect interesting 2 by 2 blocks.

If a variable has two entries and for each row there are only two entries with same other variable then we can get rid of one constraint and modify costs.

This is a work in progress - I need more examples

Definition at line 163 of file CoinPresolveDupcol.hpp.

9.126.2 Constructor & Destructor Documentation

9.126.2.1 `twotwo_action::~~twotwo_action () [inline]`

Definition at line 194 of file CoinPresolveDupcol.hpp.

9.126.3 Member Function Documentation

9.126.3.1 `const char* twotwo_action::name () const [virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

9.126.3.2 `static const CoinPresolveAction* twotwo_action::presolve (CoinPresolveMatrix * prob, const CoinPresolveAction * next) [static]`

9.126.3.3 `void twotwo_action::postsolve (CoinPostsolveMatrix * prob) const [virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

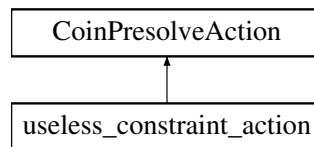
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDupcol.hpp](#)

9.127 useless_constraint_action Class Reference

```
#include <CoinPresolveUseless.hpp>
```

Inheritance diagram for `useless_constraint_action`:



Public Member Functions

- `const char * name () const`
A name for debug printing.
- `void postsolve (CoinPostsolveMatrix *prob) const`
Apply the postsolve transformation for this particular presolve action.
- `virtual ~useless_constraint_action ()`

Static Public Member Functions

- `static const CoinPresolveAction * presolve (CoinPresolveMatrix *prob, const int *useless_rows, int nuseless_rows, const CoinPresolveAction *next)`

Related Functions

(Note that these are not member functions.)

- `const CoinPresolveAction * testRedundant (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`
Scan constraints looking for useless constraints.

Additional Inherited Members

9.127.1 Detailed Description

Definition at line 10 of file `CoinPresolveUseless.hpp`.

9.127.2 Constructor & Destructor Documentation

9.127.2.1 `virtual useless_constraint_action::~useless_constraint_action () [virtual]`

9.127.3 Member Function Documentation

9.127.3.1 `const char* useless_constraint_action::name () const` `[virtual]`

A name for debug printing.

It is expected that the name is not stored in the transform itself.

Implements [CoinPresolveAction](#).

9.127.3.2 `static const CoinPresolveAction* useless_constraint_action::presolve (CoinPresolveMatrix * prob, const int * useless_rows, int nuseless_rows, const CoinPresolveAction * next)` `[static]`

9.127.3.3 `void useless_constraint_action::postsolve (CoinPostsolveMatrix * prob) const` `[virtual]`

Apply the postsolve transformation for this particular presolve action.

Implements [CoinPresolveAction](#).

9.127.4 Friends And Related Function Documentation

9.127.4.1 `const CoinPresolveAction * testRedundant (CoinPresolveMatrix * prob, const CoinPresolveAction * next)` `[related]`

Scan constraints looking for useless constraints.

A front end to identify useless constraints and hand them to [useless_constraint_action::presolve\(\)](#) for processing.

In a bit more detail, the routine implements a greedy algorithm that identifies a set of necessary constraints. A constraint is necessary if it implies a tighter bound on a variable than the original column bound. These tighter column bounds are then used to calculate row activity and identify constraints that are useless given the presence of the necessary constraints.

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

- [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveUseless.hpp](#)

10 File Documentation

10.1 [/home/ted/COIN/trunk/CoinUtils/src/Coin_C_defines.h](#) File Reference

Macros

- `#define COINLIBAPI`
This has #defines etc for the "C" interface to [Coin](#).
- `#define COINLINKAGE`
- `#define COINLINKAGE_CB`

Typedefs

- `typedef void Clp_Simplex`
User does not need to see structure of model but C++ code does.
- `typedef int msgno`
- `typedef int int ndouble`

- typedef int int const double * [dvec](#)
- typedef int int const double int [nint](#)
- typedef int int const double
int const int * [ivec](#)
- typedef int int const double
int const int int [nchar](#)
- typedef int int const double
int const int int char ** [cvec](#)
- typedef [void Sbb_Model](#)

User does not need to see structure of model but C++ code does.

- typedef [void Cbc_Model](#)
- typedef int [CoinBigIndex](#)

Functions

- typedef [void \(COINLINKAGE_CB *clp_callback\)\(Clp_Simplex *model](#)
typedef for user call back.

10.1.1 Macro Definition Documentation

10.1.1.1 #define COINLIBAPI

This has #defines etc for the "C" interface to [Coin](#).

If COIN_EXTERN_C defined then an extra extern C

Definition at line 43 of file [Coin_C_defines.h](#).

10.1.1.2 #define COINLINKAGE

Definition at line 45 of file [Coin_C_defines.h](#).

10.1.1.3 #define COINLINKAGE_CB

Definition at line 46 of file [Coin_C_defines.h](#).

10.1.2 Typedef Documentation

10.1.2.1 typedef void Clp_Simplex

User does not need to see structure of model but C++ code does.

Definition at line 59 of file [Coin_C_defines.h](#).

10.1.2.2 typedef int msgno

Definition at line 65 of file [Coin_C_defines.h](#).

10.1.2.3 typedef int int ndouble

Definition at line 65 of file [Coin_C_defines.h](#).

10.1.2.4 typedef int int const double * dvec

Definition at line 65 of file [Coin_C_defines.h](#).

10.1.2.5 typedef int int const double int nint

Definition at line 65 of file Coin_C_defines.h.

10.1.2.6 typedef int int const double int const int * ivec

Definition at line 65 of file Coin_C_defines.h.

10.1.2.7 typedef int int const double int const int int nchar

Definition at line 65 of file Coin_C_defines.h.

10.1.2.8 typedef int int const double int const int int char ** cvec

Definition at line 65 of file Coin_C_defines.h.

10.1.2.9 typedef void Sbb_Model

User does not need to see structure of model but C++ code does.

Definition at line 80 of file Coin_C_defines.h.

10.1.2.10 typedef void Cbc_Model

Definition at line 92 of file Coin_C_defines.h.

10.1.2.11 typedef int CoinBigIndex

Definition at line 105 of file Coin_C_defines.h.

10.1.3 Function Documentation**10.1.3.1 typedef void (COINLINKAGE_CB * sbb_callback)**

typedef for user call back.

The cvec are constructed so don't need to be const

10.2 /home/ted/COIN/trunk/CoinUtils/src/CoinAlloc.hpp File Reference

```
#include "CoinUtilsConfig.h"
#include <cstdlib>
```

Macros

- `#define COINUTILS_MEMPOOL_MAXPOOLED -1`

10.2.1 Macro Definition Documentation**10.2.1.1 #define COINUTILS_MEMPOOL_MAXPOOLED -1**

Definition at line 13 of file CoinAlloc.hpp.

10.3 /home/ted/COIN/trunk/CoinUtils/src/CoinBuild.hpp File Reference

```
#include "CoinPragma.hpp"
#include "CoinTypes.hpp"
#include "CoinFinite.hpp"
```

Classes

- class [CoinBuild](#)

In many cases it is natural to build a model by adding one row at a time.

10.4 /home/ted/COIN/trunk/CoinUtils/src/CoinDenseFactorization.hpp File Reference

```
#include <iostream>
#include <string>
#include <cassert>
#include "CoinTypes.hpp"
#include "CoinIndexedVector.hpp"
#include "CoinFactorization.hpp"
```

Classes

- class [CoinOtherFactorization](#)

Abstract base class which also has some scalars so can be used from Dense or Simp.

- class [CoinDenseFactorization](#)

This deals with Factorization and Updates This is a simple dense version so other people can write a better one.

10.5 /home/ted/COIN/trunk/CoinUtils/src/CoinDenseVector.hpp File Reference

```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include "CoinHelperFunctions.hpp"
```

Classes

- class [CoinDenseVector< T >](#)

Dense Vector.

Functions

- template<typename T >
[void CoinDenseVectorUnitTest](#) (T dummy)

A function that tests the methods in the [CoinDenseVector](#) class.

Arithmetic operators on dense vectors.

NOTE: Because these methods return an object (they can't return a reference, though they could return a pointer...) they are very inefficient...

- `template<typename T >`
`CoinDenseVector< T > operator+ (const CoinDenseVector< T > &op1, const CoinDenseVector< T > &op2)`
Return the sum of two dense vectors.
- `template<typename T >`
`CoinDenseVector< T > operator- (const CoinDenseVector< T > &op1, const CoinDenseVector< T > &op2)`
Return the difference of two dense vectors.
- `template<typename T >`
`CoinDenseVector< T > operator* (const CoinDenseVector< T > &op1, const CoinDenseVector< T > &op2)`
Return the element-wise product of two dense vectors.
- `template<typename T >`
`CoinDenseVector< T > operator/ (const CoinDenseVector< T > &op1, const CoinDenseVector< T > &op2)`
Return the element-wise ratio of two dense vectors.

Arithmetic operators on dense vector and a constant.

These functions create a dense vector as a result.

That dense vector will have the same indices as `op1` and the specified operation is done entry-wise with the given value.

- `template<typename T >`
`CoinDenseVector< T > operator+ (const CoinDenseVector< T > &op1, T value)`
Return the sum of a dense vector and a constant.
- `template<typename T >`
`CoinDenseVector< T > operator- (const CoinDenseVector< T > &op1, T value)`
Return the difference of a dense vector and a constant.
- `template<typename T >`
`CoinDenseVector< T > operator* (const CoinDenseVector< T > &op1, T value)`
Return the element-wise product of a dense vector and a constant.
- `template<typename T >`
`CoinDenseVector< T > operator/ (const CoinDenseVector< T > &op1, T value)`
Return the element-wise ratio of a dense vector and a constant.
- `template<typename T >`
`CoinDenseVector< T > operator+ (T value, const CoinDenseVector< T > &op1)`
Return the sum of a constant and a dense vector.
- `template<typename T >`
`CoinDenseVector< T > operator- (T value, const CoinDenseVector< T > &op1)`
Return the difference of a constant and a dense vector.
- `template<typename T >`
`CoinDenseVector< T > operator* (T value, const CoinDenseVector< T > &op1)`
Return the element-wise product of a constant and a dense vector.
- `template<typename T >`
`CoinDenseVector< T > operator/ (T value, const CoinDenseVector< T > &op1)`
Return the element-wise ratio of a a constant and dense vector.

10.5.1 Function Documentation**10.5.1.1 `template<typename T > void CoinDenseVectorUnitTest (T dummy)`**

A function that tests the methods in the `CoinDenseVector` class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

10.5.1.2 `template<typename T> CoinDenseVector<T> operator+ (const CoinDenseVector< T> & op1, const CoinDenseVector< T> & op2) [inline]`

Return the sum of two dense vectors.

Definition at line 213 of file CoinDenseVector.hpp.

10.5.1.3 `template<typename T> CoinDenseVector<T> operator- (const CoinDenseVector< T> & op1, const CoinDenseVector< T> & op2) [inline]`

Return the difference of two dense vectors.

Definition at line 228 of file CoinDenseVector.hpp.

10.5.1.4 `template<typename T> CoinDenseVector<T> operator* (const CoinDenseVector< T> & op1, const CoinDenseVector< T> & op2) [inline]`

Return the element-wise product of two dense vectors.

Definition at line 244 of file CoinDenseVector.hpp.

10.5.1.5 `template<typename T> CoinDenseVector<T> operator/ (const CoinDenseVector< T> & op1, const CoinDenseVector< T> & op2) [inline]`

Return the element-wise ratio of two dense vectors.

Definition at line 259 of file CoinDenseVector.hpp.

10.5.1.6 `template<typename T> CoinDenseVector<T> operator+ (const CoinDenseVector< T> & op1, T value) [inline]`

Return the sum of a dense vector and a constant.

Definition at line 280 of file CoinDenseVector.hpp.

10.5.1.7 `template<typename T> CoinDenseVector<T> operator- (const CoinDenseVector< T> & op1, T value) [inline]`

Return the difference of a dense vector and a constant.

Definition at line 293 of file CoinDenseVector.hpp.

10.5.1.8 `template<typename T> CoinDenseVector<T> operator* (const CoinDenseVector< T> & op1, T value) [inline]`

Return the element-wise product of a dense vector and a constant.

Definition at line 306 of file CoinDenseVector.hpp.

10.5.1.9 `template<typename T> CoinDenseVector<T> operator/ (const CoinDenseVector< T> & op1, T value) [inline]`

Return the element-wise ratio of a dense vector and a constant.

Definition at line 319 of file CoinDenseVector.hpp.

10.5.1.10 `template<typename T> CoinDenseVector<T> operator+ (T value, const CoinDenseVector< T> & op1) [inline]`

Return the sum of a constant and a dense vector.

Definition at line 332 of file CoinDenseVector.hpp.

10.5.1.11 `template<typename T> CoinDenseVector<T> operator- (T value, const CoinDenseVector< T> & op1)`
`[inline]`

Return the difference of a constant and a dense vector.

Definition at line 345 of file CoinDenseVector.hpp.

10.5.1.12 `template<typename T> CoinDenseVector<T> operator* (T value, const CoinDenseVector< T> & op1)`
`[inline]`

Return the element-wise product of a constant and a dense vector.

Definition at line 358 of file CoinDenseVector.hpp.

10.5.1.13 `template<typename T> CoinDenseVector<T> operator/ (T value, const CoinDenseVector< T> & op1)`
`[inline]`

Return the element-wise ratio of a a constant and dense vector.

Definition at line 371 of file CoinDenseVector.hpp.

10.6 /home/ted/COIN/trunk/CoinUtils/src/CoinDistance.hpp File Reference

```
#include <iterator>
```

Functions

- `template<class ForwardIterator , class Distance>`
`void coinDistance` (ForwardIterator *first*, ForwardIterator *last*, Distance &*n*)
CoinDistance.
- `template<class ForwardIterator>`
`size_t coinDistance` (ForwardIterator *first*, ForwardIterator *last*)

10.6.1 Function Documentation

10.6.1.1 `template<class ForwardIterator , class Distance> void coinDistance (ForwardIterator first, ForwardIterator last, Distance & n)`

CoinDistance.

This is the [Coin](#) implementation of the `std::function` that is designed to work on multiple platforms.

Definition at line 24 of file CoinDistance.hpp.

10.6.1.2 `template<class ForwardIterator> size_t coinDistance (ForwardIterator first, ForwardIterator last)`

Definition at line 36 of file CoinDistance.hpp.

10.7 /home/ted/COIN/trunk/CoinUtils/src/CoinError.hpp File Reference

```
#include <string>
#include <iostream>
#include <cassert>
#include <cstring>
#include "CoinUtilsConfig.h"
#include "CoinPragma.hpp"
```

Classes

- class [CoinError](#)
Error Class thrown by an exception.

Macros

- `#define __STRING(x) #x`
- `#define __GNUC_PREREQ(maj, min) (0)`
- `#define CoinAssertDebug(expression) assert(expression)`
- `#define CoinAssertDebugHint(expression, hint) assert(expression)`
- `#define CoinAssert(expression) assert(expression)`
- `#define CoinAssertHint(expression, hint) assert(expression)`

Functions

- `void WindowsErrorPopupBlocker ()`
A function to block the popup windows that windows creates when the code crashes.
- `void CoinErrorUnitTest ()`
A function that tests the methods in the [CoinError](#) class.

10.7.1 Macro Definition Documentation

10.7.1.1 `#define __STRING(x) #x`

Definition at line 169 of file CoinError.hpp.

10.7.1.2 `#define __GNUC_PREREQ(maj, min) (0)`

Definition at line 173 of file CoinError.hpp.

10.7.1.3 `#define CoinAssertDebug(expression) assert(expression)`

Definition at line 177 of file CoinError.hpp.

10.7.1.4 `#define CoinAssertDebugHint(expression, hint) assert(expression)`

Definition at line 178 of file CoinError.hpp.

10.7.1.5 `#define CoinAssert(expression) assert(expression)`

Definition at line 179 of file CoinError.hpp.

10.7.1.6 `#define CoinAssertHint(expression, hint) assert(expression)`

Definition at line 180 of file CoinError.hpp.

10.7.2 Function Documentation

10.7.2.1 `void WindowsErrorPopupBlocker ()`

A function to block the popup windows that windows creates when the code crashes.

10.7.2.2 `void CoinErrorUnitTest ()`

A function that tests the methods in the [CoinError](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

10.8 /home/ted/COIN/trunk/CoinUtils/src/CoinFactorization.hpp File Reference

```
#include <iostream>
#include <string>
#include <cassert>
#include <cstdio>
#include <cmath>
#include "CoinTypes.hpp"
#include "CoinIndexedVector.hpp"
```

Classes

- class [CoinFactorization](#)

This deals with Factorization and Updates.

Macros

used by factorization

- `#define COINFACTORIZATION_BITS_PER_INT 32`
- `#define COINFACTORIZATION_SHIFT_PER_INT 5`
- `#define COINFACTORIZATION_MASK_PER_INT 0x1f`

10.8.1 Macro Definition Documentation

10.8.1.1 `#define COINFACTORIZATION_BITS_PER_INT 32`

Definition at line 698 of file CoinFactorization.hpp.

10.8.1.2 `#define COINFACTORIZATION_SHIFT_PER_INT 5`

Definition at line 699 of file CoinFactorization.hpp.

10.8.1.3 `#define COINFACTORIZATION_MASK_PER_INT 0x1f`

Definition at line 700 of file `CoinFactorization.hpp`.

10.9 `/home/ted/COIN/trunk/CoinUtils/src/CoinFileIO.hpp` File Reference

```
#include <string>
```

Classes

- class [CoinFileIOBase](#)
Base class for FileIO classes.
- class [CoinFileInput](#)
Abstract base class for file input classes.
- class [CoinFileOutput](#)
Abstract base class for file output classes.

10.10 `/home/ted/COIN/trunk/CoinUtils/src/CoinFinite.hpp` File Reference

```
#include <limits>
```

Functions

- bool [CoinFinite](#) (double val)
checks if a double value is finite (not infinity and not NaN)
- bool [CoinIsnan](#) (double val)
checks if a double value is not a number

Variables

- const double [COIN_DBL_MIN](#) = std::numeric_limits<double>::min()
- const double [COIN_DBL_MAX](#) = std::numeric_limits<double>::max()
- const int [COIN_INT_MAX](#) = std::numeric_limits<int>::max()
- const double [COIN_INT_MAX_AS_DOUBLE](#) = std::numeric_limits<int>::max()

10.10.1 Function Documentation

10.10.1.1 `bool CoinFinite (double val)`

checks if a double value is finite (not infinity and not NaN)

10.10.1.2 `bool CoinIsnan (double val)`

checks if a double value is not a number

10.10.2 Variable Documentation

10.10.2.1 `const double COIN_DBL_MIN = std::numeric_limits<double>::min()`

Definition at line 17 of file `CoinFinite.hpp`.

10.10.2.2 `const double COIN_DBL_MAX = std::numeric_limits<double>::max()`

Definition at line 18 of file `CoinFinite.hpp`.

10.10.2.3 `const int COIN_INT_MAX = std::numeric_limits<int>::max()`

Definition at line 19 of file `CoinFinite.hpp`.

10.10.2.4 `const double COIN_INT_MAX_AS_DOUBLE = std::numeric_limits<int>::max()`

Definition at line 20 of file `CoinFinite.hpp`.

10.11 /home/ted/COIN/trunk/CoinUtils/src/CoinFloatEqual.hpp File Reference

Function objects for testing equality of real numbers.

```
#include <algorithm>
#include <cmath>
#include "CoinFinite.hpp"
```

Classes

- class [CoinAbsFltEq](#)
Equality to an absolute tolerance.
- class [CoinRelFltEq](#)
Equality to a scaled tolerance.

10.11.1 Detailed Description

Function objects for testing equality of real numbers. Two objects are provided; one tests for equality to an absolute tolerance, one to a scaled tolerance. The tests will handle IEEE floating point, but note that infinity == infinity. Mathematicians are rolling in their graves, but this matches the behaviour for the common practice of using `DBL_MAX` (`numeric_limits<double>::max()`, or similar large finite number) as infinity.

Example usage:

```
double d1 = 3.14159 ;
double d2 = d1 ;
double d3 = d1+.0001 ;

CoinAbsFltEq eq1 ;
CoinAbsFltEq eq2(.001) ;

assert( eq1(d1,d2) ) ;
assert( !eq1(d1,d3) ) ;
assert( eq2(d1,d3) ) ;
```

[CoinRelFltEq](#) follows the same pattern.

Definition in file [CoinFloatEqual.hpp](#).

10.12 /home/ted/COIN/trunk/CoinUtils/src/CoinHelperFunctions.hpp File Reference

```
#include "CoinUtilsConfig.h"
#include <unistd.h>
#include <cstdlib>
#include <cstdio>
#include <algorithm>
#include "CoinTypes.hpp"
#include "CoinError.hpp"
```

Classes

- class [CoinThreadRandom](#)
Class for thread specific random numbers.

Macros

- #define [COIN_RESTRICT](#)
- #define [COIN_OWN_RANDOM_32](#)
- #define [CoinSizeofAsInt](#)(type) (static_cast<int>(sizeof(type)))
Cube Root.
- #define [COIN_DETAIL_PRINT](#)(s) {}

Functions

- template<class T >
[void CoinCopyN](#) (register const T *from, const int size, register T *to)
This helper function copies an array to another location using Duff's device (for a speedup of ~2).
- template<class T >
[void CoinCopy](#) (register const T *first, register const T *last, register T *to)
This helper function copies an array to another location using Duff's device (for a speedup of ~2).
- template<class T >
[void CoinDisjointCopyN](#) (register const T *from, const int size, register T *to)
This helper function copies an array to another location.
- template<class T >
[void CoinDisjointCopy](#) (register const T *first, register const T *last, register T *to)
This helper function copies an array to another location.
- template<class T >
T * [CoinCopyOfArray](#) (const T *array, const int size)
Return an array of length size filled with input from array, or null if array is null.
- template<class T >
T * [CoinCopyOfArrayPartial](#) (const T *array, const int size, const int copySize)
Return an array of length size filled with first copySize from array, or null if array is null.
- template<class T >
T * [CoinCopyOfArray](#) (const T *array, const int size, T value)
Return an array of length size filled with input from array, or filled with (scalar) value if array is null.
- template<class T >
T * [CoinCopyOfArrayOrZero](#) (const T *array, const int size)

Return an array of length `size` filled with input from `array`, or filled with zero if `array` is null.

- `template<class T >`
`void CoinMemcpyN` (register const T *from, const int size, register T *to)
This helper function copies an array to another location.
- `template<class T >`
`void CoinMemcpy` (register const T *first, register const T *last, register T *to)
This helper function copies an array to another location.
- `template<class T >`
`void CoinFillN` (register T *to, const int size, register const T value)
This helper function fills an array with a given value.
- `template<class T >`
`void CoinFill` (register T *first, register T *last, const T value)
This helper function fills an array with a given value.
- `template<class T >`
`void CoinZeroN` (register T *to, const int size)
This helper function fills an array with zero.
- `void CoinCheckDoubleZero` (double *to, const int size)
This Debug helper function checks an array is all zero.
- `void CoinCheckIntZero` (int *to, const int size)
This Debug helper function checks an array is all zero.
- `template<class T >`
`void CoinZero` (register T *first, register T *last)
This helper function fills an array with a given value.
- `char * CoinStrdup` (const char *name)
Returns strdup or NULL if original NULL.
- `template<class T >`
`T CoinMax` (register const T x1, register const T x2)
Return the larger (according to `operator<()` of the arguments.
- `template<class T >`
`T CoinMin` (register const T x1, register const T x2)
Return the smaller (according to `operator<()` of the arguments.
- `template<class T >`
`T CoinAbs` (const T value)
Return the absolute value of the argument.
- `template<class T >`
`bool CoinsSorted` (register const T *first, const int size)
This helper function tests whether the entries of an array are sorted according to `operator<`.
- `template<class T >`
`bool CoinsSorted` (register const T *first, register const T *last)
This helper function tests whether the entries of an array are sorted according to `operator<`.
- `template<class T >`
`void CoinlotaN` (register T *first, const int size, register T init)
This helper function fills an array with the values `init`, `init+1`, `init+2`, etc.
- `template<class T >`
`void Coinlota` (T *first, const T *last, T init)
This helper function fills an array with the values `init`, `init+1`, `init+2`, etc.
- `template<class T >`
`T * CoinDeleteEntriesFromArray` (register T *arrayFirst, register T *arrayLast, const int *firstDelPos, const int *lastDelPos)

This helper function deletes certain entries from an array.

- double `CoinDrand48` (bool isSeed=false, unsigned int seed=1)

Return a random number between 0 and 1.

- void `CoinSeedRandom` (int iseed)

Set the seed for the random number generator.

- char `CoinFindDirSeparator` ()

This function figures out whether file names should contain slashes or backslashes as directory separator.

- int `CoinStrNCaseCmp` (const char *s0, const char *s1, const size_t len)

- template<class T >

`void CoinSwap` (T &x, T &y)

Swap the arguments.

- template<class T >

int `CoinToFile` (const T *array, `CoinBigIndex` size, FILE *fp)

This helper function copies an array to file Returns 0 if OK, 1 if bad write.

- template<class T >

int `CoinFromFile` (T *&array, `CoinBigIndex` size, FILE *fp, `CoinBigIndex` &newSize)

This helper function copies an array from file and creates with new.

- int `CoinStrlenAsInt` (const char *string)

This helper returns "strlen" as an int.

10.12.1 Macro Definition Documentation

10.12.1.1 #define COIN_RESTRICT

Definition at line 30 of file `CoinHelperFunctions.hpp`.

10.12.1.2 #define COIN_OWN_RANDOM_32

Definition at line 753 of file `CoinHelperFunctions.hpp`.

10.12.1.3 #define CoinSizeofAsInt(type) (static_cast<int>(sizeof(type)))

Cube Root.

This helper returns "sizeof" as an int

Definition at line 940 of file `CoinHelperFunctions.hpp`.

10.12.1.4 #define COIN_DETAIL_PRINT(s) {}

Definition at line 1105 of file `CoinHelperFunctions.hpp`.

10.12.2 Function Documentation

10.12.2.1 template<class T > void CoinCopyN (register const T * from, const int size, register T * to) [inline]

This helper function copies an array to another location using Duff's device (for a speedup of ~2).

The arrays are given by pointers to their first entries and by the size of the source array. Overlapping arrays are handled correctly.

Definition at line 42 of file `CoinHelperFunctions.hpp`.

10.12.2.2 `template<class T> void CoinCopy (register const T * first, register const T * last, register T * to)` `[inline]`

This helper function copies an array to another location using Duff's device (for a speedup of ~2).

The source array is given by its first and "after last" entry; the target array is given by its first entry. Overlapping arrays are handled correctly.

All of the various CoinCopyN variants use an int for size. On 64-bit architectures, the address diff last-first will be a 64-bit quantity. Given that everything else uses an int, I'm going to choose to kick the difference down to int. –lh, 100823 –

Definition at line 100 of file CoinHelperFunctions.hpp.

10.12.2.3 `template<class T> void CoinDisjointCopyN (register const T * from, const int size, register T * to)` `[inline]`

This helper function copies an array to another location.

The two arrays must not overlap (otherwise an exception is thrown). For speed 8 entries are copied at a time. The arrays are given by pointers to their first entries and by the size of the source array.

Note JJF - the speed claim seems to be false on IA32 so I have added CoinMemcpyN which can be used for atomic data

Definition at line 115 of file CoinHelperFunctions.hpp.

10.12.2.4 `template<class T> void CoinDisjointCopy (register const T * first, register const T * last, register T * to)`
`[inline]`

This helper function copies an array to another location.

The two arrays must not overlap (otherwise an exception is thrown). For speed 8 entries are copied at a time. The source array is given by its first and "after last" entry; the target array is given by its first entry.

Definition at line 168 of file CoinHelperFunctions.hpp.

10.12.2.5 `template<class T> T* CoinCopyOfArray (const T * array, const int size)` `[inline]`

Return an array of length *size* filled with input from *array*, or null if *array* is null.

Definition at line 181 of file CoinHelperFunctions.hpp.

10.12.2.6 `template<class T> T* CoinCopyOfArrayPartial (const T * array, const int size, const int copySize)` `[inline]`

Return an array of length *size* filled with first *copySize* from *array*, or null if *array* is null.

Definition at line 198 of file CoinHelperFunctions.hpp.

10.12.2.7 `template<class T> T* CoinCopyOfArray (const T * array, const int size, T value)` `[inline]`

Return an array of length *size* filled with input from *array*, or filled with (scalar) *value* if *array* is null.

Definition at line 215 of file CoinHelperFunctions.hpp.

10.12.2.8 `template<class T> T* CoinCopyOfArrayOrZero (const T * array, const int size)` `[inline]`

Return an array of length *size* filled with input from *array*, or filled with zero if *array* is null.

Definition at line 234 of file CoinHelperFunctions.hpp.

10.12.2.9 `template<class T> void CoinMemcpyN (register const T * from, const int size, register T * to)` `[inline]`

This helper function copies an array to another location.

The two arrays must not overlap (otherwise an exception is thrown). For speed 8 entries are copied at a time. The

arrays are given by pointers to their first entries and by the size of the source array.

Note JJF - the speed claim seems to be false on IA32 so I have added alternative coding if USE_MEMCPY defined
Definition at line 257 of file CoinHelperFunctions.hpp.

10.12.2.10 `template<class T> void CoinMemcpy (register const T * first, register const T * last, register T * to) [inline]`

This helper function copies an array to another location.

The two arrays must not overlap (otherwise an exception is thrown). For speed 8 entries are copied at a time. The source array is given by its first and "after last" entry; the target array is given by its first entry.

Definition at line 344 of file CoinHelperFunctions.hpp.

10.12.2.11 `template<class T> void CoinFillN (register T * to, const int size, register const T value) [inline]`

This helper function fills an array with a given value.

For speed 8 entries are filled at a time. The array is given by a pointer to its first entry and its size.

Note JJF - the speed claim seems to be false on IA32 so I have added CoinZero to allow for memset.

Definition at line 359 of file CoinHelperFunctions.hpp.

10.12.2.12 `template<class T> void CoinFill (register T * first, register T * last, const T value) [inline]`

This helper function fills an array with a given value.

For speed 8 entries are filled at a time. The array is given by its first and "after last" entry.

Definition at line 414 of file CoinHelperFunctions.hpp.

10.12.2.13 `template<class T> void CoinZeroN (register T * to, const int size) [inline]`

This helper function fills an array with zero.

For speed 8 entries are filled at a time. The array is given by a pointer to its first entry and its size.

Note JJF - the speed claim seems to be false on IA32 so I have allowed for memset as an alternative

Definition at line 428 of file CoinHelperFunctions.hpp.

10.12.2.14 `void CoinCheckDoubleZero (double * to, const int size) [inline]`

This Debug helper function checks an array is all zero.

Definition at line 489 of file CoinHelperFunctions.hpp.

10.12.2.15 `void CoinCheckIntZero (int * to, const int size) [inline]`

This Debug helper function checks an array is all zero.

Definition at line 502 of file CoinHelperFunctions.hpp.

10.12.2.16 `template<class T> void CoinZero (register T * first, register T * last) [inline]`

This helper function fills an array with a given value.

For speed 8 entries are filled at a time. The array is given by its first and "after last" entry.

Definition at line 520 of file CoinHelperFunctions.hpp.

10.12.2.17 `char* CoinStrdup (const char * name) [inline]`

Returns strdup or NULL if original NULL.

Definition at line 528 of file CoinHelperFunctions.hpp.

10.12.2.18 `template<class T > T CoinMax (register const T x1, register const T x2) [inline]`

Return the larger (according to `operator<()` of the arguments.

This function was introduced because for some reason compiler tend to handle the `max()` function differently.

Definition at line 546 of file CoinHelperFunctions.hpp.

10.12.2.19 `template<class T > T CoinMin (register const T x1, register const T x2) [inline]`

Return the smaller (according to `operator<()` of the arguments.

This function was introduced because for some reason compiler tend to handle the `min()` function differently.

Definition at line 557 of file CoinHelperFunctions.hpp.

10.12.2.20 `template<class T > T CoinAbs (const T value) [inline]`

Return the absolute value of the argument.

This function was introduced because for some reason compiler tend to handle the `abs()` function differently.

Definition at line 568 of file CoinHelperFunctions.hpp.

10.12.2.21 `template<class T > bool CoinsSorted (register const T * first, const int size) [inline]`

This helper function tests whether the entries of an array are sorted according to `operator<`.

The array is given by a pointer to its first entry and by its size.

Definition at line 579 of file CoinHelperFunctions.hpp.

10.12.2.22 `template<class T > bool CoinsSorted (register const T * first, register const T * last) [inline]`

This helper function tests whether the entries of an array are sorted according to `operator<`.

The array is given by its first and "after last" entry.

Definition at line 628 of file CoinHelperFunctions.hpp.

10.12.2.23 `template<class T > void CoinlotaN (register T * first, const int size, register T init) [inline]`

This helper function fills an array with the values `init`, `init+1`, `init+2`, etc.

For speed 8 entries are filled at a time. The array is given by a pointer to its first entry and its size.

Definition at line 639 of file CoinHelperFunctions.hpp.

10.12.2.24 `template<class T > void Coinlota (T * first, const T * last, T init) [inline]`

This helper function fills an array with the values `init`, `init+1`, `init+2`, etc.

For speed 8 entries are filled at a time. The array is given by its first and "after last" entry.

Definition at line 694 of file CoinHelperFunctions.hpp.

10.12.2.25 `template<class T> T* CoinDeleteEntriesFromArray (register T * arrayFirst, register T * arrayLast, const int * firstDelPos, const int * lastDelPos) [inline]`

This helper function deletes certain entries from an array.

The array is given by pointers to its first and "after last" entry (first two arguments). The positions of the entries to be deleted are given in the integer array specified by the last two arguments (again, first and "after last" entry).

Definition at line 707 of file CoinHelperFunctions.hpp.

10.12.2.26 `double CoinDrand48 (bool isSeed = false, unsigned int seed = 1) [inline]`

Return a random number between 0 and 1.

A platform-independent linear congruential generator. For a given seed, the generated sequence is always the same regardless of the (32-bit) architecture. This allows to build & test in different environments, getting in most cases the same optimization path.

Set *isSeed* to true and supply an integer seed to set the seed (vid. [CoinSeedRandom](#))

Todo Anyone want to volunteer an upgrade for 64-bit architectures?

Definition at line 771 of file CoinHelperFunctions.hpp.

10.12.2.27 `void CoinSeedRandom (int iseed) [inline]`

Set the seed for the random number generator.

Definition at line 784 of file CoinHelperFunctions.hpp.

10.12.2.28 `char CoinFindDirSeparator () [inline]`

This function figures out whether file names should contain slashes or backslashes as directory separator.

Definition at line 813 of file CoinHelperFunctions.hpp.

10.12.2.29 `int CoinStrNCaseCmp (const char * s0, const char * s1, const size_t len) [inline]`

Definition at line 833 of file CoinHelperFunctions.hpp.

10.12.2.30 `template<class T> void CoinSwap (T & x, T & y) [inline]`

Swap the arguments.

Definition at line 856 of file CoinHelperFunctions.hpp.

10.12.2.31 `template<class T> int CoinToFile (const T * array, CoinBigIndex size, FILE * fp) [inline]`

This helper function copies an array to file Returns 0 if OK, 1 if bad write.

Definition at line 870 of file CoinHelperFunctions.hpp.

10.12.2.32 `template<class T> int CoinFromFile (T *& array, CoinBigIndex size, FILE * fp, CoinBigIndex & newSize) [inline]`

This helper function copies an array from file and creates with new.

Passed in array is ignored i.e. not deleted. But if NULL and size does not match and newSize 0 then leaves as NULL and 0 Returns 0 if OK, 1 if bad read, 2 if size did not match.

Definition at line 901 of file CoinHelperFunctions.hpp.

10.12.2.33 `int CoinStrlenAsInt (const char * string) [inline]`

This helper returns "strlen" as an int.

Definition at line 943 of file CoinHelperFunctions.hpp.

10.13 /home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.hpp File Reference

```
#include <map>
#include "CoinFinite.hpp"
#include "CoinPackedVectorBase.hpp"
#include "CoinSort.hpp"
#include "CoinHelperFunctions.hpp"
#include <cassert>
```

Classes

- class [CoinIndexedVector](#)
Indexed Vector.
- class [CoinArrayWithLength](#)
Pointer with length in bytes.
- class [CoinDoubleArrayWithLength](#)
*double * version*
- class [CoinFactorizationDoubleArrayWithLength](#)
*CoinFactorizationDouble * version.*
- class [CoinFactorizationLongDoubleArrayWithLength](#)
*CoinFactorizationLongDouble * version.*
- class [CoinIntArrayWithLength](#)
*int * version*
- class [CoinBigIndexArrayWithLength](#)
*CoinBigIndex * version.*
- class [CoinUnsignedIntArrayWithLength](#)
*unsigned int * version*
- class [CoinVoidStarArrayWithLength](#)
*void * version*
- class [CoinArbitraryArrayWithLength](#)
arbitrary version
- class [CoinPartitionedVector](#)

Macros

- `#define COIN_INDEXED_TINY_ELEMENT 1.0e-50`
- `#define COIN_INDEXED_REALLY_TINY_ELEMENT 1.0e-100`
- `#define COIN_PARTITIONS 8`

Functions

- [void CoinIndexedVectorUnitTest \(\)](#)
A function that tests the methods in the [CoinIndexedVector](#) class.

10.13.1 Macro Definition Documentation

10.13.1.1 `#define COIN_INDEXED_TINY_ELEMENT 1.0e-50`

Definition at line 24 of file `CoinIndexedVector.hpp`.

10.13.1.2 `#define COIN_INDEXED_REALLY_TINY_ELEMENT 1.0e-100`

Definition at line 25 of file `CoinIndexedVector.hpp`.

10.13.1.3 `#define COIN_PARTITIONS 8`

Definition at line 1059 of file `CoinIndexedVector.hpp`.

10.13.2 Function Documentation

10.13.2.1 `void CoinIndexedVectorUnitTest ()`

A function that tests the methods in the [CoinIndexedVector](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

10.14 `/home/ted/COIN/trunk/CoinUtils/src/CoinLpIO.hpp` File Reference

```
#include <cstdio>
#include "CoinPackedMatrix.hpp"
#include "CoinMessage.hpp"
```

Classes

- class [CoinLpIO](#)
Class to read and write Lp files.
- struct [CoinLpIO::CoinHashLink](#)

Typedefs

- typedef int [COINColumnIndex](#)

Functions

- `void CoinLpIOUnitTest (const std::string &lpDir)`

10.14.1 Typedef Documentation

10.14.1.1 typedef int `COINColumnIndex`

Definition at line 23 of file `CoinLpIO.hpp`.

10.14.2 Function Documentation

10.14.2.1 void CoinLpIOUnitTest (const std::string & lpDir)

10.15 /home/ted/COIN/trunk/CoinUtils/src/CoinMessage.hpp File Reference

This file contains the enum for the standard set of [Coin](#) messages and a class definition whose sole purpose is to supply a constructor.

```
#include "CoinMessageHandler.hpp"
```

Classes

- class [CoinMessage](#)
The standard set of [Coin](#) messages.

Enumerations

- enum [COIN_Message](#) {
[COIN_MPS_LINE](#) =0, [COIN_MPS_STATS](#), [COIN_MPS_ILLEGAL](#), [COIN_MPS_BADIMAGE](#),
[COIN_MPS_DUPOBJ](#), [COIN_MPS_DUPROW](#), [COIN_MPS_NOMATCHROW](#), [COIN_MPS_NOMATCHCOL](#),
[COIN_MPS_FILE](#), [COIN_MPS_BADFILE1](#), [COIN_MPS_BADFILE2](#), [COIN_MPS_EOF](#),
[COIN_MPS_RETURNING](#), [COIN_MPS_CHANGED](#), [COIN_SOLVER_MPS](#), [COIN_PREOLVE_COLINFEAS](#),
[COIN_PREOLVE_ROWINFEAS](#), [COIN_PREOLVE_COLUMNBOUNDA](#), [COIN_PREOLVE_COLUMNBOU-](#)
[NDB](#), [COIN_PREOLVE_NONOPTIMAL](#),
[COIN_PREOLVE_STATS](#), [COIN_PREOLVE_INFEAS](#), [COIN_PREOLVE_UNBOUND](#), [COIN_PREOLVE-](#)
[_INFEASUNBOUND](#),
[COIN_PREOLVE_INTEGERMODS](#), [COIN_PREOLVE_POSTSOLVE](#), [COIN_PREOLVE_NEEDS_CLEANI-](#)
[NG](#), [COIN_PREOLVE_PASS](#),
[COIN_GENERAL_INFO](#), [COIN_GENERAL_WARNING](#), [COIN_DUMMY_END](#) }

Symbolic names for the standard set of COIN messages.

10.15.1 Detailed Description

This file contains the enum for the standard set of [Coin](#) messages and a class definition whose sole purpose is to supply a constructor. The text of the messages is defined in [CoinMessage.cpp](#),

[CoinMessageHandler.hpp](#) contains the generic facilities for message handling.

Definition in file [CoinMessage.hpp](#).

10.15.2 Enumeration Type Documentation

10.15.2.1 enum [COIN_Message](#)

Symbolic names for the standard set of COIN messages.

Enumerator

[COIN_MPS_LINE](#)

[COIN_MPS_STATS](#)

COIN_MPS_ILLEGAL
COIN_MPS_BADIMAGE
COIN_MPS_DUPOBJ
COIN_MPS_DUPROW
COIN_MPS_NOMATCHROW
COIN_MPS_NOMATCHCOL
COIN_MPS_FILE
COIN_MPS_BADFILE1
COIN_MPS_BADFILE2
COIN_MPS_EOF
COIN_MPS_RETURNING
COIN_MPS_CHANGED
COIN_SOLVER_MPS
COIN_PRESOLVE_COLINFEAS
COIN_PRESOLVE_ROWINFEAS
COIN_PRESOLVE_COLUMNBOUND A
COIN_PRESOLVE_COLUMNBOUND B
COIN_PRESOLVE_NONOPTIMAL
COIN_PRESOLVE_STATS
COIN_PRESOLVE_INFEAS
COIN_PRESOLVE_UNBOUND
COIN_PRESOLVE_INFEASUNBOUND
COIN_PRESOLVE_INTEGERMODS
COIN_PRESOLVE_POSTSOLVE
COIN_PRESOLVE_NEEDS_CLEANING
COIN_PRESOLVE_PASS
COIN_GENERAL_INFO
COIN_GENERAL_WARNING
COIN_DUMMY_END

Definition at line 28 of file CoinMessage.hpp.

10.16 /home/ted/COIN/trunk/CoinUtils/src/CoinMessageHandler.hpp File Reference

This is a first attempt at a message handler.

```
#include "CoinUtilsConfig.h"  
#include "CoinPragma.hpp"  
#include <iostream>  
#include <cstdio>  
#include <string>  
#include <vector>
```


Classes

- class [CoinOneMessage](#)
Class for one massaged message.
- class [CoinMessages](#)
Class to hold and manipulate an array of massaged messages.
- class [CoinMessageHandler](#)
Base class for message handling.

Macros

- `#define COIN_NUM_LOG 4`
Log levels will be by type and will then use type given in [CoinMessage::class_](#).
- `#define COIN_MESSAGE_HANDLER_MAX_BUFFER_SIZE 1000`
Maximum length of constructed message (characters)

Enumerations

- enum [CoinMessageMarker](#) { [CoinMessageEol](#) = 0, [CoinMessageNewline](#) = 1 }

Functions

- bool [CoinMessageHandlerUnitTest](#) ()
A function that tests the methods in the [CoinMessageHandler](#) class.

10.16.1 Detailed Description

This is a first attempt at a message handler. The COIN Project is in favo(u)r of multi-language support. This implementation of a message handler tries to make it as lightweight as possible in the sense that only a subset of messages need to be defined — the rest default to US English.

The default handler at present just prints to stdout or to a FILE pointer

Todo This needs to be worked over for correct operation with ISO character codes.

Definition in file [CoinMessageHandler.hpp](#).

10.16.2 Macro Definition Documentation

10.16.2.1 `#define COIN_NUM_LOG 4`

Log levels will be by type and will then use type given in [CoinMessage::class_](#).

- 0 - Branch and bound code or similar
- 1 - Solver
- 2 - Stuff in [Coin](#) directory
- 3 - Cut generators

Definition at line 586 of file [CoinMessageHandler.hpp](#).

10.16.2.2 `#define COIN_MESSAGE_HANDLER_MAX_BUFFER_SIZE 1000`

Maximum length of constructed message (characters)

Definition at line 588 of file `CoinMessageHandler.hpp`.

10.16.3 Enumeration Type Documentation

10.16.3.1 `enum CoinMessageMarker`

Enumerator

CoinMessageEol

CoinMessageNewline

Definition at line 217 of file `CoinMessageHandler.hpp`.

10.16.4 Function Documentation

10.16.4.1 `bool CoinMessageHandlerUnitTest ()`

A function that tests the methods in the [CoinMessageHandler](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

10.17 `/home/ted/COIN/trunk/CoinUtils/src/CoinModel.hpp` File Reference

```
#include "CoinModelUseful.hpp"
#include "CoinPackedMatrix.hpp"
#include "CoinFinite.hpp"
```

Classes

- class [CoinBaseModel](#)
- class [CoinModel](#)

This is a simple minded model which is stored in a format which makes it easier to construct and modify but not efficient for algorithms.

Functions

- double [getFunctionValueFromString](#) (const char *string, const char *x, double xValue)
Just function of single variable x.
- double [getDoubleFromString](#) ([CoinYacc](#) &info, const char *string, const char *x, double xValue)
faster version

10.17.1 Function Documentation

10.17.1.1 double `getFunctionValueFromString` (const char * *string*, const char * *x*, double *xValue*)Just function of single variable *x*.10.17.1.2 double `getDoubleFromString` (CoinYacc & *info*, const char * *string*, const char * *x*, double *xValue*)

faster version

10.18 /home/ted/COIN/trunk/CoinUtils/src/CoinModelUseful.hpp File Reference

```
#include <cstdlib>
#include <cmath>
#include <cassert>
#include <cfloat>
#include <cstring>
#include <cstdio>
#include <iostream>
#include "CoinPragma.hpp"
```

Classes

- class [CoinModelLink](#)
This is for various structures/classes needed by [CoinModel](#).
- struct [CoinModelTriple](#)
for linked lists
- struct [CoinModelHashLink](#)
for names and hashing
- struct [symrec](#)
For string evaluation.
- class [CoinYacc](#)
- class [CoinModelHash](#)
- class [CoinModelHash2](#)
For int,int hashing.
- class [CoinModelLinkedList](#)

Typedefs

- typedef double(* [func_t](#))(double)
- typedef struct [symrec](#) [symrec](#)

Functions

- int [rowInTriple](#) (const [CoinModelTriple](#) &triple)
- void [setRowInTriple](#) ([CoinModelTriple](#) &triple, int iRow)
- bool [stringInTriple](#) (const [CoinModelTriple](#) &triple)
- void [setStringInTriple](#) ([CoinModelTriple](#) &triple, bool string)
- void [setRowAndStringInTriple](#) ([CoinModelTriple](#) &triple, int iRow, bool string)

10.18.1 Typedef Documentation

10.18.1.1 `typedef double(* func_t)(double)`

Definition at line 133 of file CoinModelUseful.hpp.

10.18.1.2 `typedef struct symrec symrec`

Definition at line 149 of file CoinModelUseful.hpp.

10.18.2 Function Documentation

10.18.2.1 `int rowInTriple (const CoinModelTriple & triple) [inline]`

Definition at line 115 of file CoinModelUseful.hpp.

10.18.2.2 `void setRowInTriple (CoinModelTriple & triple, int iRow) [inline]`

Definition at line 117 of file CoinModelUseful.hpp.

10.18.2.3 `bool stringInTriple (const CoinModelTriple & triple) [inline]`

Definition at line 119 of file CoinModelUseful.hpp.

10.18.2.4 `void setStringInTriple (CoinModelTriple & triple, bool string) [inline]`

Definition at line 121 of file CoinModelUseful.hpp.

10.18.2.5 `void setRowAndStringInTriple (CoinModelTriple & triple, int iRow, bool string) [inline]`

Definition at line 123 of file CoinModelUseful.hpp.

10.19 /home/ted/COIN/trunk/CoinUtils/src/CoinMpsIO.hpp File Reference

```
#include <vector>
#include <string>
#include "CoinUtilsConfig.h"
#include "CoinPackedMatrix.hpp"
#include "CoinMessageHandler.hpp"
#include "CoinFileIO.hpp"
```

Classes

- class [CoinMpsCardReader](#)
Very simple code for reading MPS data.
- class [CoinSet](#)
Very simple class for containing data on set.
- class [CoinSosSet](#)
Very simple class for containing SOS set.
- class [CoinMpsIO](#)
MPS IO Interface.
- struct [CoinMpsIO::CoinHashLink](#)

Macros

- `#define COIN_MAX_FIELD_LENGTH 160`
- `#define MAX_CARD_LENGTH 5*COIN_MAX_FIELD_LENGTH+80`

Typedefs

- typedef int [COINColumnIndex](#)
The following lengths are in decreasing order (for 64 bit etc) Large enough to contain element index This is already defined as CoinBigIndex Large enough to contain column index.
- typedef int [COINRowIndex](#)
Large enough to contain row index (or basis)

Enumerations

- enum [COINSectionType](#) {
[COIN_NO_SECTION](#), [COIN_NAME_SECTION](#), [COIN_ROW_SECTION](#), [COIN_COLUMN_SECTION](#),
[COIN_RHS_SECTION](#), [COIN_RANGES_SECTION](#), [COIN_BOUNDS_SECTION](#), [COIN_ENDATA_SECTION](#),
[COIN_EOF_SECTION](#), [COIN_QUADRATIC_SECTION](#), [COIN_CONIC_SECTION](#), [COIN_QUAD_SECTION](#),
[COIN_SOS_SECTION](#), [COIN_BASIS_SECTION](#), [COIN_UNKNOWN_SECTION](#) }
- enum [COINMpsType](#) {
[COIN_N_ROW](#), [COIN_E_ROW](#), [COIN_L_ROW](#), [COIN_G_ROW](#),
[COIN_BLANK_COLUMN](#), [COIN_S1_COLUMN](#), [COIN_S2_COLUMN](#), [COIN_S3_COLUMN](#),
[COIN_INTORG](#), [COIN_INTEND](#), [COIN_SOSEND](#), [COIN_UNSET_BOUND](#),
[COIN_UP_BOUND](#), [COIN_FX_BOUND](#), [COIN_LO_BOUND](#), [COIN_FR_BOUND](#),
[COIN_MI_BOUND](#), [COIN_PL_BOUND](#), [COIN_BV_BOUND](#), [COIN_UI_BOUND](#),
[COIN_LI_BOUND](#), [COIN_BOTH_BOUNDS_SET](#), [COIN_SC_BOUND](#), [COIN_S1_BOUND](#),
[COIN_S2_BOUND](#), [COIN_BS_BASIS](#), [COIN_XL_BASIS](#), [COIN_XU_BASIS](#),
[COIN_LL_BASIS](#), [COIN_UL_BASIS](#), [COIN_UNKNOWN_MPS_TYPE](#) }

Functions

- [void CoinMpsIOUnitTest](#) (const std::string &mpsDir)
A function that tests the methods in the [CoinMpsIO](#) class.
- [void CoinConvertDouble](#) (int section, int formatType, double value, char outputValue[24])

10.19.1 Macro Definition Documentation

10.19.1.1 `#define COIN_MAX_FIELD_LENGTH 160`

Definition at line 35 of file CoinMpslO.hpp.

10.19.1.2 `#define MAX_CARD_LENGTH 5*COIN_MAX_FIELD_LENGTH+80`

Definition at line 37 of file CoinMpslO.hpp.

10.19.2 Typedef Documentation

10.19.2.1 typedef int COINColumnIndex

The following lengths are in decreasing order (for 64 bit etc) Large enough to contain element index This is already defined as CoinBigIndex Large enough to contain column index.

Definition at line 21 of file CoinMpsIO.hpp.

10.19.2.2 typedef int COINRowIndex

Large enough to contain row index (or basis)

Definition at line 30 of file CoinMpsIO.hpp.

10.19.3 Enumeration Type Documentation

10.19.3.1 enum COINSectionType

Enumerator

COIN_NO_SECTION
COIN_NAME_SECTION
COIN_ROW_SECTION
COIN_COLUMN_SECTION
COIN_RHS_SECTION
COIN_RANGES_SECTION
COIN_BOUNDS_SECTION
COIN_ENDATA_SECTION
COIN_EOF_SECTION
COIN_QUADRATIC_SECTION
COIN_CONIC_SECTION
COIN_QUAD_SECTION
COIN_SOS_SECTION
COIN_BASIS_SECTION
COIN_UNKNOWN_SECTION

Definition at line 39 of file CoinMpsIO.hpp.

10.19.3.2 enum COINMpsType

Enumerator

COIN_N_ROW
COIN_E_ROW
COIN_L_ROW
COIN_G_ROW
COIN_BLANK_COLUMN
COIN_S1_COLUMN
COIN_S2_COLUMN
COIN_S3_COLUMN
COIN_INTORG

COIN_INTEND
COIN_SOSEND
COIN_UNSET_BOUND
COIN_UP_BOUND
COIN_FX_BOUND
COIN_LO_BOUND
COIN_FR_BOUND
COIN_MI_BOUND
COIN_PL_BOUND
COIN_BV_BOUND
COIN_UI_BOUND
COIN_LI_BOUND
COIN_BOTH_BOUNDS_SET
COIN_SC_BOUND
COIN_S1_BOUND
COIN_S2_BOUND
COIN_BS_BASIS
COIN_XL_BASIS
COIN_XU_BASIS
COIN_LL_BASIS
COIN_UL_BASIS
COIN_UNKNOWN_MPS_TYPE

Definition at line 47 of file CoinMpsIO.hpp.

10.19.4 Function Documentation

10.19.4.1 void CoinMpsIOUnitTest (const std::string & mpsDir)

A function that tests the methods in the [CoinMpsIO](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging. Also, if this method is compiled with optimization, the compilation takes 10-15 minutes and the machine pages (has 256M core memory!)...

10.19.4.2 void CoinConvertDouble (int section, int formatType, double value, char outputValue[24])

10.20 /home/ted/COIN/trunk/CoinUtils/src/CoinOslC.h File Reference

```

#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "CoinHelperFunctions.hpp"
#include <stddef.h>
#include <assert.h>

```

Macros

- `#define CLP_OSL 0`
- `#define C_EKK_GO_SPARSE 200`
- `#define SPARSE_UPDATE`
- `#define NO_SHIFT`
- `#define c_ekkscpy_0_1(s, ival, array) CoinFillN(array,s,ival)`
- `#define c_ekks1cpy(n, marr1, marr2) CoinMemcpyN(marr1,n, marr2)`
- `#define SLACK_VALUE -1.0`
- `#define C_EKK_REMOVE_LINK(hpiv, hin, link, ipivot)`
- `#define C_EKK_ADD_LINK(hpiv, nzi, link, npr)`
- `#define SHIFT_INDEX(limit) (limit)`
- `#define UNSHIFT_INDEX(limit) (limit)`
- `#define SHIFT_REF(arr, ind) (arr)[ind]`
- `#define NOT_ZERO(x) ((x) != 0.0)`
- `#define SWAP(type, _x, _y) { type _tmp = (_x); (_x) = (_y); (_y) = _tmp;}`
- `#define UNROLL_LOOP_BODY1(code) {{code}}`
- `#define UNROLL_LOOP_BODY2(code) {{code} {code}}`
- `#define UNROLL_LOOP_BODY4(code) {{code} {code} {code} {code}}`

Functions

- `int c_ekkbtrn` (register const `EKKfactinfo` *fact, double *dwork1, int *mpt, int first_nonzero)
- `int c_ekkbtrn_ipivrw` (register const `EKKfactinfo` *fact, double *dwork1, int *mpt, int ipivrw, int *spare)
- `int c_ekketsj` (register `EKKfactinfo` *fact, double *dwork1, int *mpt2, double dalpha, int orig_nincol, int npivot, int *nuspikp, const int ipivrw, int *spare)
- `int c_ekkftrn` (register const `EKKfactinfo` *fact, double *dwork1, double *dpermu, int *mpt, int numberNonZero)
- `int c_ekkftrn_ft` (register `EKKfactinfo` *fact, double *dwork1, int *mpt, int *nincolp)
- `void c_ekkftrn2` (register `EKKfactinfo` *fact, double *dwork1, double *dpermu1, int *mpt1, int *nincolp, double *dwork1_ft, int *mpt_ft, int *nincolp_ft)
- `int c_ekklfct` (register `EKKfactinfo` *fact)
- `int c_ekkslcf` (register const `EKKfactinfo` *fact)
- `void c_ekkscpy` (int n, const int *marr1, int *marr2)
- `void c_ekkdcpy` (int n, const double *marr1, double *marr2)
- `int c_ekk_IsSet` (const int *array, int bit)
- `void c_ekk_Set` (int *array, int bit)
- `void c_ekk_Unset` (int *array, int bit)
- `void c_ekkzero` (int length, int n, void *array)
- `void c_ekkdzero` (int n, double *marray)
- `void c_ekkizero` (int n, int *marray)
- `void c_ekkczero` (int n, char *marray)
- `void clp_setup_pointers` (`EKKfactinfo` *fact)
- `void clp_memory` (int type)
- `double * clp_double` (int number_entries)
- `int * clp_int` (int number_entries)
- `void * clp_malloc` (int number_entries)
- `void clp_free` (void *oldArray)

10.20.1 Macro Definition Documentation

10.20.1.1 `#define CLP_OSL 0`

Definition at line 12 of file CoinOslC.h.

10.20.1.2 `#define C_EKK_GO_SPARSE 200`

Definition at line 14 of file CoinOslC.h.

10.20.1.3 `#define SPARSE_UPDATE`

Definition at line 28 of file CoinOslC.h.

10.20.1.4 `#define NO_SHIFT`

Definition at line 29 of file CoinOslC.h.

10.20.1.5 `#define c_ekkscpy_0_1(s, ival, array) CoinFillN(array,s,ival)`

Definition at line 80 of file CoinOslC.h.

10.20.1.6 `#define c_ekks1cpy(n, marr1, marr2) CoinMemcpyN(marr1,n, marr2)`

Definition at line 81 of file CoinOslC.h.

10.20.1.7 `#define SLACK_VALUE -1.0`

Definition at line 89 of file CoinOslC.h.

10.20.1.8 `#define C_EKK_REMOVE_LINK(hpiv, hin, link, ipivot)`

Value:

```
{
    int ipre = link[ipivot].pre;
    int isuc = link[ipivot].suc;
    if (ipre > 0) {
        link[ipre].suc = isuc;
    }
    if (ipre <= 0) {
        hpiv[hin[ipivot]] = isuc;
    }
    if (isuc > 0) {
        link[isuc].pre = ipre;
    }
}
```

Definition at line 90 of file CoinOslC.h.

10.20.1.9 `#define C_EKK_ADD_LINK(hpiv, nzi, link, npr)`

Value:

```
{
    int ifiri = hpiv[nzi];
    hpiv[nzi] = npr;
    link[npr].suc = ifiri;
    link[npr].pre = 0;
    if (ifiri != 0) {
        link[ifiri].pre = npr;
    }
}
```

Definition at line 105 of file CoinOslC.h.

10.20.1.10 `#define SHIFT_INDEX(limit) (limit)`

Definition at line 118 of file CoinOslC.h.

10.20.1.11 `#define UNSHIFT_INDEX(limit) (limit)`

Definition at line 119 of file CoinOslC.h.

10.20.1.12 `#define SHIFT_REF(arr, ind) (arr)[ind]`

Definition at line 120 of file CoinOslC.h.

10.20.1.13 `#define NOT_ZERO(x) ((x) != 0.0)`

Definition at line 133 of file CoinOslC.h.

10.20.1.14 `#define SWAP(type, _x, _y) { type _tmp = (_x); (_x) = (_y); (_y) = _tmp;}`

Definition at line 136 of file CoinOslC.h.

10.20.1.15 `#define UNROLL_LOOP_BODY1(code) {{code}}`

Definition at line 138 of file CoinOslC.h.

10.20.1.16 `#define UNROLL_LOOP_BODY2(code) {{code} {code}}`

Definition at line 140 of file CoinOslC.h.

10.20.1.17 `#define UNROLL_LOOP_BODY4(code) {{code} {code} {code} {code}}`

Definition at line 142 of file CoinOslC.h.

10.20.2 Function Documentation

10.20.2.1 `int c_ekkbtrn (register const EKKfactinfo * fact, double * dwork1, int * mpt, int first_nonzero)`

10.20.2.2 `int c_ekkbtrn_ipivrw (register const EKKfactinfo * fact, double * dwork1, int * mpt, int ipivrw, int * spare)`

10.20.2.3 `int c_ekketsj (register EKKfactinfo * fact, double * dwork1, int * mpt2, double dalpha, int orig_nincol, int npivot, int * nuspiqp, const int ipivrw, int * spare)`

10.20.2.4 `int c_ekkftrn (register const EKKfactinfo * fact, double * dwork1, double * dpermu, int * mpt, int numberNonZero)`

10.20.2.5 `int c_ekkftrn_ft (register EKKfactinfo * fact, double * dwork1, int * mpt, int * nincolp)`

10.20.2.6 `void c_ekkftrn2 (register EKKfactinfo * fact, double * dwork1, double * dpermu1, int * mpt1, int * nincolp, double * dwork1_ft, int * mpt_ft, int * nincolp_ft)`

10.20.2.7 `int c_ekklfct (register EKKfactinfo * fact)`

10.20.2.8 `int c_ekkslcf (register const EKKfactinfo * fact)`

10.20.2.9 `void c_ekkscpy (int n, const int * marr1, int * marr2) [inline]`

Definition at line 61 of file CoinOslC.h.

10.20.2.10 `void c_ekkdcp (int n, const double * marr1, double * marr2)` `[inline]`

Definition at line 63 of file CoinOslC.h.

10.20.2.11 `int c_ekk_IsSet (const int * array, int bit)`

10.20.2.12 `void c_ekk_Set (int * array, int bit)`

10.20.2.13 `void c_ekk_Unset (int * array, int bit)`

10.20.2.14 `void c_ekkdzero (int length, int n, void * array)`

10.20.2.15 `void c_ekkdzero (int n, double * marray)` `[inline]`

Definition at line 70 of file CoinOslC.h.

10.20.2.16 `void c_ekkzero (int n, int * marray)` `[inline]`

Definition at line 72 of file CoinOslC.h.

10.20.2.17 `void c_ekkczero (int n, char * marray)` `[inline]`

Definition at line 74 of file CoinOslC.h.

10.20.2.18 `void clp_setup_pointers (EKKfactinfo * fact)`

10.20.2.19 `void clp_memory (int type)`

10.20.2.20 `double* clp_double (int number_entries)`

10.20.2.21 `int* clp_int (int number_entries)`

10.20.2.22 `void* clp_malloc (int number_entries)`

10.20.2.23 `void clp_free (void * oldArray)`

10.21 /home/ted/COIN/trunk/CoinUtils/src/CoinOslFactorization.hpp File Reference

```
#include <iostream>
#include <string>
#include <cassert>
#include "CoinTypes.hpp"
#include "CoinIndexedVector.hpp"
#include "CoinDenseFactorization.hpp"
```

Classes

- struct [EKKHlink](#)

This deals with Factorization and Updates This is ripped off from OSL!!!!!!!!!!

- struct [_EKKfactinfo](#)
- class [CoinOslFactorization](#)

Typedefs

- typedef struct [_EKKfactinfo](#) [EKKfactinfo](#)

10.21.1 Typedef Documentation

10.21.1.1 typedef struct [_EKKfactinfo](#) [EKKfactinfo](#)

10.22 /home/ted/COIN/trunk/CoinUtils/src/CoinPackedMatrix.hpp File Reference

```
#include "CoinError.hpp"
#include "CoinTypes.hpp"
#include "CoinPackedVectorBase.hpp"
#include "CoinShallowPackedVector.hpp"
```

Classes

- class [CoinPackedMatrix](#)
Sparse Matrix Base Class.

Functions

- void [CoinPackedMatrixUnitTest](#) ()
Test the methods in the [CoinPackedMatrix](#) class.

10.22.1 Function Documentation

10.22.1.1 void [CoinPackedMatrixUnitTest](#) ()

Test the methods in the [CoinPackedMatrix](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

10.23 /home/ted/COIN/trunk/CoinUtils/src/CoinPackedVector.hpp File Reference

```
#include <map>
#include "CoinPragma.hpp"
#include "CoinPackedVectorBase.hpp"
#include "CoinSort.hpp"
```

Classes

- class [CoinPackedVector](#)
Sparse Vector.

Macros

- `#define COIN_DEFAULT_VALUE_FOR_DUPLICATE true`

Functions

- `double sparseDotProduct (const CoinPackedVectorBase &op1, const CoinPackedVectorBase &op2)`
Returns the dot product of two *CoinPackedVector* objects whose elements are doubles.
- `double sortedSparseDotProduct (const CoinPackedVectorBase &op1, const CoinPackedVectorBase &op2)`
Returns the dot product of two sorted *CoinPackedVector* objects.
- `void CoinPackedVectorUnitTest ()`
A function that tests the methods in the *CoinPackedVector* class.

Arithmetic operators on packed vectors.

NOTE: These methods operate on those positions where at least one of the arguments has a value listed.

At those positions the appropriate operation is executed, Otherwise the result of the operation is considered 0.

NOTE 2: There are two kind of operators here. One is used like "*c* = *binaryOp*(*a*, *b*)", the other is used like "*binaryOp*(*c*, *a*, *b*)", but they are really the same. The first is much more natural to use, but it involves the creation of a temporary object (the function must return an object), while the second form puts the result directly into the argument "*c*". Therefore, depending on the circumstances, the second form can be significantly faster.

- `template<class BinaryFunction >`
`void binaryOp (CoinPackedVector &retVal, const CoinPackedVectorBase &op1, double value, BinaryFunction bf)`
- `template<class BinaryFunction >`
`void binaryOp (CoinPackedVector &retVal, double value, const CoinPackedVectorBase &op2, BinaryFunction bf)`
- `template<class BinaryFunction >`
`void binaryOp (CoinPackedVector &retVal, const CoinPackedVectorBase &op1, const CoinPackedVectorBase &op2, BinaryFunction bf)`
- `template<class BinaryFunction >`
`CoinPackedVector binaryOp (const CoinPackedVectorBase &op1, double value, BinaryFunction bf)`
- `template<class BinaryFunction >`
`CoinPackedVector binaryOp (double value, const CoinPackedVectorBase &op2, BinaryFunction bf)`
- `template<class BinaryFunction >`
`CoinPackedVector binaryOp (const CoinPackedVectorBase &op1, const CoinPackedVectorBase &op2, BinaryFunction bf)`
- `CoinPackedVector operator+ (const CoinPackedVectorBase &op1, const CoinPackedVectorBase &op2)`
Return the sum of two packed vectors.
- `CoinPackedVector operator- (const CoinPackedVectorBase &op1, const CoinPackedVectorBase &op2)`
Return the difference of two packed vectors.
- `CoinPackedVector operator* (const CoinPackedVectorBase &op1, const CoinPackedVectorBase &op2)`
Return the element-wise product of two packed vectors.
- `CoinPackedVector operator/ (const CoinPackedVectorBase &op1, const CoinPackedVectorBase &op2)`
Return the element-wise ratio of two packed vectors.

Arithmetic operators on packed vector and a constant.

These functions create a packed vector as a result.

That packed vector will have the same indices as *op1* and the specified operation is done entry-wise with the given value.

- `CoinPackedVector operator+ (const CoinPackedVectorBase &op1, double value)`

- Return the sum of a packed vector and a constant.*
- [CoinPackedVector operator-](#) (const [CoinPackedVectorBase](#) &op1, double value)
Return the difference of a packed vector and a constant.
- [CoinPackedVector operator*](#) (const [CoinPackedVectorBase](#) &op1, double value)
Return the element-wise product of a packed vector and a constant.
- [CoinPackedVector operator/](#) (const [CoinPackedVectorBase](#) &op1, double value)
Return the element-wise ratio of a packed vector and a constant.
- [CoinPackedVector operator+](#) (double value, const [CoinPackedVectorBase](#) &op1)
Return the sum of a constant and a packed vector.
- [CoinPackedVector operator-](#) (double value, const [CoinPackedVectorBase](#) &op1)
Return the difference of a constant and a packed vector.
- [CoinPackedVector operator*](#) (double value, const [CoinPackedVectorBase](#) &op1)
Return the element-wise product of a constant and a packed vector.
- [CoinPackedVector operator/](#) (double value, const [CoinPackedVectorBase](#) &op1)
Return the element-wise ratio of a constant and a packed vector.

10.23.1 Macro Definition Documentation

10.23.1.1 `#define COIN_DEFAULT_VALUE_FOR_DUPLICATE true`

Definition at line 22 of file `CoinPackedVector.hpp`.

10.23.2 Function Documentation

10.23.2.1 `template<class BinaryFunction> void binaryOp (CoinPackedVector & retVal, const CoinPackedVectorBase & op1, double value, BinaryFunction bf)`

Definition at line 360 of file `CoinPackedVector.hpp`.

10.23.2.2 `template<class BinaryFunction> void binaryOp (CoinPackedVector & retVal, double value, const CoinPackedVectorBase & op2, BinaryFunction bf) [inline]`

Definition at line 377 of file `CoinPackedVector.hpp`.

10.23.2.3 `template<class BinaryFunction> void binaryOp (CoinPackedVector & retVal, const CoinPackedVectorBase & op1, const CoinPackedVectorBase & op2, BinaryFunction bf)`

Definition at line 385 of file `CoinPackedVector.hpp`.

10.23.2.4 `template<class BinaryFunction> CoinPackedVector binaryOp (const CoinPackedVectorBase & op1, double value, BinaryFunction bf)`

Definition at line 433 of file `CoinPackedVector.hpp`.

10.23.2.5 `template<class BinaryFunction> CoinPackedVector binaryOp (double value, const CoinPackedVectorBase & op2, BinaryFunction bf)`

Definition at line 443 of file `CoinPackedVector.hpp`.

10.23.2.6 `template<class BinaryFunction> CoinPackedVector binaryOp (const CoinPackedVectorBase & op1, const CoinPackedVectorBase & op2, BinaryFunction bf)`

Definition at line 453 of file `CoinPackedVector.hpp`.

10.23.2.7 `CoinPackedVector operator+ (const CoinPackedVectorBase & op1, const CoinPackedVectorBase & op2)`
[inline]

Return the sum of two packed vectors.

Definition at line 464 of file CoinPackedVector.hpp.

10.23.2.8 `CoinPackedVector operator- (const CoinPackedVectorBase & op1, const CoinPackedVectorBase & op2)`
[inline]

Return the difference of two packed vectors.

Definition at line 474 of file CoinPackedVector.hpp.

10.23.2.9 `CoinPackedVector operator* (const CoinPackedVectorBase & op1, const CoinPackedVectorBase & op2)`
[inline]

Return the element-wise product of two packed vectors.

Definition at line 484 of file CoinPackedVector.hpp.

10.23.2.10 `CoinPackedVector operator/ (const CoinPackedVectorBase & op1, const CoinPackedVectorBase & op2)`
[inline]

Return the element-wise ratio of two packed vectors.

Definition at line 494 of file CoinPackedVector.hpp.

10.23.2.11 `double sparseDotProduct (const CoinPackedVectorBase & op1, const CoinPackedVectorBase & op2)`
[inline]

Returns the dot product of two [CoinPackedVector](#) objects whose elements are doubles.

Use this version if the vectors are *not* guaranteed to be sorted.

Definition at line 506 of file CoinPackedVector.hpp.

10.23.2.12 `double sortedSparseDotProduct (const CoinPackedVectorBase & op1, const CoinPackedVectorBase & op2)`
[inline]

Returns the dot product of two sorted [CoinPackedVector](#) objects.

The vectors should be sorted in ascending order of indices.

Definition at line 525 of file CoinPackedVector.hpp.

10.23.2.13 `CoinPackedVector operator+ (const CoinPackedVectorBase & op1, double value)` [inline]

Return the sum of a packed vector and a constant.

Definition at line 567 of file CoinPackedVector.hpp.

10.23.2.14 `CoinPackedVector operator- (const CoinPackedVectorBase & op1, double value)` [inline]

Return the difference of a packed vector and a constant.

Definition at line 576 of file CoinPackedVector.hpp.

10.23.2.15 `CoinPackedVector operator* (const CoinPackedVectorBase & op1, double value)` [inline]

Return the element-wise product of a packed vector and a constant.

Definition at line 585 of file CoinPackedVector.hpp.

10.23.2.16 `CoinPackedVector operator/ (const CoinPackedVectorBase & op1, double value) [inline]`

Return the element-wise ratio of a packed vector and a constant.

Definition at line 594 of file CoinPackedVector.hpp.

10.23.2.17 `CoinPackedVector operator+ (double value, const CoinPackedVectorBase & op1) [inline]`

Return the sum of a constant and a packed vector.

Definition at line 605 of file CoinPackedVector.hpp.

10.23.2.18 `CoinPackedVector operator- (double value, const CoinPackedVectorBase & op1) [inline]`

Return the difference of a constant and a packed vector.

Definition at line 614 of file CoinPackedVector.hpp.

10.23.2.19 `CoinPackedVector operator* (double value, const CoinPackedVectorBase & op1) [inline]`

Return the element-wise product of a constant and a packed vector.

Definition at line 627 of file CoinPackedVector.hpp.

10.23.2.20 `CoinPackedVector operator/ (double value, const CoinPackedVectorBase & op1) [inline]`

Return the element-wise ratio of a a constant and packed vector.

Definition at line 636 of file CoinPackedVector.hpp.

10.23.2.21 `void CoinPackedVectorUnitTest ()`

A function that tests the methods in the [CoinPackedVector](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

10.24 /home/ted/COIN/trunk/CoinUtils/src/CoinPackedVectorBase.hpp File Reference

```
#include <set>
#include <map>
#include "CoinPragma.hpp"
#include "CoinError.hpp"
```

Classes

- class [CoinPackedVectorBase](#)

Abstract base class for various sparse vectors.

10.25 /home/ted/COIN/trunk/CoinUtils/src/CoinParam.hpp File Reference

Declaration of a class for command line parameters.


```
#include <vector>
#include <string>
#include <cstdio>
```

Classes

- class [CoinParam](#)
A base class for 'keyword value' command line parameters.

Namespaces

- [CoinParamUtils](#)
Utility functions for processing [CoinParam](#) parameters.

Functions

- `std::ostream & operator<< (std::ostream &s, const CoinParam ¶m)`
A stream output function for a [CoinParam](#) object.
- `void CoinParamUtils::setInputSrc (FILE *src)`
Take command input from the file specified by src.
- `bool CoinParamUtils::isCommandLine ()`
Returns true if command line parameters are being processed.
- `bool CoinParamUtils::isInteractive ()`
Returns true if parameters are being obtained from stdin.
- `std::string CoinParamUtils::getStringField (int argc, const char *argv[], int *valid)`
Attempt to read a string from the input.
- `int CoinParamUtils::getIntField (int argc, const char *argv[], int *valid)`
Attempt to read an integer from the input.
- `double CoinParamUtils::getDoubleField (int argc, const char *argv[], int *valid)`
Attempt to read a real (double) from the input.
- `int CoinParamUtils::matchParam (const CoinParamVec ¶mVec, std::string name, int &matchNdx, int &short-Cnt)`
*Scan a parameter vector for parameters whose keyword (name) string matches *name* using minimal match rules.*
- `std::string CoinParamUtils::getCommand (int argc, const char *argv[], const std::string prompt, std::string *pfx=0)`
Get the next command keyword (name)
- `int CoinParamUtils::lookupParam (std::string name, CoinParamVec ¶mVec, int *matchCnt=0, int *short-Cnt=0, int *queryCnt=0)`
Look up the command keyword (name) in the parameter vector. Print help if requested.
- `void CoinParamUtils::printIt (const char *msg)`
Utility to print a long message as filled lines of text.
- `void CoinParamUtils::shortOrHelpOne (CoinParamVec ¶mVec, int matchNdx, std::string name, int num-Query)`
Utility routine to print help given a short match or explicit request for help.
- `void CoinParamUtils::shortOrHelpMany (CoinParamVec ¶mVec, std::string name, int numQuery)`
Utility routine to print help given multiple matches.
- `void CoinParamUtils::printGenericHelp ()`

Print a generic 'how to use the command interface' help message.

- `void CoinParamUtils::printHelp (CoinParamVec ¶mVec, int firstParam, int lastParam, std::string prefix, bool shortHelp, bool longHelp, bool hidden)`

Utility routine to print help messages for one or more parameters.

10.25.1 Detailed Description

Declaration of a class for command line parameters.

Definition in file [CoinParam.hpp](#).

10.25.2 Function Documentation

10.25.2.1 `std::ostream & operator<< (std::ostream & s, const CoinParam & param)`

A stream output function for a [CoinParam](#) object.

10.26 [/home/ted/COIN/trunk/CoinUtils/src/CoinPragma.hpp](#) File Reference

10.27 [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDoubleton.hpp](#) File Reference

Classes

- class [doubleton_action](#)
Solve $ax+by=c$ for y and substitute y out of the problem.
- struct [doubleton_action::action](#)

Macros

- `#define DOUBLETON 5`

10.27.1 Macro Definition Documentation

10.27.1.1 `#define DOUBLETON 5`

Definition at line 9 of file [CoinPresolveDoubleton.hpp](#).

10.28 [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDual.hpp](#) File Reference

Classes

- class [remove_dual_action](#)
Attempt to fix variables by bounding reduced costs.

10.29 [/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDupcol.hpp](#) File Reference

```
#include "CoinPresolveMatrix.hpp"
```

Classes

- class [dupcol_action](#)
Detect and remove duplicate columns.
- class [duprow_action](#)
Detect and remove duplicate rows.
- class [gubrow_action](#)
Detect and remove entries whose sum is known.
- class [twotwo_action](#)
Detect interesting 2 by 2 blocks.

Macros

- `#define DUPCOL 10`

10.29.1 Macro Definition Documentation

10.29.1.1 `#define DUPCOL 10`

Definition at line 15 of file `CoinPresolveDupcol.hpp`.

10.30 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveEmpty.hpp File Reference

Drop/reinsert empty rows/columns.

Classes

- class [drop_empty_cols_action](#)
Physically removes empty columns in presolve, and reinserts empty columns in postsolve.
- class [drop_empty_rows_action](#)
Physically removes empty rows in presolve, and reinserts empty rows in postsolve.

Variables

- `const int DROP_ROW = 3`
- `const int DROP_COL = 4`

10.30.1 Detailed Description

Drop/reinsert empty rows/columns.

Definition in file [CoinPresolveEmpty.hpp](#).

10.30.2 Variable Documentation

10.30.2.1 `const int DROP_ROW = 3`

Definition at line 14 of file `CoinPresolveEmpty.hpp`.

10.30.2.2 `const int DROP_COL = 4`

Definition at line 15 of file `CoinPresolveEmpty.hpp`.

10.31 `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveFixed.hpp` File Reference

Classes

- class [remove_fixed_action](#)
Excise fixed variables from the model.
- struct [remove_fixed_action::action](#)
Structure to hold information necessary to reintroduce a column into the problem representation.
- class [make_fixed_action](#)
Fix a variable at a specified bound.

Macros

- `#define FIXED_VARIABLE 1`

10.31.1 Macro Definition Documentation

10.31.1.1 `#define FIXED_VARIABLE 1`

Definition at line 8 of file `CoinPresolveFixed.hpp`.

10.32 `/home/ted/COIN/trunk/CoinUtils/src/CoinPresolveForcing.hpp` File Reference

```
#include "CoinPresolveMatrix.hpp"
```

Classes

- class [forcing_constraint_action](#)
Detect and process forcing constraints and useless constraints.
- struct [forcing_constraint_action::action](#)

Macros

- `#define IMPLIED_BOUND 7`

10.32.1 Macro Definition Documentation

10.32.1.1 `#define IMPLIED_BOUND 7`

Definition at line 15 of file `CoinPresolveForcing.hpp`.

10.33 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveImpliedFree.hpp File Reference

Classes

- class [implied_free_action](#)
Detect and process implied free variables.

Macros

- #define [CoinPresolveImpliedFree_H](#)
- #define [IMPLIED_FREE](#) 9

10.33.1 Macro Definition Documentation

10.33.1.1 #define [CoinPresolveImpliedFree_H](#)

Definition at line 7 of file [CoinPresolveImpliedFree.hpp](#).

10.33.1.2 #define [IMPLIED_FREE](#) 9

Definition at line 13 of file [CoinPresolveImpliedFree.hpp](#).

10.34 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveIsolated.hpp File Reference

```
#include "CoinPresolveMatrix.hpp"
```

Classes

- class [isolated_constraint_action](#)

10.35 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMatrix.hpp File Reference

Declarations for [CoinPresolveMatrix](#) and [CoinPostsolveMatrix](#) and their common base class [CoinPrePostsolveMatrix](#).

```
#include "CoinPragma.hpp"  
#include "CoinPackedMatrix.hpp"  
#include "CoinMessage.hpp"  
#include "CoinTime.hpp"  
#include <cmath>  
#include <cassert>  
#include <cfloat>  
#include <cstdlib>
```

Classes

- class [CoinPresolveAction](#)
Abstract base class of all presolve routines.
- class [CoinPrePostsolveMatrix](#)

Collects all the information about the problem that is needed in both presolve and postsolve.

- class [presolvehlink](#)

Links to aid in packed matrix modification.

- class [CoinPresolveMatrix](#)

Augments [CoinPrePostsolveMatrix](#) with information about the problem that is only needed during presolve.

- class [CoinPostsolveMatrix](#)

Augments [CoinPrePostsolveMatrix](#) with information about the problem that is only needed during postsolve.

Macros

- #define [deleteAction](#)(array, type) delete [] array

- #define [PRESOLVEASSERT](#)(x) {}

- #define [PRESOLVE_STMT](#)(s) {}

- #define [PRESOLVE_DETAIL_PRINT](#)(s) {}

- #define [PRESOLVE_INF](#) [COIN_DBL_MAX](#)

The usual finite infinity.

- #define [PRESOLVE_SMALL_INF](#) 1.0e20

And a small infinity.

- #define [PRESOLVEFINITE](#)(n) ([-PRESOLVE_INF](#) < (n) && (n) < [PRESOLVE_INF](#))

Check for infinity using finite infinity.

- #define [NO_LINK](#) -66666666

Functions

- void [DIE](#) (const char *)

- double * [presolve_dupmajor](#) (const double *elems, const int *indices, int length, [CoinBigIndex](#) offset, int tgt=-1)

*Duplicate a major-dimension vector; optionally omit the entry with minor index *tgt*.*

- void [coin_init_random_vec](#) (double *work, int n)

Initialize a vector with random numbers.

Variables

- const double [ZTOLDP](#) = 1e-12

Zero tolerance.

- const double [ZTOLDP2](#) = 1e-10

Alternate zero tolerance.

10.35.1 Detailed Description

Declarations for [CoinPresolveMatrix](#) and [CoinPostsolveMatrix](#) and their common base class [CoinPrePostsolveMatrix](#). Also declarations for [CoinPresolveAction](#) and a number of non-member utility functions.

Definition in file [CoinPresolveMatrix.hpp](#).

10.35.2 Macro Definition Documentation

10.35.2.1 #define deleteAction(array, type) delete [] array

Definition at line 38 of file [CoinPresolveMatrix.hpp](#).

10.35.2.2 `#define PRESOLVEASSERT(x) {}`

Definition at line 69 of file CoinPresolveMatrix.hpp.

10.35.2.3 `#define PRESOLVE_STMT(s) {}`

Definition at line 70 of file CoinPresolveMatrix.hpp.

10.35.2.4 `#define PRESOLVE_DETAIL_PRINT(s) {}`

Definition at line 80 of file CoinPresolveMatrix.hpp.

10.35.2.5 `#define PRESOLVE_INF COIN_DBL_MAX`

The usual finite infinity.

Definition at line 97 of file CoinPresolveMatrix.hpp.

10.35.2.6 `#define PRESOLVE_SMALL_INF 1.0e20`

And a small infinity.

Definition at line 99 of file CoinPresolveMatrix.hpp.

10.35.2.7 `#define PRESOLVEFINITE(n) (-PRESOLVE_INF < (n) && (n) < PRESOLVE_INF)`

Check for infinity using finite infinity.

Definition at line 101 of file CoinPresolveMatrix.hpp.

10.35.2.8 `#define NO_LINK -66666666`

Definition at line 743 of file CoinPresolveMatrix.hpp.

10.35.3 Function Documentation

10.35.3.1 `void DIE (const char *) [inline]`

Definition at line 72 of file CoinPresolveMatrix.hpp.

10.35.4 Variable Documentation

10.35.4.1 `const double ZTOLDP = 1e-12`

Zero tolerance.

OSL had a fixed zero tolerance; we still use that here.

Definition at line 89 of file CoinPresolveMatrix.hpp.

10.35.4.2 `const double ZTOLDP2 = 1e-10`

Alternate zero tolerance.

Use a different one if we are doing doubletons, etc.

Definition at line 94 of file CoinPresolveMatrix.hpp.

10.36 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMonitor.hpp File Reference

Classes

- class [CoinPresolveMonitor](#)
Monitor a row or column for modification.

10.37 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolvePsdebug.hpp File Reference

10.38 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSingleton.hpp File Reference

Classes

- class [slack_doubleton_action](#)
Convert an explicit bound constraint to a column bound.
- class [slack_singleton_action](#)
For variables with one entry.

Macros

- #define [SLACK_DOUBLETION](#) 2
- #define [SLACK_SINGLETON](#) 8

10.38.1 Macro Definition Documentation

10.38.1.1 #define SLACK_DOUBLETION 2

Definition at line 8 of file CoinPresolveSingleton.hpp.

10.38.1.2 #define SLACK_SINGLETON 8

Definition at line 9 of file CoinPresolveSingleton.hpp.

10.39 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSubst.hpp File Reference

```
#include "CoinPresolveMatrix.hpp"
```

Classes

- class [subst_constraint_action](#)
Detect and process implied free variables.

Macros

- #define [SUBST_ROW](#) 21

Functions

- [void implied_bounds](#) (const double *els, const double *clo, const double *cup, const int *hcol, [CoinBigIndex](#) krs, [CoinBigIndex](#) kre, double *maxupp, double *maxdownp, int jcol, double rlo, double rup, double *iclb, double *icub)

10.39.1 Macro Definition Documentation

10.39.1.1 #define SUBST_ROW 21

Definition at line 13 of file CoinPresolveSubst.hpp.

10.39.2 Function Documentation

- 10.39.2.1 **void implied_bounds** (const double * *els*, const double * *clo*, const double * *cup*, const int * *hcol*, [CoinBigIndex](#) *krs*, [CoinBigIndex](#) *kre*, double * *maxupp*, double * *maxdownp*, int *jcol*, double *rlo*, double *rup*, double * *iclb*, double * *icub*)

10.40 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTighten.hpp File Reference

```
#include "CoinPresolveMatrix.hpp"
```

Classes

- class [do_tighten_action](#)

Macros

- #define [DO_TIGHTEN](#) 30

Functions

- const [CoinPresolveAction](#) * [tighten_zero_cost](#) ([CoinPresolveMatrix](#) *prob, const [CoinPresolveAction](#) *next)

10.40.1 Macro Definition Documentation

10.40.1.1 #define DO_TIGHTEN 30

Definition at line 17 of file CoinPresolveTighten.hpp.

10.40.2 Function Documentation

- 10.40.2.1 **const CoinPresolveAction*** [tighten_zero_cost](#) ([CoinPresolveMatrix](#) * *prob*, const [CoinPresolveAction](#) * *next*)

10.41 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTripletion.hpp File Reference

Classes

- class [tripleton_action](#)
We are only going to do this if it does not increase number of elements?.
- struct [tripleton_action::action](#)

Macros

- #define [TRIPLETON](#) 11

10.41.1 Macro Definition Documentation

10.41.1.1 #define TRIPLETON 11

Definition at line 8 of file CoinPresolveTripleton.hpp.

10.42 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveUseless.hpp File Reference

Classes

- class [useless_constraint_action](#)

Macros

- #define [USELESS](#) 20

10.42.1 Macro Definition Documentation

10.42.1.1 #define USELESS 20

Definition at line 8 of file CoinPresolveUseless.hpp.

10.43 /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveZeros.hpp File Reference

Drop/reintroduce explicit zeros.

Classes

- struct [dropped_zero](#)
Tracking information for an explicit zero coefficient.
- class [drop_zero_coefficients_action](#)
Removal of explicit zeros.

Macros

- #define [DROP_ZERO](#) 8

Functions

- const [CoinPresolveAction](#) * [drop_zero_coefficients](#) ([CoinPresolveMatrix](#) *prob, const [CoinPresolveAction](#) *next)

10.43.1 Detailed Description

Drop/reintroduce explicit zeros.

Definition in file [CoinPresolveZeros.hpp](#).

10.43.2 Macro Definition Documentation

10.43.2.1 #define DROP_ZERO 8

Definition at line 14 of file [CoinPresolveZeros.hpp](#).

10.43.3 Function Documentation

- 10.43.3.1 **const [CoinPresolveAction](#)* [drop_zero_coefficients](#) ([CoinPresolveMatrix](#) * *prob*, const [CoinPresolveAction](#) * *next*)**

10.44 /home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp File Reference

```
#include <vector>
#include <algorithm>
#include <cmath>
#include <string>
#include "CoinFinite.hpp"
#include "CoinHelperFunctions.hpp"
```

Classes

- class [BitVector128](#)
- class [CoinTreeNode](#)
 - A class from which the real tree nodes should be derived from.*
- class [CoinTreeSiblings](#)
- struct [CoinSearchTreeComparePreferred](#)
 - Function objects to compare search tree nodes.*
- struct [CoinSearchTreeCompareDepth](#)
 - Depth First Search.*
- struct [CoinSearchTreeCompareBreadth](#)
- struct [CoinSearchTreeCompareBest](#)
 - Best first search.*
- class [CoinSearchTreeBase](#)
- class [CoinSearchTree< Comp >](#)
- class [CoinSearchTreeManager](#)

Enumerations

- enum [CoinNodeAction](#) { [CoinAddNodeToCandidates](#), [CoinTestNodeForDiving](#), [CoinDiveIntoNode](#) }

Functions

- bool [operator<](#) (const [BitVector128](#) &b0, const [BitVector128](#) &b1)

10.44.1 Enumeration Type Documentation

10.44.1.1 enum [CoinNodeAction](#)

Enumerator

[CoinAddNodeToCandidates](#)***[CoinTestNodeForDiving](#)******[CoinDiveIntoNode](#)***

Definition at line 398 of file [CoinSearchTree.hpp](#).

10.44.2 Function Documentation

10.44.2.1 bool [operator<](#) (const [BitVector128](#) & *b0*, const [BitVector128](#) & *b1*)10.45 [/home/ted/COIN/trunk/CoinUtils/src/CoinShallowPackedVector.hpp](#) File Reference

```
#include "CoinError.hpp"
#include "CoinPackedVectorBase.hpp"
```

Classes

- class [CoinShallowPackedVector](#)
Shallow Sparse Vector.

Functions

- void [CoinShallowPackedVectorUnitTest](#) ()
A function that tests the methods in the [CoinShallowPackedVector](#) class.

10.45.1 Function Documentation

10.45.1.1 void [CoinShallowPackedVectorUnitTest](#) ()

A function that tests the methods in the [CoinShallowPackedVector](#) class.

The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

10.46 /home/ted/COIN/trunk/CoinUtils/src/CoinSignal.hpp File Reference

```
#include <csignal>
```

Typedefs

- typedef [void\(* CoinSigHandler_t\)](#)(int)

10.46.1 Typedef Documentation

10.46.1.1 typedef void(* CoinSigHandler_t)(int)

Definition at line 111 of file CoinSignal.hpp.

10.47 /home/ted/COIN/trunk/CoinUtils/src/CoinSimpFactorization.hpp File Reference

```
#include <iostream>
#include <string>
#include <cassert>
#include "CoinTypes.hpp"
#include "CoinIndexedVector.hpp"
#include "CoinDenseFactorization.hpp"
```

Classes

- class [FactorPointers](#)
pointers used during factorization
- class [CoinSimpFactorization](#)

10.48 /home/ted/COIN/trunk/CoinUtils/src/CoinSmartPtr.hpp File Reference

```
#include <list>
#include <cassert>
#include <cstddef>
#include <cstring>
```

Classes

- class [Coin::ReferencedObject](#)
ReferencedObject class.
- class [Coin::SmartPtr< T >](#)
Template class for Smart Pointers.

Namespaces

- [Coin](#)

Macros

- `#define dbg_smartptr_verbosity 0`
- `#define CoinReferencedObject Coin::ReferencedObject`
- `#define CoinSmartPtr Coin::SmartPtr`
- `#define CoinComparePointers Coin::ComparePointers`

Functions

- `template<class U1 , class U2 >`
`bool Coin::ComparePointers (const U1 *lhs, const U2 *rhs)`

SmartPtr friends that are overloaded operators, so they are not in the *Coin* namespace.

- `template<class U1 , class U2 >`
`bool operator== (const Coin::SmartPtr< U1 > &lhs, const Coin::SmartPtr< U2 > &rhs)`
- `template<class U1 , class U2 >`
`bool operator== (const Coin::SmartPtr< U1 > &lhs, U2 *raw_rhs)`
- `template<class U1 , class U2 >`
`bool operator== (U1 *raw_lhs, const Coin::SmartPtr< U2 > &rhs)`
- `template<class U1 , class U2 >`
`bool operator!= (const Coin::SmartPtr< U1 > &lhs, const Coin::SmartPtr< U2 > &rhs)`
- `template<class U1 , class U2 >`
`bool operator!= (const Coin::SmartPtr< U1 > &lhs, U2 *raw_rhs)`
- `template<class U1 , class U2 >`
`bool operator!= (U1 *raw_lhs, const Coin::SmartPtr< U2 > &rhs)`

10.48.1 Macro Definition Documentation

10.48.1.1 `#define dbg_smartptr_verbosity 0`

Definition at line 379 of file CoinSmartPtr.hpp.

10.48.1.2 `#define CoinReferencedObject Coin::ReferencedObject`

Definition at line 524 of file CoinSmartPtr.hpp.

10.48.1.3 `#define CoinSmartPtr Coin::SmartPtr`

Definition at line 525 of file CoinSmartPtr.hpp.

10.48.1.4 `#define CoinComparePointers Coin::ComparePointers`

Definition at line 526 of file CoinSmartPtr.hpp.

10.48.2 Function Documentation

10.48.2.1 `template<class U1 , class U2 > bool operator== (const Coin::SmartPtr< U1 > & lhs, const Coin::SmartPtr< U2 > & rhs)`

Definition at line 494 of file CoinSmartPtr.hpp.

10.48.2.2 `template<class U1 , class U2 > bool operator== (const Coin::SmartPtr< U1 > & lhs, U2 * raw_rhs)`

Definition at line 499 of file CoinSmartPtr.hpp.

10.48.2.3 `template<class U1 , class U2 > bool operator== (U1 * raw_lhs, const Coin::SmartPtr< U2 > & rhs)`

Definition at line 504 of file CoinSmartPtr.hpp.

10.48.2.4 `template<class U1 , class U2 > bool operator!= (const Coin::SmartPtr< U1 > & lhs, const Coin::SmartPtr< U2 > & rhs)`

Definition at line 509 of file CoinSmartPtr.hpp.

10.48.2.5 `template<class U1 , class U2 > bool operator!= (const Coin::SmartPtr< U1 > & lhs, U2 * raw_rhs)`

Definition at line 514 of file CoinSmartPtr.hpp.

10.48.2.6 `template<class U1 , class U2 > bool operator!= (U1 * raw_lhs, const Coin::SmartPtr< U2 > & rhs)`

Definition at line 519 of file CoinSmartPtr.hpp.

10.49 /home/ted/COIN/trunk/CoinUtils/src/CoinSnapshot.hpp File Reference

```
#include "CoinTypes.hpp"
```

Classes

- class [CoinSnapshot](#)
NON Abstract Base Class for interfacing with cut generators or branching code or .

10.50 /home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp File Reference

```
#include <functional>
#include <new>
#include <algorithm>
#include "CoinDistance.hpp"
```

Classes

- struct [CoinPair< S, T >](#)
An ordered pair.
- class [CoinFirstLess_2< S, T >](#)
Function operator.
- class [CoinFirstGreater_2< S, T >](#)
Function operator.
- class [CoinFirstAbsLess_2< S, T >](#)
Function operator.
- class [CoinFirstAbsGreater_2< S, T >](#)

- Function operator.*
- class [CoinExternalVectorFirstLess_2< S, T, V >](#)
- Function operator.*
- class [CoinExternalVectorFirstGreater_2< S, T, V >](#)
- Function operator.*
- class [CoinTriple< S, T, U >](#)
- class [CoinFirstLess_3< S, T, U >](#)
- Function operator.*
- class [CoinFirstGreater_3< S, T, U >](#)
- Function operator.*
- class [CoinFirstAbsLess_3< S, T, U >](#)
- Function operator.*
- class [CoinFirstAbsGreater_3< S, T, U >](#)
- Function operator.*
- class [CoinExternalVectorFirstLess_3< S, T, U, V >](#)
- Function operator.*
- class [CoinExternalVectorFirstGreater_3< S, T, U, V >](#)
- Function operator.*

Typedefs

Typedefs for sorting the entries of a packed vector based on an external vector.

- typedef
[CoinExternalVectorFirstLess_3](#)
 < int, int, double, double > [CoinIncrSolutionOrdered](#)
Sort packed vector in increasing order of the external vector.
- typedef
[CoinExternalVectorFirstGreater_3](#)
 < int, int, double, double > [CoinDecrSolutionOrdered](#)
Sort packed vector in decreasing order of the external vector.

Functions

- template<class S , class T , class CoinCompare2 >
[void CoinSort_2](#) (S *sfirst, S *slast, T *tfirst, const CoinCompare2 &pc)
Sort a pair of containers.
- template<class S , class T >
[void CoinSort_2Std](#) (S *sfirst, S *slast, T *tfirst)
- template<class S , class T >
[void CoinSort_2](#) (S *sfirst, S *slast, T *tfirst)
- template<class S , class T >
[void CoinShortSort_2](#) (S *key, S *lastKey, T *array2)
Sort without new and delete.
- template<class S , class T , class U , class CoinCompare3 >
[void CoinSort_3](#) (S *sfirst, S *slast, T *tfirst, U *ufirst, const CoinCompare3 &tc)
Sort a triple of containers.
- template<class S , class T , class U >
[void CoinSort_3](#) (S *sfirst, S *slast, T *tfirst, U *ufirst)

10.50.1 Typedef Documentation

10.50.1.1 `typedef CoinExternalVectorFirstLess_3<int, int, double, double> CoinIncrSolutionOrdered`

Sort packed vector in increasing order of the external vector.

Definition at line 571 of file CoinSort.hpp.

10.50.1.2 `typedef CoinExternalVectorFirstGreater_3<int, int, double, double> CoinDecrSolutionOrdered`

Sort packed vector in decreasing order of the external vector.

Definition at line 574 of file CoinSort.hpp.

10.50.2 Function Documentation

10.50.2.1 `template<class S, class T, class CoinCompare2 > void CoinSort_2 (S * sfirst, S * slast, T * tfirst, const CoinCompare2 & pc)`

Sort a pair of containers.

Iter_S - iterator for first container

Iter_T - iterator for 2nd container

CoinCompare2 - class comparing CoinPairs

Definition at line 188 of file CoinSort.hpp.

10.50.2.2 `template<class S, class T > void CoinSort_2Std (S * sfirst, S * slast, T * tfirst)`

Definition at line 222 of file CoinSort.hpp.

10.50.2.3 `template<class S, class T > void CoinSort_2 (S * sfirst, S * slast, T * tfirst)`

Definition at line 229 of file CoinSort.hpp.

10.50.2.4 `template<class S, class T > void CoinShortSort_2 (S * key, S * lastKey, T * array2)`

Sort without new and delete.

Definition at line 357 of file CoinSort.hpp.

10.50.2.5 `template<class S, class T, class U, class CoinCompare3 > void CoinSort_3 (S * sfirst, S * slast, T * tfirst, U * ufirst, const CoinCompare3 & tc)`

Sort a triple of containers.

Iter_S - iterator for first container

Iter_T - iterator for 2nd container

Iter_U - iterator for 3rd container

CoinCompare3 - class comparing CoinTriples

Definition at line 636 of file CoinSort.hpp.

10.50.2.6 `template<class S, class T, class U > void CoinSort_3 (S * sfirst, S * slast, T * tfirst, U * ufirst)`

Definition at line 669 of file CoinSort.hpp.

10.51 /home/ted/COIN/trunk/CoinUtils/src/CoinStructuredModel.hpp File Reference

```
#include "CoinModel.hpp"  
#include <vector>
```

Classes

- struct [CoinModelInfo2](#)
This is a model which is made up of Coin(Structured)Model blocks.
- class [CoinStructuredModel](#)

Typedefs

- typedef struct [CoinModelInfo2](#) [CoinModelBlockInfo](#)
This is a model which is made up of Coin(Structured)Model blocks.

10.51.1 Typedef Documentation

10.51.1.1 typedef struct [CoinModelInfo2](#) [CoinModelBlockInfo](#)

This is a model which is made up of Coin(Structured)Model blocks.

10.52 /home/ted/COIN/trunk/CoinUtils/src/CoinTime.hpp File Reference

```
#include <ctime>  
#include <sys/resource.h>  
#include <sys/time.h>  
#include <fstream>
```

Classes

- class [CoinTimer](#)
This class implements a timer that also implements a tracing functionality.

Functions

- double [CoinGetTimeOfDay](#) ()
- double [CoinWallclockTime](#) (double callType=0)
Query the elapsed wallclock time since the first call to this function.
- static double [CoinCpuTime](#) ()
- static double [CoinSysTime](#) ()
- static double [CoinCpuTimeJustChildren](#) ()

10.52.1 Function Documentation

10.52.1.1 double CoinGetTimeOfDay () [inline]

Definition at line 69 of file CoinTime.hpp.

10.52.1.2 double CoinWallclockTime (double *callType* = 0) [inline]

Query the elapsed wallclock time since the first call to this function.

If a positive argument is passed to the function then the time of the first call is set to that value (this kind of argument is allowed only at the first call!). If a negative argument is passed to the function then it returns the time when it was set.

Definition at line 86 of file CoinTime.hpp.

10.52.1.3 static double CoinCpuTime () [inline],[static]

Definition at line 106 of file CoinTime.hpp.

10.52.1.4 static double CoinSysTime () [inline],[static]

Definition at line 141 of file CoinTime.hpp.

10.52.1.5 static double CoinCpuTimeJustChildren () [inline],[static]

Definition at line 161 of file CoinTime.hpp.

10.53 /home/ted/COIN/trunk/CoinUtils/src/CoinTypes.hpp File Reference

```
#include "CoinUtilsConfig.h"
```

Macros

- `#define CoinInt64 COIN_INT64_T`
- `#define CoinUInt64 COIN_UINT64_T`
- `#define CoinIntPtr COIN_INTPTR_T`
- `#define COIN_BIG_INDEX 0`
- `#define COIN_BIG_DOUBLE 0`
- `#define COIN_LONG_WORK 0`

Typedefs

- `typedef int CoinBigIndex`
- `typedef double CoinWorkDouble`
- `typedef double CoinFactorizationDouble`

10.53.1 Macro Definition Documentation

10.53.1.1 #define CoinInt64 COIN_INT64_T

Definition at line 15 of file CoinTypes.hpp.

10.53.1.2 `#define CoinUInt64 COIN_UINT64_T`

Definition at line 16 of file CoinTypes.hpp.

10.53.1.3 `#define CoinIntPtr COIN_INTPTR_T`

Definition at line 17 of file CoinTypes.hpp.

10.53.1.4 `#define COIN_BIG_INDEX 0`

Definition at line 21 of file CoinTypes.hpp.

10.53.1.5 `#define COIN_BIG_DOUBLE 0`

Definition at line 34 of file CoinTypes.hpp.

10.53.1.6 `#define COIN_LONG_WORK 0`

Definition at line 49 of file CoinTypes.hpp.

10.53.2 Typedef Documentation

10.53.2.1 `typedef int CoinBigIndex`

Definition at line 25 of file CoinTypes.hpp.

10.53.2.2 `typedef double CoinWorkDouble`

Definition at line 50 of file CoinTypes.hpp.

10.53.2.3 `typedef double CoinFactorizationDouble`

Definition at line 54 of file CoinTypes.hpp.

10.54 `/home/ted/COIN/trunk/CoinUtils/src/CoinUtility.hpp` File Reference

```
#include "CoinSort.hpp"
```

Functions

- `template<typename S, typename T>`
[CoinPair](#)< S, T > [CoinMakePair](#) (const S &s, const T &t)
- `template<typename S, typename T, typename U>`
[CoinTriple](#)< S, T, U > [CoinMakeTriple](#) (const S &s, const T &t, const U &u)

10.54.1 Function Documentation

10.54.1.1 `template<typename S, typename T> CoinPair<S,T> CoinMakePair (const S & s, const T & t)`

Definition at line 12 of file CoinUtility.hpp.

10.54.1.2 `template<typename S , typename T , typename U > CoinTriple<S,T,U> CoinMakeTriple (const S & s, const T & t, const U & u)`

Definition at line 16 of file `CoinUtility.hpp`.

10.55 /home/ted/COIN/trunk/CoinUtils/src/CoinUtilsConfig.h File Reference

```
#include "config_coinutils_default.h"
```

10.56 /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart.hpp File Reference

Copyright (C) 2000 – 2003, International Business Machines Corporation and others.

Classes

- class [CoinWarmStart](#)
Abstract base class for warm start information.
- class [CoinWarmStartDiff](#)
Abstract base class for warm start 'diff' objects.

10.56.1 Detailed Description

Copyright (C) 2000 – 2003, International Business Machines Corporation and others. All Rights Reserved. This code is licensed under the terms of the Eclipse Public License (EPL).

Declaration of the generic simplex (basis-oriented) warm start class. Also contains a basis diff class.

Definition in file [CoinWarmStart.hpp](#).

10.57 /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartBasis.hpp File Reference

```
#include <vector>
#include "CoinSort.hpp"
#include "CoinHelperFunctions.hpp"
#include "CoinWarmStart.hpp"
```

Classes

- class [CoinWarmStartBasis](#)
The default COIN simplex (basis-oriented) warm start class.
- class [CoinWarmStartBasisDiff](#)
A 'diff' between two [CoinWarmStartBasis](#) objects.

10.58 /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartDual.hpp File Reference

```
#include "CoinHelperFunctions.hpp"
#include "CoinWarmStart.hpp"
#include "CoinWarmStartVector.hpp"
```

Classes

- class [CoinWarmStartDual](#)
WarmStart information that is only a dual vector.
- class [CoinWarmStartDualDiff](#)
A 'diff' between two [CoinWarmStartDual](#) objects.

10.59 /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartPrimalDual.hpp File Reference

```
#include "CoinHelperFunctions.hpp"
#include "CoinWarmStart.hpp"
#include "CoinWarmStartVector.hpp"
```

Classes

- class [CoinWarmStartPrimalDual](#)
WarmStart information that is only a dual vector.
- class [CoinWarmStartPrimalDualDiff](#)
A 'diff' between two [CoinWarmStartPrimalDual](#) objects.

10.60 /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartVector.hpp File Reference

```
#include <cassert>
#include <cmath>
#include "CoinHelperFunctions.hpp"
#include "CoinWarmStart.hpp"
```

Classes

- class [CoinWarmStartVector< T >](#)
WarmStart information that is only a vector.
- class [CoinWarmStartVectorDiff< T >](#)
A 'diff' between two [CoinWarmStartVector](#) objects.
- class [CoinWarmStartVectorPair< T, U >](#)
- class [CoinWarmStartVectorPairDiff< T, U >](#)

10.61 /home/ted/COIN/trunk/CoinUtils/src/config_coinutils_default.h File Reference

Macros

- `#define COINUTILS_VERSION "trunk"`
- `#define COINUTILS_VERSION_MAJOR 9999`
- `#define COINUTILS_VERSION_MINOR 9999`
- `#define COINUTILS_VERSION_RELEASE 9999`
- `#define COIN_INT64_T long long`
- `#define COIN_UINT64_T unsigned long long`
- `#define COIN_INTPTR_T int*`

10.61.1 Macro Definition Documentation

10.61.1.1 `#define COINUTILS_VERSION "trunk"`

Definition at line 8 of file config_coinutils_default.h.

10.61.1.2 `#define COINUTILS_VERSION_MAJOR 9999`

Definition at line 11 of file config_coinutils_default.h.

10.61.1.3 `#define COINUTILS_VERSION_MINOR 9999`

Definition at line 14 of file config_coinutils_default.h.

10.61.1.4 `#define COINUTILS_VERSION_RELEASE 9999`

Definition at line 17 of file config_coinutils_default.h.

10.61.1.5 `#define COIN_INT64_T long long`

Definition at line 36 of file config_coinutils_default.h.

10.61.1.6 `#define COIN_UINT64_T unsigned long long`

Definition at line 37 of file config_coinutils_default.h.

10.61.1.7 `#define COIN_INTPTR_T int*`

Definition at line 38 of file config_coinutils_default.h.

10.62 /home/ted/COIN/trunk/CoinUtils/src/config_default.h File Reference

```
#include "configall_system.h"
#include "config_coinutils_default.h"
```

Macros

- `#define COIN_COINUTILS_CHECKLEVEL 0`
- `#define COIN_COINUTILS_VERBOSITY 0`

10.62.1 Macro Definition Documentation

10.62.1.1 `#define COIN_COINUTILS_CHECKLEVEL 0`

Definition at line 14 of file `config_default.h`.

10.62.1.2 `#define COIN_COINUTILS_VERBOSITY 0`

Definition at line 17 of file `config_default.h`.

Index

- ~BitVector128
 - BitVector128, [44](#)
- ~CoinAbsFltEq
 - CoinAbsFltEq, [46](#)
- ~CoinArrayWithLength
 - CoinArrayWithLength, [52](#)
- ~CoinBaseModel
 - CoinBaseModel, [56](#)
- ~CoinBuild
 - CoinBuild, [62](#)
- ~CoinDenseFactorization
 - CoinDenseFactorization, [66](#)
- ~CoinDenseVector
 - CoinDenseVector, [71](#)
- ~CoinError
 - CoinError, [77](#)
- ~CoinFactorization
 - CoinFactorization, [92](#)
- ~CoinFileOBase
 - CoinFileOBase, [122](#)
- ~CoinFileInput
 - CoinFileInput, [120](#)
- ~CoinFileOutput
 - CoinFileOutput, [124](#)
- ~CoinIndexedVector
 - CoinIndexedVector, [138](#)
- ~CoinLpIO
 - CoinLpIO, [155](#)
- ~CoinMessageHandler
 - CoinMessageHandler, [173](#)
- ~CoinMessages
 - CoinMessages, [181](#)
- ~CoinModel
 - CoinModel, [191](#)
- ~CoinModelHash
 - CoinModelHash, [208](#)
- ~CoinModelHash2
 - CoinModelHash2, [210](#)
- ~CoinModelLink
 - CoinModelLink, [214](#)
- ~CoinModelLinkedList
 - CoinModelLinkedList, [217](#)
- ~CoinMpsCardReader
 - CoinMpsCardReader, [223](#)
- ~CoinMpsIO
 - CoinMpsIO, [234](#)
- ~CoinOneMessage
 - CoinOneMessage, [248](#)
- ~CoinOslFactorization
 - CoinOslFactorization, [252](#)
- ~CoinOtherFactorization
 - CoinOtherFactorization, [260](#)
- ~CoinPackedMatrix
 - CoinPackedMatrix, [275](#)
- ~CoinPackedVector
 - CoinPackedVector, [293](#)
- ~CoinPackedVectorBase
 - CoinPackedVectorBase, [299](#)
- ~CoinParam
 - CoinParam, [308](#)
- ~CoinPartitionedVector
 - CoinPartitionedVector, [316](#)
- ~CoinPostsolveMatrix
 - CoinPostsolveMatrix, [321](#)
- ~CoinPrePostsolveMatrix
 - CoinPrePostsolveMatrix, [329](#)
- ~CoinPresolveAction
 - CoinPresolveAction, [339](#)
- ~CoinPresolveMatrix
 - CoinPresolveMatrix, [346](#)
- ~CoinRelFltEq
 - CoinRelFltEq, [359](#)
- ~CoinSearchTree
 - CoinSearchTree, [360](#)
- ~CoinSearchTreeBase
 - CoinSearchTreeBase, [362](#)
- ~CoinSearchTreeManager
 - CoinSearchTreeManager, [367](#)
- ~CoinSet
 - CoinSet, [369](#)
- ~CoinShallowPackedVector
 - CoinShallowPackedVector, [373](#)
- ~CoinSimpFactorization
 - CoinSimpFactorization, [380](#)
- ~CoinSnapshot
 - CoinSnapshot, [396](#)
- ~CoinSosSet
 - CoinSosSet, [404](#)
- ~CoinStructuredModel
 - CoinStructuredModel, [406](#)
- ~CoinThreadRandom
 - CoinThreadRandom, [411](#)
- ~CoinTreeNode
 - CoinTreeNode, [415](#)
- ~CoinTreeSiblings
 - CoinTreeSiblings, [416](#)
- ~CoinWarmStart
 - CoinWarmStart, [423](#)
- ~CoinWarmStartBasis
 - CoinWarmStartBasis, [427](#)
- ~CoinWarmStartBasisDiff
 - CoinWarmStartBasisDiff, [433](#)

- ~CoinWarmStartDiff
 - CoinWarmStartDiff, [434](#)
- ~CoinWarmStartDual
 - CoinWarmStartDual, [436](#)
- ~CoinWarmStartDualDiff
 - CoinWarmStartDualDiff, [438](#)
- ~CoinWarmStartPrimalDual
 - CoinWarmStartPrimalDual, [440](#)
- ~CoinWarmStartPrimalDualDiff
 - CoinWarmStartPrimalDualDiff, [443](#)
- ~CoinWarmStartVector
 - CoinWarmStartVector, [445](#)
- ~CoinWarmStartVectorDiff
 - CoinWarmStartVectorDiff, [448](#)
- ~CoinWarmStartVectorPair
 - CoinWarmStartVectorPair, [450](#)
- ~CoinWarmStartVectorPairDiff
 - CoinWarmStartVectorPairDiff, [452](#)
- ~CoinYacc
 - CoinYacc, [453](#)
- ~FactorPointers
 - FactorPointers, [465](#)
- ~ReferencedObject
 - Coin::ReferencedObject, [476](#)
- ~SmartPtr
 - Coin::SmartPtr, [487](#)
- ~do_tighten_action
 - do_tighten_action, [454](#)
- ~doubleton_action
 - doubleton_action, [456](#)
- ~drop_empty_cols_action
 - drop_empty_cols_action, [457](#)
- ~drop_empty_rows_action
 - drop_empty_rows_action, [459](#)
- ~drop_zero_coefficients_action
 - drop_zero_coefficients_action, [460](#)
- ~dupcol_action
 - dupcol_action, [462](#)
- ~forcing_constraint_action
 - forcing_constraint_action, [467](#)
- ~implied_free_action
 - implied_free_action, [469](#)
- ~isolated_constraint_action
 - isolated_constraint_action, [470](#)
- ~make_fixed_action
 - make_fixed_action, [472](#)
- ~remove_dual_action
 - remove_dual_action, [478](#)
- ~remove_fixed_action
 - remove_fixed_action, [480](#)
- ~slack_doubleton_action
 - slack_doubleton_action, [482](#)
- ~slack_singleton_action
 - slack_singleton_action, [483](#)
- ~subst_constraint_action
 - subst_constraint_action, [489](#)
- ~tripleton_action
 - tripleton_action, [492](#)
- ~twoxtwo_action
 - twoxtwo_action, [493](#)
- ~useless_constraint_action
 - useless_constraint_action, [494](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinAlloc.hpp, [497](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinBuild.hpp, [498](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinDenseFactorization.-
hpp, [498](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinDenseVector.-
hpp, [498](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinDistance.hpp,
[501](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinError.hpp, [502](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinFactorization.-
hpp, [503](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinFileIO.hpp, [504](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinFinite.hpp, [504](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinFloatEqual.hpp,
[505](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinHelperFunctions.-
hpp, [506](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinIndexedVector.-
hpp, [513](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinLpIO.hpp, [514](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinMessage.hpp,
[515](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinMessageHandler.-
hpp, [516](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinModel.hpp, [518](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinModelUseful.-
hpp, [519](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinMpsIO.hpp, [520](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinOslC.h, [523](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinOslFactorization.-
hpp, [527](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinPackedMatrix.-
hpp, [528](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinPackedVector.-
hpp, [528](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinPackedVector-
Base.hpp, [532](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinParam.hpp, [532](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinPragma.hpp, [534](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDoubleton.-
hpp, [534](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDual.-
hpp, [534](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveDupcol.-
hpp, [534](#)
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveEmpty.-

- hpp, 535
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveFixed.-hpp, 536
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveForcing.-hpp, 536
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveImplied-Free.hpp, 537
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveIsolated.-hpp, 537
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMatrix.-hpp, 537
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveMonitor.-hpp, 540
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolvePsdebug.-hpp, 540
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSingleton.-hpp, 540
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveSubst.-hpp, 540
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTighten.-hpp, 541
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveTriplet.-hpp, 541
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveUseless.-hpp, 542
- /home/ted/COIN/trunk/CoinUtils/src/CoinPresolveZeros.-hpp, 542
- /home/ted/COIN/trunk/CoinUtils/src/CoinSearchTree.hpp, 543
- /home/ted/COIN/trunk/CoinUtils/src/CoinShallowPacked-Vector.hpp, 544
- /home/ted/COIN/trunk/CoinUtils/src/CoinSignal.hpp, 545
- /home/ted/COIN/trunk/CoinUtils/src/CoinSimpFactorization.-hpp, 545
- /home/ted/COIN/trunk/CoinUtils/src/CoinSmartPtr.hpp, 545
- /home/ted/COIN/trunk/CoinUtils/src/CoinSnapshot.hpp, 547
- /home/ted/COIN/trunk/CoinUtils/src/CoinSort.hpp, 547
- /home/ted/COIN/trunk/CoinUtils/src/CoinStructured-Model.hpp, 550
- /home/ted/COIN/trunk/CoinUtils/src/CoinTime.hpp, 550
- /home/ted/COIN/trunk/CoinUtils/src/CoinTypes.hpp, 551
- /home/ted/COIN/trunk/CoinUtils/src/CoinUtility.hpp, 552
- /home/ted/COIN/trunk/CoinUtils/src/CoinUtilsConfig.h, 553
- /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart.hpp, 553
- /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart-Basis.hpp, 553
- /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStartDual.-hpp, 554
- /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart-PrimalDual.hpp, 554
- /home/ted/COIN/trunk/CoinUtils/src/CoinWarmStart-Vector.hpp, 554
- /home/ted/COIN/trunk/CoinUtils/src/Coin_C_defines.h, 495
- /home/ted/COIN/trunk/CoinUtils/src/config_coinutils_-default.h, 555
- /home/ted/COIN/trunk/CoinUtils/src/config_default.h, 555
- _EKKfactinfo, 32
- areaFactor, 34
- back, 35
- bitArray, 35
- demark, 34
- drtpiv, 34
- eta_size, 39
- first_dense, 37
- firstDoRow, 36
- firstLRow, 36
- firstNonSlack, 38
- hpivcoR, 36
- if_sparse_update, 38
- ifvsol, 38
- invok, 37
- iter0, 37
- iterin, 37
- iterno, 37
- kadrpm, 35
- kcpadr, 35
- kmxeta, 37
- kp1adr, 36
- kp2adr, 36
- krpadr, 35
- kw1adr, 36
- kw2adr, 36
- kw3adr, 36
- last_dense, 37
- last_eta_size, 39
- lastEtaCount, 38
- lastSlack, 38
- lstart, 38
- maxNNetas, 39
- maxinv, 36
- mpermu, 35
- nR_etas, 38
- nbfinv, 37
- ndenuc, 37
- nnentl, 37
- nnentu, 37
- nnetas, 36
- nonzero, 35
- npivots, 37
- nrow, 36
- nrowmx, 36
- num_resets, 37
- numberSlacks, 38

- nuspike, [38](#)
- packedMode, [38](#)
- R_etas_element, [35](#)
- R_etas_index, [35](#)
- R_etas_start, [35](#)
- rows_ok, [38](#)
- sortedEta, [38](#)
- switch_off_sparse_update, [38](#)
- trueStart, [35](#)
- xcnadr, [35](#)
- xcsadr, [34](#)
- xe2adr, [36](#)
- xecadr, [35](#)
- xeeadr, [36](#)
- xeradr, [35](#)
- xnetal, [37](#)
- xnetalval, [38](#)
- xrnadr, [34](#)
- xrsadr, [34](#)
- zeroTolerance, [34](#)
- zpivlu, [34](#)
- __GNUC_PREREQ
 - CoinError.hpp, [502](#)
- __STRING
 - CoinError.hpp, [502](#)
- actions_
 - doubleton_action, [456](#)
 - remove_fixed_action, [481](#)
 - tripleton_action, [492](#)
- acts_
 - CoinPrePostsolveMatrix, [336](#)
- add
 - CoinFactorization, [99](#)
 - CoinIndexedVector, [140](#)
- addBlock
 - CoinStructuredModel, [406](#), [407](#)
- addCol
 - CoinBuild, [63](#)
 - CoinModel, [191](#)
 - CoinPresolveMatrix, [349](#)
- addColumn
 - CoinBuild, [62](#)
 - CoinFactorization, [100](#)
 - CoinModel, [191](#)
- addColumnBlock
 - CoinStructuredModel, [408](#)
- addEasy
 - CoinModelLinkedList, [219](#)
- addHard
 - CoinModelLinkedList, [219](#)
- addHash
 - CoinModelHash, [209](#)
 - CoinModelHash2, [211](#)
- addLink
 - CoinFactorization, [102](#)
- addMessage
 - CoinMessages, [181](#)
- AddRef
 - Coin::ReferencedObject, [476](#)
- addRow
 - CoinBuild, [62](#)
 - CoinFactorization, [100](#)
 - CoinModel, [191](#)
 - CoinPresolveMatrix, [351](#)
- addRowBlock
 - CoinStructuredModel, [408](#)
- addString
 - CoinMpsIO, [243](#)
- adjustedAreaFactor
 - CoinFactorization, [95](#)
- advanceNode
 - CoinTreeSiblings, [416](#)
- alignment_
 - CoinArrayWithLength, [54](#)
- allocate
 - CoinArrayWithLength, [53](#)
- allocateSomeArrays
 - CoinSimpFactorization, [384](#)
- allocateSpaceForU
 - CoinSimpFactorization, [384](#)
- allowStringElements
 - CoinMpsIO, [238](#)
- allowStringElements_
 - CoinMpsIO, [246](#)
- almostDestructor
 - CoinFactorization, [92](#)
- anyInteger
 - CoinPresolveMatrix, [347](#)
- anyInteger_
 - CoinPresolveMatrix, [353](#)
- anyProhibited
 - CoinPresolveMatrix, [352](#)
- anyProhibited_
 - CoinPresolveMatrix, [355](#)
- append
 - CoinDenseVector, [72](#)
 - CoinIndexedVector, [142](#)
 - CoinPackedVector, [295](#)
- appendCol
 - CoinPackedMatrix, [277](#)
- appendCols
 - CoinPackedMatrix, [277](#), [278](#)
- appendKwd
 - CoinParam, [308](#)
- appendMajor
 - CoinPackedMatrix, [286](#)
- appendMajorVector

- CoinPackedMatrix, 282
- appendMajorVectors
 - CoinPackedMatrix, 282
- appendMinor
 - CoinPackedMatrix, 287
- appendMinorFast
 - CoinPackedMatrix, 283
- appendMinorVector
 - CoinPackedMatrix, 282, 283
- appendMinorVectors
 - CoinPackedMatrix, 283
- appendRow
 - CoinPackedMatrix, 278
- appendRows
 - CoinPackedMatrix, 278
- applyDiff
 - CoinWarmStart, 423
 - CoinWarmStartBasis, 429
 - CoinWarmStartDual, 437
 - CoinWarmStartPrimalDual, 441
 - CoinWarmStartVector, 446
 - CoinWarmStartVectorPair, 451
- are_invalid_names
 - CoinLpIO, 159
- areaFactor
 - _EKKfactinfo, 34
 - CoinFactorization, 95
- areaFactor_
 - CoinFactorization, 105
- array
 - CoinArbitraryArrayWithLength, 48
 - CoinArrayWithLength, 52
 - CoinBigIndexArrayWithLength, 60
 - CoinDoubleArrayWithLength, 75
 - CoinFactorizationDoubleArrayWithLength, 116
 - CoinFactorizationLongDoubleArrayWithLength, 118
 - CoinIntArrayWithLength, 147
 - CoinUnsignedIntArrayWithLength, 420
 - CoinVoidStarArrayWithLength, 422
- array_
 - CoinArrayWithLength, 53
- artificialStatus_
 - CoinWarmStartBasis, 431
- assign
 - CoinWarmStartPrimalDual, 441
- assignBasisStatus
 - CoinWarmStartBasis, 430
- assignDual
 - CoinWarmStartDual, 436
- assignMatrix
 - CoinPackedMatrix, 280
- assignPresolveToPostsolve
 - CoinPostsolveMatrix, 321
 - CoinPresolveMatrix, 352
- assignVector
 - CoinPackedVector, 294
 - CoinWarmStartVector, 445
- assignVector0
 - CoinWarmStartVectorPair, 450
- assignVector1
 - CoinWarmStartVectorPair, 450
- associateElement
 - CoinModel, 192
- associatedArray
 - CoinModel, 203
- atLowerBound
 - CoinPrePostsolveMatrix, 328
 - CoinWarmStartBasis, 427
- atUpperBound
 - CoinPrePostsolveMatrix, 328
 - CoinWarmStartBasis, 427
- auxInd_
 - CoinSimpFactorization, 386
- auxVector_
 - CoinSimpFactorization, 386
- back
 - _EKKfactinfo, 35
- baseL
 - CoinFactorization, 95
- baseL_
 - CoinFactorization, 110
- basic
 - CoinPrePostsolveMatrix, 328
 - CoinWarmStartBasis, 427
- bestQuality
 - CoinSearchTreeManager, 367
- bestQualityCandidate
 - CoinSearchTreeManager, 367
- biasLU
 - CoinFactorization, 98
- biasLU_
 - CoinFactorization, 114
- biggerDimension_
 - CoinFactorization, 109
- binaryOp
 - CoinPackedVector.hpp, 530
- bitArray
 - _EKKfactinfo, 35
- BitVector128, 44
 - ~BitVector128, 44
 - BitVector128, 44
 - BitVector128, 44
 - clearBit, 45
 - operator<, 45
 - set, 45
 - setBit, 45
 - str, 45

- block
 - CoinStructuredModel, [409](#)
- blockIndex
 - CoinStructuredModel, [409](#)
- blockType
 - CoinStructuredModel, [408](#)
- borrowVector
 - CoinIndexedVector, [139](#)
- bottomAppendPackedMatrix
 - CoinPackedMatrix, [278](#)
- boundName_
 - CoinMpsIO, [244](#)
- bounds
 - CoinModelInfo2, [213](#)
 - forcing_constraint_action::action, [42](#)
- btran
 - CoinSimpFactorization, [386](#)
- btranAverageAfterL_
 - CoinFactorization, [113](#)
- btranAverageAfterR_
 - CoinFactorization, [113](#)
- btranAverageAfterU_
 - CoinFactorization, [113](#)
- btranCountAfterL_
 - CoinFactorization, [113](#)
- btranCountAfterR_
 - CoinFactorization, [112](#)
- btranCountAfterU_
 - CoinFactorization, [112](#)
- btranCountInput_
 - CoinFactorization, [112](#)
- bulk0_
 - CoinPrePostsolveMatrix, [334](#)
- bulkRatio_
 - CoinPrePostsolveMatrix, [334](#)
- COIN_BASIS_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_BLANK_COLUMN
 - CoinMpsIO.hpp, [522](#)
- COIN_BOTH_BOUNDS_SET
 - CoinMpsIO.hpp, [523](#)
- COIN_BOUNDS_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_BS_BASIS
 - CoinMpsIO.hpp, [523](#)
- COIN_BV_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_COLUMN_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_CONIC_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_DUMMY_END
 - CoinMessage.hpp, [516](#)
- COIN_E_ROW
 - CoinMpsIO.hpp, [522](#)
- COIN_ENDATA_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_EOF_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_FR_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_FX_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_G_ROW
 - CoinMpsIO.hpp, [522](#)
- COIN_GENERAL_INFO
 - CoinMessage.hpp, [516](#)
- COIN_GENERAL_WARNING
 - CoinMessage.hpp, [516](#)
- COIN_INTEND
 - CoinMpsIO.hpp, [522](#)
- COIN_INTORG
 - CoinMpsIO.hpp, [522](#)
- COIN_L_ROW
 - CoinMpsIO.hpp, [522](#)
- COIN_LI_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_LL_BASIS
 - CoinMpsIO.hpp, [523](#)
- COIN_LO_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_MI_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_MPS_BADFILE1
 - CoinMessage.hpp, [516](#)
- COIN_MPS_BADFILE2
 - CoinMessage.hpp, [516](#)
- COIN_MPS_BADIMAGE
 - CoinMessage.hpp, [516](#)
- COIN_MPS_CHANGED
 - CoinMessage.hpp, [516](#)
- COIN_MPS_DUOBJ
 - CoinMessage.hpp, [516](#)
- COIN_MPS_DUPROW
 - CoinMessage.hpp, [516](#)
- COIN_MPS_EOF
 - CoinMessage.hpp, [516](#)
- COIN_MPS_FILE
 - CoinMessage.hpp, [516](#)
- COIN_MPS_ILLEGAL
 - CoinMessage.hpp, [515](#)
- COIN_MPS_LINE
 - CoinMessage.hpp, [515](#)
- COIN_MPS_NOMATCHCOL
 - CoinMessage.hpp, [516](#)
- COIN_MPS_NOMATCHROW
 - CoinMessage.hpp, [516](#)

- COIN_MPS_RETURNING
 - CoinMessage.hpp, [516](#)
- COIN_MPS_STATS
 - CoinMessage.hpp, [515](#)
- COIN_N_ROW
 - CoinMpsIO.hpp, [522](#)
- COIN_NAME_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_NO_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_PL_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_PREOLVE_COLINFEAS
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_COLUMNBOUNDA
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_COLUMNBOUNDB
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_INFEAS
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_INFEASUNBOUND
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_INTEGERMODS
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_NEEDS_CLEANING
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_NOOPTIMAL
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_PASS
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_POSTSOLVE
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_ROWINFEAS
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_STATS
 - CoinMessage.hpp, [516](#)
- COIN_PREOLVE_UNBOUND
 - CoinMessage.hpp, [516](#)
- COIN_QUAD_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_QUADRATIC_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_RANGES_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_RHS_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_ROW_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_S1_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_S1_COLUMN
 - CoinMpsIO.hpp, [522](#)
- COIN_S2_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_S2_COLUMN
 - CoinMpsIO.hpp, [522](#)
- COIN_S3_COLUMN
 - CoinMpsIO.hpp, [522](#)
- COIN_SC_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_SOLVER_MPS
 - CoinMessage.hpp, [516](#)
- COIN_SOS_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_SOSEND
 - CoinMpsIO.hpp, [523](#)
- COIN_UI_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_UL_BASIS
 - CoinMpsIO.hpp, [523](#)
- COIN_UNKNOWN_MPS_TYPE
 - CoinMpsIO.hpp, [523](#)
- COIN_UNKNOWN_SECTION
 - CoinMpsIO.hpp, [522](#)
- COIN_UNSET_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_UP_BOUND
 - CoinMpsIO.hpp, [523](#)
- COIN_XL_BASIS
 - CoinMpsIO.hpp, [523](#)
- COIN_XU_BASIS
 - CoinMpsIO.hpp, [523](#)
- COMPRESS_BZIP2
 - CoinFileOutput, [124](#)
- COMPRESS_GZIP
 - CoinFileOutput, [124](#)
- COMPRESS_NONE
 - CoinFileOutput, [124](#)
- C_EKK_ADD_LINK
 - CoinOsIC.h, [525](#)
- C_EKK_GO_SPARSE
 - CoinOsIC.h, [525](#)
- C_EKK_REMOVE_LINK
 - CoinOsIC.h, [525](#)
- c_ekk_IsSet
 - CoinOsIC.h, [527](#)
- c_ekk_Set
 - CoinOsIC.h, [527](#)
- c_ekk_Unset
 - CoinOsIC.h, [527](#)
- c_ekkbtrn
 - CoinOsIC.h, [526](#)
- c_ekkbtrn_ipivrw
 - CoinOsIC.h, [526](#)
- c_ekkczero
 - CoinOsIC.h, [527](#)
- c_ekkdcpv
 - CoinOsIC.h, [526](#)

- c_ekkdzero
 - CoinOslC.h, [527](#)
- c_ekketsj
 - CoinOslC.h, [526](#)
- c_ekkftn
 - CoinOslC.h, [526](#)
- c_ekkftn2
 - CoinOslC.h, [526](#)
- c_ekkftn_ft
 - CoinOslC.h, [526](#)
- c_ekkizero
 - CoinOslC.h, [527](#)
- c_ekklfct
 - CoinOslC.h, [526](#)
- c_ekks1cpy
 - CoinOslC.h, [525](#)
- c_ekkscpy
 - CoinOslC.h, [526](#)
- c_ekkscpy_0_1
 - CoinOslC.h, [525](#)
- c_ekkslcf
 - CoinOslC.h, [526](#)
- c_ekkzero
 - CoinOslC.h, [527](#)
- CLP_OSL
 - CoinOslC.h, [525](#)
- COIN_BIG_DOUBLE
 - CoinTypes.hpp, [552](#)
- COIN_BIG_INDEX
 - CoinTypes.hpp, [552](#)
- COIN_DBL_MAX
 - CoinFinite.hpp, [505](#)
- COIN_DBL_MIN
 - CoinFinite.hpp, [505](#)
- COIN_DETAIL_PRINT
 - CoinHelperFunctions.hpp, [508](#)
- COIN_INT64_T
 - config_coinutils_default.h, [555](#)
- COIN_INT_MAX
 - CoinFinite.hpp, [505](#)
- COIN_INTPTR_T
 - config_coinutils_default.h, [555](#)
- COIN_LONG_WORK
 - CoinTypes.hpp, [552](#)
- COIN_Message
 - CoinMessage.hpp, [515](#)
- COIN_NUM_LOG
 - CoinMessageHandler.hpp, [517](#)
- COIN_OWN_RANDOM_32
 - CoinHelperFunctions.hpp, [508](#)
- COIN_PARTITIONS
 - CoinIndexedVector.hpp, [514](#)
- COIN_RESTRICT
 - CoinHelperFunctions.hpp, [508](#)
- COIN_UINT64_T
 - config_coinutils_default.h, [555](#)
- COINColumnIndex
 - CoinLpIO.hpp, [514](#)
 - CoinMpsIO.hpp, [521](#)
- COINLIBAPI
 - Coin_C_defines.h, [496](#)
- COINLINKAGE
 - Coin_C_defines.h, [496](#)
- COINLINKAGE_CB
 - Coin_C_defines.h, [496](#)
- COINMpsType
 - CoinMpsIO.hpp, [522](#)
- COINRowIndex
 - CoinMpsIO.hpp, [522](#)
- COINSectionType
 - CoinMpsIO.hpp, [522](#)
- COINUTILS_VERSION
 - config_coinutils_default.h, [555](#)
- candidateList_
 - CoinSearchTreeBase, [363](#)
- capacity
 - CoinArrayWithLength, [52](#)
 - CoinIndexedVector, [144](#)
 - CoinPackedVector, [296](#)
- capacity_
 - CoinIndexedVector, [145](#)
- card
 - CoinMpsCardReader, [224](#)
- card_
 - CoinMpsCardReader, [225](#)
- card_previous_names_
 - CoinLpIO, [167](#)
- cardNumber
 - CoinMpsCardReader, [225](#)
- cardNumber_
 - CoinMpsCardReader, [226](#)
- cardReader_
 - CoinMpsIO, [246](#)
- Cbc_Model
 - Coin_C_defines.h, [497](#)
- cdone_
 - CoinPostsolveMatrix, [322](#)
- change_bias
 - CoinPresolveMatrix, [347](#)
- charValue
 - CoinMessageHandler, [175](#)
- charValue_
 - CoinMessageHandler, [177](#)
- check_nbasic
 - CoinPostsolveMatrix, [321](#)
- checkAndTell
 - CoinPresolveMonitor, [358](#)
- checkClean

- CoinIndexedVector, [141](#)
- CoinPartitionedVector, [317](#)
- checkClear
 - CoinIndexedVector, [141](#)
 - CoinPartitionedVector, [317](#)
- checkColNames
 - CoinLpIO, [164](#)
- checkConsistency
 - CoinFactorization, [102](#)
- checkPivot
 - CoinDenseFactorization, [68](#)
 - CoinFactorization, [105](#)
 - CoinOslFactorization, [256](#)
 - CoinSimpFactorization, [386](#)
- checkRowNames
 - CoinLpIO, [163](#)
- checkSeverity
 - CoinMessageHandler, [173](#)
- checkSparse
 - CoinFactorization, [100](#)
- class_
 - CoinMessages, [182](#)
- className
 - CoinError, [77](#)
- clean
 - CoinIndexedVector, [140](#)
- cleanAndPack
 - CoinIndexedVector, [141](#)
- cleanAndPackSafe
 - CoinIndexedVector, [141](#)
- cleanCard
 - CoinMpsCardReader, [224](#)
- cleanMatrix
 - CoinPackedMatrix, [279](#)
- cleanup
 - CoinFactorization, [102](#)
- clear
 - CoinArrayWithLength, [53](#)
 - CoinDenseVector, [72](#)
 - CoinIndexedVector, [139](#)
 - CoinPackedMatrix, [275](#)
 - CoinPackedVector, [294](#)
 - CoinShallowPackedVector, [373](#)
 - CoinWarmStartPrimalDual, [441](#)
 - CoinWarmStartPrimalDualDiff, [443](#)
 - CoinWarmStartVector, [446](#)
 - CoinWarmStartVectorDiff, [448](#)
 - CoinWarmStartVectorPair, [451](#)
 - CoinWarmStartVectorPairDiff, [452](#)
- clearAndKeep
 - CoinPartitionedVector, [317](#)
- clearAndReset
 - CoinPartitionedVector, [317](#)
- clearArrays
 - CoinDenseFactorization, [68](#)
 - CoinFactorization, [99](#)
 - CoinOslFactorization, [255](#)
 - CoinOtherFactorization, [263](#)
 - CoinSimpFactorization, [382](#)
- clearBase
 - CoinPackedVectorBase, [302](#)
- clearBit
 - BitVector128, [45](#)
- clearIndexSet
 - CoinPackedVectorBase, [302](#)
- clearPartition
 - CoinPartitionedVector, [317](#)
- clink_
 - CoinPresolveMatrix, [352](#)
- clo_
 - CoinPrePostsolveMatrix, [335](#)
- clone
 - CoinBaseModel, [56](#)
 - CoinDenseFactorization, [66](#)
 - CoinMessageHandler, [173](#)
 - CoinModel, [206](#)
 - CoinOslFactorization, [253](#)
 - CoinOtherFactorization, [260](#)
 - CoinParam, [308](#)
 - CoinSimpFactorization, [380](#)
 - CoinStructuredModel, [409](#)
 - CoinWarmStart, [423](#)
 - CoinWarmStartBasis, [430](#)
 - CoinWarmStartBasisDiff, [433](#)
 - CoinWarmStartDiff, [435](#)
 - CoinWarmStartDual, [436](#)
 - CoinWarmStartDualDiff, [438](#)
 - CoinWarmStartPrimalDual, [441](#)
 - CoinWarmStartPrimalDualDiff, [443](#)
 - CoinWarmStartVector, [446](#)
 - CoinWarmStartVectorDiff, [448](#)
 - CoinWarmStartVectorPair, [451](#)
 - CoinWarmStartVectorPairDiff, [452](#)
- clox
 - doubleton_action::action, [39](#)
 - tripleton_action::action, [43](#)
- cloy
 - tripleton_action::action, [43](#)
- Clp_Simplex
 - Coin_C_defines.h, [496](#)
- clp_double
 - CoinOslC.h, [527](#)
- clp_free
 - CoinOslC.h, [527](#)
- clp_int
 - CoinOslC.h, [527](#)
- clp_malloc
 - CoinOslC.h, [527](#)

- clp_memory
 - CoinOslC.h, [527](#)
- clp_setup_pointers
 - CoinOslC.h, [527](#)
- coeffx
 - doubleton_action::action, [40](#)
 - tripleton_action::action, [43](#)
- coeffy
 - doubleton_action::action, [40](#)
 - tripleton_action::action, [43](#)
- coeffz
 - tripleton_action::action, [44](#)
- Coin, [29](#)
 - ComparePointers, [29](#)
- CoinAddNodeToCandidates
 - CoinSearchTree.hpp, [544](#)
- CoinDiveIntoNode
 - CoinSearchTree.hpp, [544](#)
- CoinFileOutput
 - COMPRESS_BZIP2, [124](#)
 - COMPRESS_GZIP, [124](#)
 - COMPRESS_NONE, [124](#)
- CoinMessage.hpp
 - COIN_DUMMY_END, [516](#)
 - COIN_GENERAL_INFO, [516](#)
 - COIN_GENERAL_WARNING, [516](#)
 - COIN_MPS_BADFILE1, [516](#)
 - COIN_MPS_BADFILE2, [516](#)
 - COIN_MPS_BADIMAGE, [516](#)
 - COIN_MPS_CHANGED, [516](#)
 - COIN_MPS_DUPOBJ, [516](#)
 - COIN_MPS_DUPROW, [516](#)
 - COIN_MPS_EOF, [516](#)
 - COIN_MPS_FILE, [516](#)
 - COIN_MPS_ILLEGAL, [515](#)
 - COIN_MPS_LINE, [515](#)
 - COIN_MPS_NOMATCHCOL, [516](#)
 - COIN_MPS_NOMATCHROW, [516](#)
 - COIN_MPS_RETURNING, [516](#)
 - COIN_MPS_STATS, [515](#)
 - COIN_PREOLVE_COLINFEAS, [516](#)
 - COIN_PREOLVE_COLUMNBOUND, [516](#)
 - COIN_PREOLVE_COLUMNBOUND, [516](#)
 - COIN_PREOLVE_INFEAS, [516](#)
 - COIN_PREOLVE_INFEASUNBOUND, [516](#)
 - COIN_PREOLVE_INTEGERMODS, [516](#)
 - COIN_PREOLVE_NEEDS_CLEANING, [516](#)
 - COIN_PREOLVE_NOOPTIMAL, [516](#)
 - COIN_PREOLVE_PASS, [516](#)
 - COIN_PREOLVE_POSTSOLVE, [516](#)
 - COIN_PREOLVE_ROWINFEAS, [516](#)
 - COIN_PREOLVE_STATS, [516](#)
 - COIN_PREOLVE_UNBOUND, [516](#)
 - COIN_SOLVER_MPS, [516](#)
- CoinMessageEol
 - CoinMessageHandler.hpp, [518](#)
- CoinMessageHandler.hpp
 - CoinMessageEol, [518](#)
 - CoinMessageNewline, [518](#)
- CoinMessageNewline
 - CoinMessageHandler.hpp, [518](#)
- CoinMessages
 - it, [181](#)
 - uk_en, [181](#)
 - us_en, [181](#)
- CoinMpsIO.hpp
 - COIN_BASIS_SECTION, [522](#)
 - COIN_BLANK_COLUMN, [522](#)
 - COIN_BOTH_BOUNDS_SET, [523](#)
 - COIN_BOUNDS_SECTION, [522](#)
 - COIN_BS_BASIS, [523](#)
 - COIN_BV_BOUND, [523](#)
 - COIN_COLUMN_SECTION, [522](#)
 - COIN_CONIC_SECTION, [522](#)
 - COIN_E_ROW, [522](#)
 - COIN_ENDATA_SECTION, [522](#)
 - COIN_EOF_SECTION, [522](#)
 - COIN_FR_BOUND, [523](#)
 - COIN_FX_BOUND, [523](#)
 - COIN_G_ROW, [522](#)
 - COIN_INTEND, [522](#)
 - COIN_INTORG, [522](#)
 - COIN_L_ROW, [522](#)
 - COIN_LI_BOUND, [523](#)
 - COIN_LL_BASIS, [523](#)
 - COIN_LO_BOUND, [523](#)
 - COIN_MI_BOUND, [523](#)
 - COIN_N_ROW, [522](#)
 - COIN_NAME_SECTION, [522](#)
 - COIN_NO_SECTION, [522](#)
 - COIN_PL_BOUND, [523](#)
 - COIN_QUAD_SECTION, [522](#)
 - COIN_QUADRATIC_SECTION, [522](#)
 - COIN_RANGES_SECTION, [522](#)
 - COIN_RHS_SECTION, [522](#)
 - COIN_ROW_SECTION, [522](#)
 - COIN_S1_BOUND, [523](#)
 - COIN_S1_COLUMN, [522](#)
 - COIN_S2_BOUND, [523](#)
 - COIN_S2_COLUMN, [522](#)
 - COIN_S3_COLUMN, [522](#)
 - COIN_SC_BOUND, [523](#)
 - COIN_SOS_SECTION, [522](#)
 - COIN_SOSEND, [523](#)
 - COIN_UI_BOUND, [523](#)
 - COIN_UL_BASIS, [523](#)
 - COIN_UNKNOWN_MPS_TYPE, [523](#)
 - COIN_UNKNOWN_SECTION, [522](#)

- COIN_UNSET_BOUND, [523](#)
- COIN_UP_BOUND, [523](#)
- COIN_XL_BASIS, [523](#)
- COIN_XU_BASIS, [523](#)
- CoinParam
 - coinParamAct, [307](#)
 - coinParamDbl, [307](#)
 - coinParamInt, [307](#)
 - coinParamInvalid, [307](#)
 - coinParamKwd, [307](#)
 - coinParamStr, [307](#)
- coinParamAct
 - CoinParam, [307](#)
- coinParamDbl
 - CoinParam, [307](#)
- coinParamInt
 - CoinParam, [307](#)
- coinParamInvalid
 - CoinParam, [307](#)
- coinParamKwd
 - CoinParam, [307](#)
- coinParamStr
 - CoinParam, [307](#)
- CoinPrePostsolveMatrix
 - atLowerBound, [328](#)
 - atUpperBound, [328](#)
 - basic, [328](#)
 - isFree, [328](#)
 - superBasic, [328](#)
- CoinSearchTree.hpp
 - CoinAddNodeToCandidates, [544](#)
 - CoinDiveIntoNode, [544](#)
 - CoinTestNodeForDiving, [544](#)
- CoinTestNodeForDiving
 - CoinSearchTree.hpp, [544](#)
- CoinWarmStartBasis
 - atLowerBound, [427](#)
 - atUpperBound, [427](#)
 - basic, [427](#)
 - isFree, [427](#)
- Coin::ReferencedObject, [474](#)
 - ~ReferencedObject, [476](#)
 - AddRef, [476](#)
 - ReferenceCount, [476](#)
 - ReferencedObject, [476](#)
 - ReleaseRef, [476](#)
- Coin::SmartPtr
 - ~SmartPtr, [487](#)
 - GetRawPtr, [487](#)
 - IsNull, [487](#)
 - IsValid, [487](#)
 - operator*, [487](#)
 - operator->, [487](#)
 - operator=, [487](#)
 - operator==, [488](#)
 - SmartPtr, [486](#)
- Coin::SmartPtr< T >, [483](#)
- Coin_C_defines.h
 - COINLIBAPI, [496](#)
 - COINLINKAGE, [496](#)
 - COINLINKAGE_CB, [496](#)
 - Cbc_Model, [497](#)
 - Clp_Simplex, [496](#)
 - CoinBigIndex, [497](#)
 - cvec, [497](#)
 - dvec, [496](#)
 - ivec, [497](#)
 - msgno, [496](#)
 - nchar, [497](#)
 - ndouble, [496](#)
 - nint, [496](#)
 - Sbb_Model, [497](#)
 - void, [497](#)
- coin_init_random_vec
 - Presolve Utility Functions, [25](#)
- CoinAbs
 - CoinHelperFunctions.hpp, [511](#)
- CoinAbsFltEq, [45](#)
 - ~CoinAbsFltEq, [46](#)
 - CoinAbsFltEq, [46](#)
 - CoinAbsFltEq, [46](#)
 - operator(), [46](#)
 - operator=, [46](#)
- CoinArbitraryArrayWithLength, [46](#)
 - array, [48](#)
 - CoinArbitraryArrayWithLength, [48](#)
 - CoinArbitraryArrayWithLength, [48](#)
 - conditionalNew, [48](#)
 - getSize, [48](#)
 - lengthInBytes_, [49](#)
 - operator=, [48](#)
 - setSize, [48](#)
- CoinArrayWithLength, [49](#)
 - ~CoinArrayWithLength, [52](#)
 - alignment_, [54](#)
 - allocate, [53](#)
 - array, [52](#)
 - array_, [53](#)
 - capacity, [52](#)
 - clear, [53](#)
 - CoinArrayWithLength, [51](#), [52](#)
 - CoinArrayWithLength, [51](#), [52](#)
 - conditionalDelete, [53](#)
 - conditionalNew, [53](#)
 - copy, [53](#)
 - extend, [53](#)
 - getArray, [53](#)
 - getCapacity, [53](#)

- getSize, 52
 - offset_, 54
 - operator=, 53
 - rawSize, 52
 - reallyFreeArray, 53
 - setCapacity, 52
 - setPersistence, 53
 - setSize, 52
 - size_, 54
 - swap, 53
 - switchOff, 52
 - switchOn, 53
 - switchedOn, 52
- CoinAssert
 - CoinError.hpp, 502
- CoinAssertDebug
 - CoinError.hpp, 502
- CoinAssertDebugHint
 - CoinError.hpp, 502
- CoinAssertHint
 - CoinError.hpp, 502
- CoinBaseModel, 54
 - ~CoinBaseModel, 56
 - clone, 56
 - CoinBaseModel, 56
 - CoinBaseModel, 56
 - columnBlockName_, 58
 - getColumnBlock, 57
 - getProblemName, 57
 - getRowBlock, 57
 - logLevel, 57
 - logLevel_, 58
 - numberColumns, 56
 - numberColumns_, 58
 - numberElements, 56
 - numberRows, 56
 - numberRows_, 58
 - objectiveOffset, 56
 - objectiveOffset_, 58
 - operator=, 56
 - optimizationDirection, 57
 - optimizationDirection_, 58
 - problemName_, 58
 - rowBlockName_, 58
 - setColumnBlock, 57
 - setLogLevel, 57
 - setObjectiveOffset, 56
 - setOptimizationDirection, 57
 - setProblemName, 57
 - setRowBlock, 57
- CoinBigIndex
 - Coin_C_defines.h, 497
 - CoinTypes.hpp, 552
- CoinBigIndexArrayWithLength, 58
 - array, 60
 - CoinBigIndexArrayWithLength, 60
 - CoinBigIndexArrayWithLength, 60
 - conditionalNew, 60
 - getSize, 60
 - operator=, 60
 - setSize, 60
- coinBlock
 - CoinStructuredModel, 409
- CoinBuild, 61
 - ~CoinBuild, 62
 - addCol, 63
 - addColumn, 62
 - addRow, 62
 - CoinBuild, 62
 - CoinBuild, 62
 - column, 63
 - currentColumn, 63, 64
 - currentRow, 63
 - numberColumns, 63
 - numberElements, 63
 - numberRows, 63
 - operator=, 64
 - row, 63
 - setCurrentColumn, 63
 - setCurrentRow, 63
 - type, 64
- CoinCheckDoubleZero
 - CoinHelperFunctions.hpp, 510
- CoinCheckIntZero
 - CoinHelperFunctions.hpp, 510
- CoinComparePointers
 - CoinSmartPtr.hpp, 546
- CoinConvertDouble
 - CoinMpsIO.hpp, 523
- CoinCopy
 - CoinHelperFunctions.hpp, 508
- CoinCopyN
 - CoinHelperFunctions.hpp, 508
- CoinCopyOfArray
 - CoinHelperFunctions.hpp, 509
- CoinCopyOfArrayOrZero
 - CoinHelperFunctions.hpp, 509
- CoinCopyOfArrayPartial
 - CoinHelperFunctions.hpp, 509
- CoinCpuTime
 - CoinTime.hpp, 551
- CoinCpuTimeJustChildren
 - CoinTime.hpp, 551
- CoinDecrSolutionOrdered
 - CoinSort.hpp, 549
- CoinDeleteEntriesFromArray
 - CoinHelperFunctions.hpp, 511
- CoinDenseFactorization, 64

- ~CoinDenseFactorization, 66
- checkPivot, 68
- clearArrays, 68
- clone, 66
- CoinDenseFactorization, 66
- CoinDenseFactorizationUnitTest, 69
- CoinDenseFactorization, 66
- factor, 67
- getAreas, 66
- gutsOfCopy, 68
- gutsOfDestructor, 68
- gutsOfInitialize, 68
- indices, 68
- makeNonSingular, 67
- maximumCoefficient, 67
- numberElements, 67
- operator=, 66
- permute, 68
- postProcess, 67
- preProcess, 67
- replaceColumn, 67
- updateColumn, 68
- updateColumnFT, 67
- updateColumnTranspose, 68
- updateTwoColumnsFT, 68
- CoinDenseFactorizationUnitTest
 - CoinDenseFactorization, 69
- CoinDenseVector
 - ~CoinDenseVector, 71
 - append, 72
 - clear, 72
 - CoinDenseVector, 71
 - CoinDenseVector, 71
 - getElements, 71
 - getNumElements, 71
 - infNorm, 72
 - oneNorm, 72
 - operator*=, 73
 - operator+=, 73
 - operator-=, 73
 - operator/=, 73
 - operator=, 72
 - resize, 72
 - scale, 73
 - setConstant, 72
 - setElement, 72
 - setVector, 72
 - size, 71
 - sum, 72
 - twoNorm, 72
- CoinDenseVector< T >, 69
- CoinDenseVector.hpp
 - CoinDenseVectorUnitTest, 499
 - operator*, 500, 501
 - operator+, 499, 500
 - operator-, 500, 501
 - operator/, 500, 501
- CoinDenseVectorUnitTest
 - CoinDenseVector.hpp, 499
- CoinDisjointCopy
 - CoinHelperFunctions.hpp, 509
- CoinDisjointCopyN
 - CoinHelperFunctions.hpp, 509
- coinDistance
 - CoinDistance.hpp, 501
- CoinDistance.hpp
 - coinDistance, 501
- CoinDoubleArrayWithLength, 73
 - array, 75
 - CoinDoubleArrayWithLength, 74
 - CoinDoubleArrayWithLength, 74
 - conditionalNew, 75
 - getSize, 75
 - operator=, 75
 - setSize, 75
- CoinDrand48
 - CoinHelperFunctions.hpp, 512
- CoinError, 75
 - ~CoinError, 77
 - className, 77
 - CoinError, 77
 - CoinErrorUnitTest, 78
 - CoinError, 77
 - fileName, 77
 - lineNumber, 77
 - message, 77
 - methodName, 77
 - operator=, 77
 - print, 77
 - printErrors_, 78
- CoinError.hpp
 - __GNUPREREQ, 502
 - __STRING, 502
 - CoinAssert, 502
 - CoinAssertDebug, 502
 - CoinAssertDebugHint, 502
 - CoinAssertHint, 502
 - CoinErrorUnitTest, 503
 - WindowsErrorPopupBlocker, 503
- CoinErrorUnitTest
 - CoinError, 78
 - CoinError.hpp, 503
- CoinExternalVectorFirstGreater_2
 - CoinExternalVectorFirstGreater_2, 78
 - CoinExternalVectorFirstGreater_2, 78
 - operator(), 79
- CoinExternalVectorFirstGreater_2< S, T, V >, 78
- CoinExternalVectorFirstGreater_3

- CoinExternalVectorFirstGreater_3, 79
- CoinExternalVectorFirstGreater_3, 79
- operator(), 79
- CoinExternalVectorFirstGreater_3< S, T, U, V >, 79
- CoinExternalVectorFirstLess_2
 - CoinExternalVectorFirstLess_2, 80
 - CoinExternalVectorFirstLess_2, 80
 - operator(), 80
- CoinExternalVectorFirstLess_2< S, T, V >, 80
- CoinExternalVectorFirstLess_3
 - CoinExternalVectorFirstLess_3, 81
 - CoinExternalVectorFirstLess_3, 81
 - operator(), 81
- CoinExternalVectorFirstLess_3< S, T, U, V >, 80
- CoinFactorization, 81
 - ~CoinFactorization, 92
 - add, 99
 - addColumn, 100
 - addLink, 102
 - addRow, 100
 - adjustedAreaFactor, 95
 - almostDestructor, 92
 - areaFactor, 95
 - areaFactor_, 105
 - baseL, 95
 - baseL_, 110
 - biasLU, 98
 - biasLU_, 114
 - biggerDimension_, 109
 - btranAverageAfterL_, 113
 - btranAverageAfterR_, 113
 - btranAverageAfterU_, 113
 - btranCountAfterL_, 113
 - btranCountAfterR_, 112
 - btranCountAfterU_, 112
 - btranCountInput_, 112
 - checkConsistency, 102
 - checkPivot, 105
 - checkSparse, 100
 - cleanup, 102
 - clearArrays, 99
 - CoinFactorization, 92
 - CoinFactorizationUnitTest, 105
 - CoinFactorization, 92
 - collectStatistics, 100
 - collectStatistics_, 113
 - conditionNumber, 93
 - convertRowToColumnU_, 110
 - deleteColumn, 100
 - deleteLink, 102
 - deleteRow, 100
 - denseArea_, 111
 - densePermute_, 111
 - denseThreshold, 96
 - denseThreshold_, 112
 - doForrestTomlin_, 111
 - elementByRowL, 94
 - elementByRowL_, 114
 - elementL_, 110
 - elementR_, 111
 - elementU, 98
 - elementU_, 110
 - emptyRows, 100
 - factor, 101
 - factorDense, 101
 - factorElements_, 107
 - factorSparse, 101
 - factorSparseLarge, 101
 - factorSparseSmall, 101
 - factorize, 92
 - factorizePart1, 93
 - factorizePart2, 93
 - firstCount_, 108
 - forrestTomlin, 96
 - ftranAverageAfterL_, 113
 - ftranAverageAfterR_, 113
 - ftranAverageAfterU_, 113
 - ftranCountAfterL_, 112
 - ftranCountAfterR_, 112
 - ftranCountAfterU_, 112
 - ftranCountInput_, 112
 - getAccuracyCheck, 95
 - getAreas, 101
 - getColumnSpace, 101
 - getColumnSpaceIterate, 102
 - getColumnSpaceIterateR, 101
 - getRowSpace, 102
 - getRowSpaceIterate, 102
 - goSparse, 99
 - gutsOfCopy, 100
 - gutsOfDestructor, 100
 - gutsOfInitialize, 100
 - indexColumnL, 94
 - indexColumnL_, 114
 - indexColumnU_, 109
 - indexRowL, 94
 - indexRowL_, 110
 - indexRowR_, 111
 - indexRowU, 98
 - indexRowU_, 110
 - lastColumn_, 108
 - lastCount_, 108
 - lastRow_, 108
 - lengthAreaL, 97
 - lengthAreaL_, 110
 - lengthAreaR_, 111
 - lengthAreaU, 97
 - lengthAreaU_, 109

lengthL_, 110
lengthR_, 111
lengthU_, 109
markRow_, 108
maximumCoefficient, 96
maximumColumnsExtra, 98
maximumColumnsExtra_, 106
maximumPivots, 96
maximumPivots_, 106
maximumRowsExtra, 95
maximumRowsExtra_, 106
maximumU_, 109
messageLevel, 96
messageLevel_, 109
nextColumn_, 108
nextCount_, 108
nextRow_, 108
numberBtranCounts_, 113
numberColumns, 95
numberColumns_, 106
numberColumnsExtra_, 106
numberCompressions, 97
numberCompressions_, 112
numberDense, 97
numberDense_, 112
numberElements, 95
numberElementsL, 97
numberElementsR, 97
numberElementsU, 97
numberForrestTomlin, 95
numberFtranCounts_, 113
numberGoodColumns, 95
numberGoodL_, 106
numberGoodU_, 106
numberInColumn, 98
numberInColumn_, 107
numberInColumnPlus_, 108
numberInRow, 98
numberInRow_, 107
numberL, 94
numberL_, 110
numberPivots_, 106
numberR_, 111
numberRows, 94
numberRows_, 106
numberRowsExtra, 94
numberRowsExtra_, 106
numberSlacks_, 109
numberTrials_, 107
numberU_, 109
operator=, 92
permute, 93
permute_, 107
permuteBack, 94, 103
permuteBack_, 107
persistenceFlag, 98
persistenceFlag_, 114
pivot, 105
pivotColumn, 93
pivotColumn_, 107
pivotColumnBack, 94
pivotColumnBack_, 107
pivotColumnSingleton, 101
pivotOneOtherRow, 101
pivotRegion, 93
pivotRegion_, 109
pivotRowL_, 109
pivotRowSingleton, 101
pivotTolerance, 96
pivotTolerance_, 105
pivots, 93
preProcess, 101
relaxAccuracyCheck, 95
relaxCheck_, 105
reorderU, 101
replaceColumn, 98
replaceColumnPFI, 105
replaceColumnU, 99
replaceRow, 100
resetStatistics, 101
restoreFactorization, 92
saveColumn_, 108
saveFactorization, 92
separateLinks, 102
setBiasLU, 98
setCollectStatistics, 100
setDenseThreshold, 96
setForrestTomlin, 97
setNumberElementsU, 97
setNumberRows, 94
setPersistenceFlag, 98
setPivots, 93
setStatus, 93
show_self, 92
slackValue, 96
slackValue_, 105
sort, 92
spaceForForrestTomlin, 97
sparse_, 114
sparseThreshold, 99
sparseThreshold2_, 113
sparseThreshold_, 113
startColumnL, 94
startColumnL_, 111
startColumnR_, 111
startColumnU, 98
startColumnU_, 110
startRowL, 94

- startRowL_, 114
- startRowU_, 107
- status, 93
- status_, 107
- totalElements_, 106
- updateColumn, 99
- updateColumnFT, 99
- updateColumnL, 102
- updateColumnLDensish, 102
- updateColumnLSparse, 102
- updateColumnLSparsish, 103
- updateColumnPFI, 103
- updateColumnR, 103
- updateColumnRFT, 103
- updateColumnTranspose, 99
- updateColumnTransposeL, 104
- updateColumnTransposeLByRow, 104
- updateColumnTransposeLDensish, 104
- updateColumnTransposeLSparse, 105
- updateColumnTransposeLSparsish, 104
- updateColumnTransposePFI, 103
- updateColumnTransposeR, 104
- updateColumnTransposeRDensish, 104
- updateColumnTransposeRSparse, 104
- updateColumnTransposeU, 103
- updateColumnTransposeUByColumn, 104
- updateColumnTransposeUDensish, 104
- updateColumnTransposeUSparse, 104
- updateColumnTransposeUSparsish, 104
- updateColumnU, 103
- updateColumnUDensish, 103
- updateColumnUSparse, 103
- updateColumnUSparsish, 103
- updateTwoColumnsFT, 99
- updateTwoColumnsUDensish, 103
- workArea2_, 112
- workArea_, 112
- zeroTolerance, 96
- zeroTolerance_, 105
- CoinFactorizationDouble
 - CoinTypes.hpp, 552
- CoinFactorizationDoubleArrayWithLength, 114
 - array, 116
 - CoinFactorizationDoubleArrayWithLength, 115, 116
 - CoinFactorizationDoubleArrayWithLength, 115, 116
 - conditionalNew, 116
 - getSize, 116
 - operator=, 116
 - setSize, 116
- CoinFactorizationLongDoubleArrayWithLength, 117
 - array, 118
 - CoinFactorizationLongDoubleArrayWithLength, 118
 - CoinFactorizationLongDoubleArrayWithLength, 118
 - conditionalNew, 118
 - getSize, 118
 - operator=, 118
 - setSize, 118
- CoinFactorizationUnitTest
 - CoinFactorization, 105
- CoinFileIOBase, 122
 - ~CoinFileIOBase, 122
 - CoinFileIOBase, 122
 - CoinFileIOBase, 122
 - getFileName, 123
 - getReadType, 123
 - readType_, 123
- CoinFileInput, 119
 - ~CoinFileInput, 120
 - CoinFileInput, 120
 - CoinFileInput, 120
 - create, 120
 - fileAbsPath, 121
 - fileCoinReadable, 121
 - gets, 121
 - haveBzip2Support, 120
 - haveGzipSupport, 120
 - read, 120
- CoinFileOutput, 123
 - ~CoinFileOutput, 124
 - CoinFileOutput, 124
 - CoinFileOutput, 124
 - Compression, 124
 - compressionSupported, 124
 - create, 124
 - puts, 125
 - write, 125
- CoinFill
 - CoinHelperFunctions.hpp, 510
- CoinFillN
 - CoinHelperFunctions.hpp, 510
- CoinFindDirSeparator
 - CoinHelperFunctions.hpp, 512
- CoinFinite
 - CoinFinite.hpp, 504
- CoinFinite.hpp
 - COIN_DBL_MAX, 505
 - COIN_DBL_MIN, 505
 - COIN_INT_MAX, 505
 - CoinFinite, 504
 - CoinIsnan, 504
- CoinFirstAbsGreater_2
 - operator(), 126
- CoinFirstAbsGreater_2< S, T >, 125
- CoinFirstAbsGreater_3
 - operator(), 126
- CoinFirstAbsGreater_3< S, T, U >, 126
- CoinFirstAbsLess_2
 - operator(), 127

- CoinFirstAbsLess_2< S, T >, [127](#)
- CoinFirstAbsLess_3
 - operator(), [128](#)
- CoinFirstAbsLess_3< S, T, U >, [127](#)
- CoinFirstGreater_2
 - operator(), [128](#)
- CoinFirstGreater_2< S, T >, [128](#)
- CoinFirstGreater_3
 - operator(), [129](#)
- CoinFirstGreater_3< S, T, U >, [129](#)
- CoinFirstLess_2
 - operator(), [130](#)
- CoinFirstLess_2< S, T >, [129](#)
- CoinFirstLess_3
 - operator(), [130](#)
- CoinFirstLess_3< S, T, U >, [130](#)
- CoinFromFile
 - CoinHelperFunctions.hpp, [512](#)
- CoinGetTimeOfDay
 - CoinTime.hpp, [551](#)
- CoinHelperFunctions.hpp
 - COIN_RESTRICT, [508](#)
 - CoinAbs, [511](#)
 - CoinCheckDoubleZero, [510](#)
 - CoinCheckIntZero, [510](#)
 - CoinCopy, [508](#)
 - CoinCopyN, [508](#)
 - CoinCopyOfArray, [509](#)
 - CoinCopyOfArrayOrZero, [509](#)
 - CoinCopyOfArrayPartial, [509](#)
 - CoinDeleteEntriesFromArray, [511](#)
 - CoinDisjointCopy, [509](#)
 - CoinDisjointCopyN, [509](#)
 - CoinDrand48, [512](#)
 - CoinFill, [510](#)
 - CoinFillN, [510](#)
 - CoinFindDirSeparator, [512](#)
 - CoinFromFile, [512](#)
 - CoinIota, [511](#)
 - CoinIotaN, [511](#)
 - CoinIsSorted, [511](#)
 - CoinMax, [511](#)
 - CoinMemcpy, [510](#)
 - CoinMemcpyN, [509](#)
 - CoinMin, [511](#)
 - CoinSeedRandom, [512](#)
 - CoinSizeofAsInt, [508](#)
 - CoinStrNCaseCmp, [512](#)
 - CoinStrdup, [510](#)
 - CoinStrlenAsInt, [512](#)
 - CoinSwap, [512](#)
 - CoinToFile, [512](#)
 - CoinZero, [510](#)
 - CoinZeroN, [510](#)
- CoinIncrSolutionOrdered
 - CoinSort.hpp, [549](#)
- CoinIndexedVector, [132](#)
 - ~CoinIndexedVector, [138](#)
 - add, [140](#)
 - append, [142](#)
 - borrowVector, [139](#)
 - capacity, [144](#)
 - capacity_, [145](#)
 - checkClean, [141](#)
 - checkClear, [141](#)
 - clean, [140](#)
 - cleanAndPack, [141](#)
 - cleanAndPackSafe, [141](#)
 - clear, [139](#)
 - CoinIndexedVector, [137](#), [138](#)
 - CoinIndexedVectorUnitTest, [144](#)
 - CoinIndexedVector, [137](#), [138](#)
 - copy, [139](#)
 - createOneUnpackedElement, [142](#)
 - createPacked, [141](#)
 - createUnpacked, [142](#)
 - denseVector, [138](#)
 - elements_, [145](#)
 - empty, [139](#)
 - expand, [142](#)
 - getIndices, [138](#)
 - getMaxIndex, [143](#)
 - getMinIndex, [143](#)
 - getNumElements, [138](#)
 - indices_, [145](#)
 - insert, [140](#)
 - isApproximatelyEqual, [143](#)
 - nElements_, [145](#)
 - offset_, [145](#)
 - operator*, [143](#)
 - operator*=, [142](#), [144](#)
 - operator+, [143](#)
 - operator+=, [142](#), [144](#)
 - operator-, [143](#)
 - operator-=, [142](#), [144](#)
 - operator/, [144](#)
 - operator/=: [142](#), [144](#)
 - operator=, [139](#)
 - operator==, [142](#), [143](#)
 - packedMode, [144](#)
 - packedMode_, [145](#)
 - print, [142](#)
 - quickAdd, [140](#)
 - quickAddNonZero, [140](#)
 - quickInsert, [140](#)
 - reserve, [144](#)
 - returnVector, [139](#)
 - scan, [141](#)

- scanAndPack, [141](#)
- setConstant, [140](#)
- setDenseVector, [138](#)
- setElement, [140](#)
- setFull, [140](#)
- setIndexVector, [139](#)
- setNumElements, [139](#)
- setPacked, [141](#)
- setPackedMode, [144](#)
- setVector, [139](#), [140](#)
- sort, [143](#)
- sortDecrElement, [143](#)
- sortDecrIndex, [143](#)
- sortIncrElement, [143](#)
- sortIncrIndex, [143](#)
- sortPacked, [143](#)
- swap, [142](#)
- truncate, [142](#)
- zero, [140](#)
- CoinIndexedVector.hpp
 - COIN_PARTITIONS, [514](#)
 - CoinIndexedVectorUnitTest, [514](#)
- CoinIndexedVectorUnitTest
 - CoinIndexedVector, [144](#)
 - CoinIndexedVector.hpp, [514](#)
- CoinInt64
 - CoinTypes.hpp, [551](#)
- CoinIntArrayWithLength, [145](#)
 - array, [147](#)
 - CoinIntArrayWithLength, [146](#), [147](#)
 - CoinIntArrayWithLength, [146](#), [147](#)
 - conditionalNew, [147](#)
 - getSize, [147](#)
 - operator=, [147](#)
 - setSize, [147](#)
- CoinIntPtr
 - CoinTypes.hpp, [552](#)
- Coinlota
 - CoinHelperFunctions.hpp, [511](#)
- CoinlotaN
 - CoinHelperFunctions.hpp, [511](#)
- CoinIsSorted
 - CoinHelperFunctions.hpp, [511](#)
- CoinIsnan
 - CoinFinite.hpp, [504](#)
- CoinLpIO, [147](#)
 - ~CoinLpIO, [155](#)
 - are_invalid_names, [159](#)
 - card_previous_names_, [167](#)
 - checkColNames, [164](#)
 - checkRowNames, [163](#)
 - CoinLpIO, [155](#)
 - CoinLpIOUnitTest, [164](#)
 - CoinLpIO, [155](#)
 - collower_, [165](#)
 - columnIndex, [157](#)
 - columnName, [157](#)
 - colupper_, [165](#)
 - convertBoundToSense, [155](#)
 - decimals_, [166](#)
 - defaultHandler_, [164](#)
 - epsilon_, [166](#)
 - fileName_, [166](#)
 - find_obj, [162](#)
 - findHash, [162](#)
 - first_is_number, [162](#)
 - freeAll, [155](#)
 - freePreviousNames, [155](#)
 - getColLower, [156](#)
 - getColNames, [157](#)
 - getColUpper, [156](#)
 - getDecimals, [158](#)
 - getEpsilon, [158](#)
 - getInfinity, [158](#)
 - getMatrixByCol, [157](#)
 - getMatrixByRow, [157](#)
 - getNumCols, [155](#)
 - getNumElements, [156](#)
 - getNumRows, [155](#)
 - getNumberAcross, [158](#)
 - getObjCoefficients, [157](#)
 - getObjName, [157](#)
 - getPreviousColNames, [157](#)
 - getPreviousRowNames, [157](#)
 - getProblemName, [155](#)
 - getRightHandSide, [156](#)
 - getRowLower, [156](#)
 - getRowNames, [157](#)
 - getRowRange, [156](#)
 - getRowSense, [156](#)
 - getRowUpper, [156](#)
 - gutsOfCopy, [155](#)
 - gutsOfDestructor, [155](#)
 - handler_, [164](#)
 - hash_, [167](#)
 - infinity_, [166](#)
 - insertHash, [162](#)
 - integerColumns, [158](#)
 - integerType_, [166](#)
 - is_comment, [162](#)
 - is_free, [162](#)
 - is_inf, [163](#)
 - is_invalid_name, [159](#)
 - is_keyword, [163](#)
 - is_sense, [163](#)
 - is_subject_to, [162](#)
 - isInteger, [158](#)
 - matrixByColumn_, [165](#)

- matrixByRow_, 165
- maxHash_, 167
- messageHandler, 161
- messages, 161
- messages_, 164
- messagesPointer, 161
- names_, 167
- newLanguage, 161
- numberAcross_, 166
- numberColumns_, 164
- numberElements_, 165
- numberHash_, 167
- numberRows_, 164
- objName_, 166
- objective_, 166
- objectiveOffset, 158
- objectiveOffset_, 166
- operator=, 155
- out_coeff, 162
- passInMessageHandler, 161
- previous_names_, 166
- print, 161
- problemName_, 164
- read_monom_obj, 163
- read_monom_row, 163
- read_row, 163
- readLp, 160, 161
- realloc_coeff, 163
- realloc_col, 163
- realloc_row, 163
- rhs_, 165
- rowIndex, 157
- rowName, 157
- rowlower_, 165
- rowrange_, 165
- rowsense_, 165
- rowupper_, 165
- scan_next, 162
- setDecimals, 158
- setDefaultColNames, 159
- setDefaultRowNames, 159
- setEpsilon, 158
- setInfinity, 158
- setLanguage, 161
- setLpDataRowAndColNames, 159
- setLpDataWithoutRowAndColNames, 159
- setNumberAcross, 158
- setObjectiveOffset, 158
- setProblemName, 155
- skip_comment, 162
- startHash, 161
- stopHash, 161
- writeLp, 160
- CoinLpIO.hpp
 - COINColumnIndex, 514
 - CoinLpIOUnitTest, 515
 - CoinLpIO::CoinHashLink, 131
 - index, 131
 - next, 131
 - CoinLpIOUnitTest
 - CoinLpIO, 164
 - CoinLpIO.hpp, 515
 - CoinMakePair
 - CoinUtility.hpp, 552
 - CoinMakeTriple
 - CoinUtility.hpp, 552
 - CoinMax
 - CoinHelperFunctions.hpp, 511
 - CoinMemcpy
 - CoinHelperFunctions.hpp, 510
 - CoinMemcpyN
 - CoinHelperFunctions.hpp, 509
 - CoinMessage, 167
 - CoinMessage, 168
 - CoinMessage, 168
 - CoinMessage.hpp
 - COIN_Message, 515
 - CoinMessageHandler, 168
 - ~CoinMessageHandler, 173
 - charValue, 175
 - charValue_, 177
 - checkSeverity, 173
 - clone, 173
 - CoinMessageHandler, 173
 - CoinMessageHandlerUnitTest, 177
 - CoinMessageHandler, 173
 - currentMessage, 175
 - currentMessage_, 178
 - currentSource, 175
 - detail, 173
 - doubleValue, 174
 - doubleValue_, 177
 - filePointer, 176
 - finish, 177
 - format_, 178
 - fp_, 179
 - g_format_, 179
 - g_precision_, 179
 - highestNumber, 175
 - highestNumber_, 179
 - intValue, 174
 - internalNumber_, 178
 - logLevel, 173, 174
 - logLevel_, 178
 - logLevels_, 178
 - longValue_, 177
 - message, 176
 - messageBuffer, 175

- messageBuffer_, 178
- messageOut_, 178
- numberCharFields, 175
- numberDoubleFields, 174
- numberIntFields, 175
- numberStringFields, 175
- operator<<, 176, 177
- operator=, 173
- precision, 174
- prefix, 174
- prefix_, 178
- print, 173
- printStatus_, 178
- printing, 177
- setFilePointer, 176
- setLogLevel, 173, 174
- setPrecision, 174
- setPrefix, 174
- source_, 178
- stringValue, 175
- stringValue_, 177
- CoinMessageHandler.hpp
 - COIN_NUM_LOG, 517
 - CoinMessageHandlerUnitTest, 518
 - CoinMessageMarker, 518
- CoinMessageHandlerUnitTest
 - CoinMessageHandler, 177
 - CoinMessageHandler.hpp, 518
- CoinMessageMarker
 - CoinMessageHandler.hpp, 518
- CoinMessages, 179
 - ~CoinMessages, 181
 - addMessage, 181
 - class_, 182
 - CoinMessages, 181
 - CoinMessages, 181
 - fromCompact, 182
 - getClass, 182
 - Language, 181
 - language, 181
 - language_, 182
 - lengthMessages_, 182
 - message_, 183
 - numberMessages_, 182
 - operator=, 181
 - replaceMessage, 181
 - setDetailMessage, 182
 - setDetailMessages, 182
 - setLanguage, 181
 - source_, 182
 - toCompact, 182
- CoinMin
 - CoinHelperFunctions.hpp, 511
- CoinModel, 183
 - ~CoinModel, 191
 - addCol, 191
 - addColumn, 191
 - addRow, 191
 - associateElement, 192
 - associatedArray, 203
 - clone, 206
 - CoinModel, 191
 - CoinModel, 191
 - column, 202
 - columnIsInteger, 200
 - columnIsIntegerAsString, 202
 - columnLower, 200
 - columnLowerArray, 203
 - columnLowerAsString, 202
 - columnName, 200
 - columnNames, 204
 - columnObjective, 200
 - columnObjectiveAsString, 202
 - columnUpper, 200
 - columnUpperArray, 203
 - columnUpperAsString, 202
 - computeAssociated, 206
 - convertMatrix, 197
 - countPlusMinusOne, 203
 - createArrays, 203
 - createPackedMatrix, 203
 - createPlusMinusOne, 203
 - cutMarker, 204
 - deleteCol, 195
 - deleteColumn, 195
 - deleteElement, 195
 - deleteRow, 195
 - deleteThisElement, 195
 - differentModel, 197
 - elements, 198
 - expandKnapsack, 206
 - firstInColumn, 198
 - firstInQuadraticColumn, 199
 - firstInRow, 198
 - getColIsInteger, 201
 - getColLower, 201
 - getColName, 201
 - getColObjective, 201
 - getColUpper, 201
 - getColumn, 192
 - getColumnIsInteger, 200
 - getColumnIsIntegerAsString, 202
 - getColumnLower, 200
 - getColumnLowerAsString, 201
 - getColumnName, 200
 - getColumnObjective, 200
 - getColumnObjectiveAsString, 201
 - getColumnUpper, 200

getColumnUpperAsString, 201
getElement, 198
getElementAsString, 198
getQuadraticElement, 198
getRow, 192
getRowLower, 199
getRowLowerAsString, 201
getRowName, 199
getRowUpper, 199
getRowUpperAsString, 201
integerTypeArray, 204
isInteger, 200
isIntegerAsString, 202
lastInColumn, 199
lastInQuadraticColumn, 199
lastInRow, 198
loadBlock, 205, 206
moreInfo, 204
next, 199
numberElements, 197
objective, 200
objectiveArray, 204
objectiveAsString, 202
operator(), 192, 198
operator=, 206
optimizationDirection, 204
originalColumns, 197
originalRows, 197
pack, 196
packCols, 195
packColumns, 195
packRows, 195
packedMatrix, 197
passInMatrix, 197
pointer, 198
position, 198
previous, 199
priorities, 207
quadraticRow, 206
reorder, 206
replaceQuadraticRow, 206
row, 202
rowLower, 199
rowLowerArray, 203
rowLowerAsString, 201
rowName, 199
rowNames, 204
rowUpper, 199
rowUpperArray, 203
rowUpperAsString, 201
setColBounds, 194
setColIsInteger, 194
setColLower, 193, 196
setColName, 194
setColObjective, 194
setColUpper, 193, 196
setColumnBounds, 193
setColumnIsInteger, 193, 194
setColumnLower, 193, 194, 196
setColumnName, 193
setColumnObjective, 193, 194
setColumnUpper, 193, 194, 196
setContinuous, 193
setCutMarker, 207
setElement, 192
setInteger, 193
setIsInteger, 193, 195
setMoreInfo, 205
setObjective, 193, 194, 196
setOptimizationDirection, 204
setOriginalIndices, 207
setPriorities, 207
setQuadraticElement, 192
setRowBounds, 192
setRowLower, 192, 194, 196
setRowName, 192
setRowUpper, 192, 194, 196
stringArray, 203
stringsExist, 203
type, 202
unsetValue, 202
validateLinks, 206
whatIsSet, 205
writeMps, 196
zapColumnNames, 204
zapRowNames, 204
CoinModel.hpp
 getDoubleFromString, 519
 getFunctionValueFromString, 519
coinModelBlock
 CoinStructuredModel, 409
CoinModelBlockInfo
 CoinStructuredModel.hpp, 550
CoinModelHash, 207
 ~CoinModelHash, 208
 addHash, 209
 CoinModelHash, 208
 CoinModelHash, 208
 deleteHash, 209
 getName, 209
 hash, 209
 maximumItems, 209
 name, 209
 names, 209
 numberOfItems, 209
 operator=, 208
 resize, 208
 setName, 209

- setNumberItems, 209
 - validateHash, 209
- CoinModelHash2, 210
 - ~CoinModelHash2, 210
 - addHash, 211
 - CoinModelHash2, 210, 211
 - CoinModelHash2, 210, 211
 - deleteHash, 211
 - hash, 211
 - maximumItems, 211
 - numberItems, 211
 - operator=, 211
 - resize, 211
 - setNumberItems, 211
- CoinModelHashLink, 211
 - index, 212
 - next, 212
- CoinModelInfo2, 212
 - bounds, 213
 - CoinModelInfo2, 213
 - CoinModelInfo2, 213
 - columnBlock, 213
 - columnName, 213
 - integer, 213
 - matrix, 213
 - rhs, 213
 - rowBlock, 213
 - rowName, 213
- CoinModelLink, 213
 - ~CoinModelLink, 214
 - CoinModelLink, 214
 - CoinModelLink, 214
 - column, 215
 - element, 215
 - onRow, 215
 - operator=, 215
 - position, 215
 - row, 215
 - setColumn, 215
 - setElement, 216
 - setOnRow, 216
 - setPosition, 216
 - setRow, 215
 - setValue, 215
 - value, 215
- CoinModelLinkedList, 216
 - ~CoinModelLinkedList, 217
 - addEasy, 219
 - addHard, 219
 - CoinModelLinkedList, 217, 218
 - CoinModelLinkedList, 217, 218
 - create, 218
 - deleteRowOne, 219
 - deleteSame, 219
 - fill, 220
 - first, 218
 - firstFree, 218
 - last, 219
 - lastFree, 218
 - maximumElements, 218
 - maximumMajor, 218
 - next, 219
 - numberElements, 218
 - numberMajor, 218
 - operator=, 218
 - previous, 219
 - resize, 218
 - synchronize, 220
 - updateDeleted, 219
 - updateDeletedOne, 220
 - validateLinks, 220
- CoinModelTriple, 220
 - column, 220
 - row, 220
 - value, 220
- CoinModelUseful.hpp
 - func_t, 520
 - rowInTriple, 520
 - setRowAndStringInTriple, 520
 - setRowInTriple, 520
 - setStringInTriple, 520
 - stringInTriple, 520
 - symrec, 520
- CoinMpsCardReader, 221
 - ~CoinMpsCardReader, 223
 - card, 224
 - card_, 225
 - cardNumber, 225
 - cardNumber_, 226
 - cleanCard, 224
 - CoinMpsCardReader, 223
 - CoinMpsCardReader, 223
 - columnName, 224
 - columnName_, 226
 - eightChar_, 226
 - eol_, 226
 - fileInput, 225
 - freeFormat, 223
 - freeFormat_, 226
 - getPosition, 225
 - handler_, 227
 - ieeeFormat_, 226
 - input_, 226
 - messages_, 227
 - mpsType, 224
 - mpsType_, 226
 - mutableCard, 224
 - nextBlankOr, 225

- nextField, [223](#)
- nextGmsField, [223](#)
- osi_strtod, [225](#)
- position_, [225](#)
- readToNextSection, [223](#)
- reader_, [227](#)
- rowName, [224](#)
- rowName_, [226](#)
- section_, [226](#)
- setFreeFormat, [224](#)
- setPosition, [224](#)
- setStringsAllowed, [225](#)
- setWhichSection, [223](#)
- strcpyAndCompress, [225](#)
- stringsAllowed_, [227](#)
- value, [224](#)
- value_, [225](#)
- valueString, [224](#)
- valueString_, [227](#)
- whichSection, [223](#)
- CoinMpsIO, [227](#)
 - ~CoinMpsIO, [234](#)
 - addString, [243](#)
 - allowStringElements, [238](#)
 - allowStringElements_, [246](#)
 - boundName_, [244](#)
 - cardReader_, [246](#)
 - CoinMpsIO, [234](#)
 - CoinMpsIOUnitTest, [243](#)
 - CoinMpsIO, [234](#)
 - collower_, [245](#)
 - columnIndex, [236](#)
 - columnName, [236](#)
 - colupper_, [245](#)
 - convertBoundToSense, [242](#)
 - convertObjective_, [246](#)
 - convertSenseToBound, [242](#)
 - copyInIntegerInformation, [237](#)
 - copyStringElements, [241](#)
 - dealWithFileName, [242](#)
 - decodeString, [243](#)
 - defaultBound_, [246](#)
 - defaultHandler_, [246](#)
 - fileName_, [245](#)
 - findHash, [243](#)
 - freeAll, [242](#)
 - getBoundName, [237](#)
 - getColLower, [234](#)
 - getColUpper, [234](#)
 - getDefaultBound, [238](#)
 - getFileName, [238](#)
 - getInfinity, [237](#)
 - getMatrixByCol, [235](#)
 - getMatrixByRow, [235](#)
 - getNumCols, [234](#)
 - getNumElements, [234](#)
 - getNumRows, [234](#)
 - getObjCoefficients, [235](#)
 - getObjectiveName, [236](#)
 - getProblemName, [236](#)
 - getRangeName, [236](#)
 - getRhsName, [236](#)
 - getRightHandSide, [235](#)
 - getRowLower, [235](#)
 - getRowRange, [235](#)
 - getRowSense, [234](#)
 - getRowUpper, [235](#)
 - getSmallElementValue, [238](#)
 - gutsOfCopy, [242](#)
 - gutsOfDestructor, [242](#)
 - handler_, [246](#)
 - hash_, [245](#)
 - infinity_, [246](#)
 - integerColumns, [236](#)
 - integerType_, [245](#)
 - isContinuous, [235](#)
 - isInteger, [235](#)
 - matrixByColumn_, [244](#)
 - matrixByRow_, [244](#)
 - maximumStringElements_, [246](#)
 - messageHandler, [241](#)
 - messages, [241](#)
 - messages_, [246](#)
 - messagesPointer, [241](#)
 - names_, [245](#)
 - newLanguage, [241](#)
 - numberColumns_, [244](#)
 - numberElements_, [244](#)
 - numberHash_, [245](#)
 - numberRows_, [244](#)
 - numberStringElements, [237](#)
 - numberStringElements_, [247](#)
 - objective_, [245](#)
 - objectiveName_, [243](#)
 - objectiveOffset, [236](#)
 - objectiveOffset_, [245](#)
 - operator=, [241](#)
 - passInMessageHandler, [241](#)
 - problemName_, [243](#)
 - rangeName_, [243](#)
 - readBasis, [239](#)
 - readConicMps, [240](#)
 - readGMPL, [239](#)
 - readGms, [239](#)
 - readMps, [238](#)
 - readQuadraticMps, [240](#)
 - reader, [240](#)
 - releaseColumnInformation, [241](#)

- releaseColumnNames, [242](#)
- releaseIntegerInformation, [241](#)
- releaseMatrixInformation, [242](#)
- releaseRedundantInformation, [241](#)
- releaseRowInformation, [241](#)
- releaseRowNames, [242](#)
- rhs_, [244](#)
- rhsName_, [243](#)
- rowIndex, [236](#)
- rowName, [236](#)
- rowlower_, [244](#)
- rowrange_, [244](#)
- rowsense_, [244](#)
- rowupper_, [245](#)
- setAllowStringElements, [238](#)
- setConvertObjective, [240](#)
- setDefaultBound, [237](#)
- setFileName, [238](#)
- setInfinity, [237](#)
- setLanguage, [241](#)
- setMpsData, [237](#)
- setMpsDataColAndRowNames, [242](#)
- setMpsDataWithoutRowAndColNames, [242](#)
- setObjectiveName, [237](#)
- setObjectiveOffset, [236](#)
- setProblemName, [237](#)
- setSmallElementValue, [238](#)
- smallElement_, [246](#)
- startHash, [243](#)
- stopHash, [243](#)
- stringElement, [237](#)
- stringElements_, [247](#)
- writeMps, [239](#)
- CoinMpsIO.hpp
 - COINColumnIndex, [521](#)
 - COINMpsType, [522](#)
 - COINRowIndex, [522](#)
 - COINSectionType, [522](#)
 - CoinConvertDouble, [523](#)
 - CoinMpsIOUnitTest, [523](#)
 - MAX_CARD_LENGTH, [521](#)
- CoinMpsIO::CoinHashLink, [131](#)
 - index, [132](#)
 - next, [132](#)
- CoinMpsIOUnitTest
 - CoinMpsIO, [243](#)
 - CoinMpsIO.hpp, [523](#)
- CoinNodeAction
 - CoinSearchTree.hpp, [544](#)
- CoinOneMessage, [247](#)
 - ~CoinOneMessage, [248](#)
 - CoinOneMessage, [248](#)
 - CoinOneMessage, [248](#)
 - detail, [249](#)
 - detail_, [249](#)
 - externalNumber, [249](#)
 - externalNumber_, [249](#)
 - message, [249](#)
 - message_, [250](#)
 - operator=, [248](#)
 - replaceMessage, [248](#)
 - setDetail, [249](#)
 - setExternalNumber, [249](#)
 - severity, [249](#)
 - severity_, [249](#)
- CoinOslC.h
 - C_EKK_ADD_LINK, [525](#)
 - C_EKK_GO_SPARSE, [525](#)
 - c_ekk_IsSet, [527](#)
 - c_ekk_Set, [527](#)
 - c_ekk_Unset, [527](#)
 - c_ekkbtrn, [526](#)
 - c_ekkbtrn_ipivrw, [526](#)
 - c_ekkczero, [527](#)
 - c_ekkdcpy, [526](#)
 - c_ekkdzero, [527](#)
 - c_ekketsj, [526](#)
 - c_ekkfrtn, [526](#)
 - c_ekkfrtn2, [526](#)
 - c_ekkfrtn_ft, [526](#)
 - c_ekkizero, [527](#)
 - c_ekklfct, [526](#)
 - c_ekks1cpy, [525](#)
 - c_ekkscpy, [526](#)
 - c_ekkscpy_0_1, [525](#)
 - c_ekkslcf, [526](#)
 - c_ekkzero, [527](#)
 - CLP_OSL, [525](#)
 - clp_double, [527](#)
 - clp_free, [527](#)
 - clp_int, [527](#)
 - clp_malloc, [527](#)
 - clp_memory, [527](#)
 - clp_setup_pointers, [527](#)
 - NO_SHIFT, [525](#)
 - NOT_ZERO, [526](#)
 - SHIFT_INDEX, [526](#)
 - SHIFT_REF, [526](#)
 - SLACK_VALUE, [525](#)
 - SPARSE_UPDATE, [525](#)
 - SWAP, [526](#)
 - UNROLL_LOOP_BODY1, [526](#)
 - UNROLL_LOOP_BODY2, [526](#)
 - UNROLL_LOOP_BODY4, [526](#)
 - UNSHIFT_INDEX, [526](#)
- CoinOslFactorization, [250](#)
 - ~CoinOslFactorization, [252](#)
 - checkPivot, [256](#)

- clearArrays, 255
- clone, 253
- CoinOslFactorization, 252
- CoinOslFactorizationUnitTest, 256
- CoinOslFactorization, 252
- conditionNumber, 255
- elements, 254
- factInfo_, 256
- factor, 253
- factorize, 253
- getAreas, 253
- gutsOfCopy, 256
- gutsOfDestructor, 256
- gutsOfInitialize, 256
- indices, 256
- intWorkArea, 254
- makeNonSingular, 253
- maximumCoefficient, 255
- maximumPivots, 255
- numberElements, 253
- numberInColumn, 254
- numberInRow, 254
- operator=, 253
- permute, 256
- permuteBack, 254
- pivotRow, 254
- postProcess, 253
- preProcess, 253
- replaceColumn, 255
- setUsefulInformation, 254
- starts, 254
- updateColumn, 255
- updateColumnFT, 255
- updateColumnTranspose, 255
- updateTwoColumnsFT, 255
- wantsTableauColumn, 254
- workArea, 254
- CoinOslFactorization.hpp
 - EKKfactinfo, 528
- CoinOslFactorizationUnitTest
 - CoinOslFactorization, 256
- CoinOtherFactorization, 257
 - ~CoinOtherFactorization, 260
 - clearArrays, 263
 - clone, 260
 - CoinOtherFactorization, 260
 - CoinOtherFactorization, 260
 - elements, 262
 - elements_, 266
 - factor, 264
 - factorElements_, 265
 - getAccuracyCheck, 261
 - getAreas, 264
 - indices, 263
 - intWorkArea, 262
 - makeNonSingular, 264
 - maximumPivots, 261
 - maximumPivots_, 266
 - maximumRows_, 266
 - maximumSpace_, 266
 - numberColumns, 261
 - numberColumns_, 265
 - numberElements, 263
 - numberGoodColumns, 261
 - numberGoodU_, 265
 - numberInColumn, 262
 - numberInRow, 262
 - numberPivots_, 266
 - numberRows, 261
 - numberRows_, 265
 - operator=, 260
 - permute, 263
 - permuteBack, 263
 - pivotRow, 262
 - pivotRow_, 266
 - pivotTolerance, 262
 - pivotTolerance_, 265
 - pivots, 261
 - postProcess, 264
 - preProcess, 264
 - relaxAccuracyCheck, 261
 - relaxCheck_, 265
 - replaceColumn, 264
 - setNumberRows, 261
 - setPivots, 261
 - setSolveMode, 263
 - setStatus, 260
 - setUsefulInformation, 263
 - slackValue, 262
 - slackValue_, 265
 - solveMode, 263
 - solveMode_, 266
 - starts, 262
 - status, 260
 - status_, 266
 - updateColumn, 264
 - updateColumnFT, 264
 - updateColumnTranspose, 265
 - updateTwoColumnsFT, 264
 - wantsTableauColumn, 263
 - workArea, 262
 - workArea_, 266
 - zeroTolerance, 262
 - zeroTolerance_, 265
- CoinPackedMatrix, 267
 - ~CoinPackedMatrix, 275
 - appendCol, 277
 - appendCols, 277, 278

appendMajor, [286](#)
 appendMajorVector, [282](#)
 appendMajorVectors, [282](#)
 appendMinor, [287](#)
 appendMinorFast, [283](#)
 appendMinorVector, [282](#), [283](#)
 appendMinorVectors, [283](#)
 appendRow, [278](#)
 appendRows, [278](#)
 assignMatrix, [280](#)
 bottomAppendPackedMatrix, [278](#)
 cleanMatrix, [279](#)
 clear, [275](#)
 CoinPackedMatrix, [274](#)
 CoinPackedMatrixUnitTest, [287](#)
 CoinPackedMatrix, [274](#)
 colOrdered_, [287](#)
 compress, [279](#)
 copyOf, [280](#)
 copyReuseArrays, [280](#)
 countOrthoLength, [281](#)
 deleteCols, [278](#)
 deleteMajorVectors, [284](#)
 deleteMinorVectors, [284](#)
 deleteRows, [278](#)
 dumpMatrix, [282](#)
 element_, [287](#)
 eliminateDuplicates, [279](#)
 extraGap_, [287](#)
 extraMajor_, [287](#)
 getCoefficient, [279](#)
 getElements, [276](#)
 getExtraGap, [275](#)
 getExtraMajor, [275](#)
 getIndices, [276](#)
 getMajorDim, [281](#)
 getMajorIndices, [277](#)
 getMaxMajorDim, [282](#)
 getMinorDim, [282](#)
 getMutableElements, [285](#)
 getMutableIndices, [285](#)
 getMutableVectorLengths, [285](#)
 getMutableVectorStarts, [285](#)
 getNumCols, [275](#)
 getNumElements, [275](#)
 getNumRows, [275](#)
 getSizeVectorLengths, [276](#)
 getSizeVectorStarts, [276](#)
 getVector, [277](#)
 getVectorFirst, [276](#)
 getVectorLast, [276](#)
 getVectorLengths, [276](#)
 getVectorSize, [277](#)
 getVectorStarts, [276](#)
 gutsOfCopyOf, [286](#)
 gutsOfCopyOfNoGaps, [286](#)
 gutsOfDestructor, [286](#)
 gutsOfOpEqual, [286](#)
 hasGaps, [275](#)
 index_, [287](#)
 isColOrdered, [275](#)
 isEquivalent, [284](#)
 isEquivalent2, [284](#)
 length_, [288](#)
 majorAppendOrthoOrdered, [283](#)
 majorAppendSameOrdered, [283](#)
 majorDim_, [288](#)
 maxMajorDim_, [288](#)
 maxSize_, [288](#)
 minorAppendOrthoOrdered, [283](#)
 minorAppendSameOrdered, [283](#)
 minorDim_, [288](#)
 modifyCoefficient, [279](#)
 nullElementArray, [285](#)
 nullIndexArray, [286](#)
 nullLengthArray, [285](#)
 nullStartArray, [285](#)
 operator=, [280](#)
 orderMatrix, [279](#)
 printMatrixElement, [282](#)
 removeGaps, [279](#)
 replaceVector, [279](#)
 reserve, [275](#)
 resizeForAddingMajorVectors, [286](#)
 resizeForAddingMinorVectors, [286](#)
 reverseOrderedCopyOf, [280](#)
 reverseOrdering, [280](#)
 rightAppendPackedMatrix, [278](#)
 setDimensions, [277](#)
 setExtraGap, [277](#)
 setExtraMajor, [277](#)
 setMajorDim, [282](#)
 setMinorDim, [282](#)
 setNumElements, [285](#)
 size_, [288](#)
 start_, [288](#)
 submatrixOf, [279](#)
 submatrixOfWithDuplicates, [279](#)
 swap, [281](#)
 times, [281](#)
 timesMajor, [284](#)
 timesMinor, [284](#)
 transpose, [281](#)
 transposeTimes, [281](#)
 verifyMtx, [286](#)
 CoinPackedMatrix.hpp
 CoinPackedMatrixUnitTest, [528](#)
 CoinPackedMatrixUnitTest

- CoinPackedMatrix, 287
- CoinPackedMatrix.hpp, 528
- CoinPackedVector, 288
 - ~CoinPackedVector, 293
 - append, 295
 - assignVector, 294
 - capacity, 296
 - clear, 294
 - CoinPackedVector, 292, 293
 - CoinPackedVectorUnitTest, 296
 - CoinPackedVector, 292, 293
 - getElements, 293, 294
 - getIndices, 293
 - getNumElements, 293
 - getOriginalPosition, 294
 - getVectorElements, 294
 - getVectorIndices, 294
 - getVectorNumElements, 294
 - insert, 295
 - operator*=, 295
 - operator+=, 295
 - operator-=, 295
 - operator/=: 295
 - operator=, 294
 - reserve, 296
 - setConstant, 295
 - setElement, 295
 - setFull, 295
 - setFullNonZero, 295
 - setVector, 294
 - sort, 296
 - sortDecrElement, 296
 - sortDecrIndex, 296
 - sortIncrElement, 296
 - sortIncrIndex, 296
 - sortOriginalOrder, 296
 - swap, 295
 - truncate, 295
- CoinPackedVector.hpp
 - binaryOp, 530
 - CoinPackedVectorUnitTest, 532
 - operator*, 531, 532
 - operator+, 530–532
 - operator-, 531, 532
 - operator/, 531, 532
 - sortedSparseDotProduct, 531
 - sparseDotProduct, 531
- CoinPackedVectorBase, 297
 - ~CoinPackedVectorBase, 299
 - clearBase, 302
 - clearIndexSet, 302
 - CoinPackedVectorBase, 299
 - CoinPackedVectorBase, 299
 - compare, 301
 - copyMaxMinIndex, 302
 - denseVector, 300
 - dotProduct, 301
 - duplicateIndex, 300
 - findIndex, 300
 - findMaxMinIndices, 301
 - getElements, 299
 - getIndices, 299
 - getMaxIndex, 300
 - getMinIndex, 300
 - getNumElements, 299
 - indexSet, 301
 - infNorm, 301
 - isEquivalent, 301
 - isExistingIndex, 300
 - normSquare, 301
 - oneNorm, 301
 - operator==, 300
 - setTestForDuplicateIndex, 299
 - setTestForDuplicateIndexWhenTrue, 299
 - setTestsOff, 300
 - sum, 301
 - testForDuplicateIndex, 300
 - twoNorm, 301
- CoinPackedVectorUnitTest
 - CoinPackedVector, 296
 - CoinPackedVector.hpp, 532
- CoinPair
 - CoinPair, 302
 - CoinPair, 302
 - first, 303
 - second, 303
- CoinPair< S, T >, 302
- CoinParam, 303
 - ~CoinParam, 308
 - appendKwd, 308
 - clone, 308
 - CoinParam, 307, 308
 - CoinParamFunc, 307
 - CoinParamType, 307
 - CoinParamVec, 311
 - CoinParam, 307, 308
 - dblVal, 309
 - display, 311
 - getCommand, 312
 - getDoubleField, 312
 - getIntField, 312
 - getStringField, 311
 - intVal, 309
 - isCommandLine, 311
 - isInteractive, 311
 - kwdIndex, 308
 - kwdVal, 308
 - longHelp, 310

- lookupParam, [312](#)
- matchName, [310](#)
- matchParam, [312](#)
- matches, [310](#)
- name, [310](#)
- operator<<, [311](#)
- operator=, [308](#)
- printGenericHelp, [313](#)
- printHelp, [314](#)
- printIt, [313](#)
- printKwds, [309](#)
- printLongHelp, [310](#)
- pullFunc, [311](#)
- pushFunc, [311](#)
- setDbIVal, [309](#)
- setDisplay, [310](#)
- setInputSrc, [311](#)
- setIntVal, [309](#)
- setKwdVal, [309](#)
- setLongHelp, [309](#)
- setName, [310](#)
- setPullFunc, [311](#)
- setPushFunc, [311](#)
- setShortHelp, [309](#)
- setStrVal, [309](#)
- setType, [310](#)
- shortHelp, [309](#)
- shortOrHelpMany, [313](#)
- shortOrHelpOne, [313](#)
- strVal, [309](#)
- type, [310](#)
- CoinParam.hpp
 - operator<<, [534](#)
- CoinParamFunc
 - CoinParam, [307](#)
- CoinParamType
 - CoinParam, [307](#)
- CoinParamUtils, [29](#)
 - getCommand, [31](#)
 - getDoubleField, [30](#)
 - getIntField, [30](#)
 - getStringField, [30](#)
 - isCommandLine, [30](#)
 - isInteractive, [30](#)
 - lookupParam, [31](#)
 - matchParam, [30](#)
 - printGenericHelp, [32](#)
 - printHelp, [32](#)
 - printIt, [32](#)
 - setInputSrc, [30](#)
 - shortOrHelpMany, [32](#)
 - shortOrHelpOne, [32](#)
- CoinParamVec
 - CoinParam, [311](#)
- CoinPartitionedVector, [314](#)
 - ~CoinPartitionedVector, [316](#)
 - checkClean, [317](#)
 - checkClear, [317](#)
 - clearAndKeep, [317](#)
 - clearAndReset, [317](#)
 - clearPartition, [317](#)
 - CoinPartitionedVector, [316](#)
 - CoinPartitionedVector, [316](#)
 - compact, [317](#)
 - computeNumberElements, [317](#)
 - getNumElements, [316](#)
 - getNumPartitions, [316](#)
 - numberElementsPartition_, [318](#)
 - numberPartitions_, [318](#)
 - operator=, [318](#)
 - print, [318](#)
 - reserve, [317](#)
 - scan, [317](#)
 - setNumElementsPartition, [317](#)
 - setPartitions, [317](#)
 - setTempNumElementsPartition, [317](#)
 - sort, [318](#)
 - startPartition, [316](#)
 - startPartition_, [318](#)
 - startPartitions, [317](#)
- CoinPostsolveMatrix, [318](#)
 - ~CoinPostsolveMatrix, [321](#)
 - assignPresolveToPostsolve, [321](#)
 - cdone_, [322](#)
 - check_nbasic, [321](#)
 - CoinPostsolveMatrix, [321](#)
 - CoinPostsolveMatrix, [321](#)
 - free_list_, [321](#)
 - link_, [321](#)
 - maxlink_, [321](#)
 - rdone_, [322](#)
- CoinPrePostsolveMatrix, [322](#)
 - ~CoinPrePostsolveMatrix, [329](#)
 - acts_, [336](#)
 - bulk0_, [334](#)
 - bulkRatio_, [334](#)
 - clo_, [335](#)
 - CoinPrePostsolveMatrix, [328](#), [329](#)
 - CoinPrePostsolveMatrix, [328](#), [329](#)
 - coles_, [334](#)
 - colstat_, [336](#)
 - columnIsBasic, [329](#)
 - columnStatusString, [330](#)
 - cost_, [334](#)
 - countEmptyCols, [332](#)
 - cup_, [335](#)
 - defaultHandler_, [336](#)
 - getColLengths, [331](#)

getColLower, [332](#)
getColSolution, [332](#)
getColStarts, [331](#)
getColUpper, [332](#)
getColumnStatus, [329](#)
getCost, [332](#)
getElementsByCol, [331](#)
getNumCols, [331](#)
getNumElems, [331](#)
getNumRows, [331](#)
getReducedCost, [332](#)
getRowActivity, [332](#)
getRowIndicesByCol, [331](#)
getRowLower, [332](#)
getRowPrice, [332](#)
getRowStatus, [329](#)
getRowUpper, [332](#)
getStatus, [330](#)
handler_, [336](#)
hincol_, [334](#)
hrow_, [334](#)
maxmin_, [335](#)
mcstrt_, [334](#)
messageHandler, [333](#)
messages, [333](#)
messages_, [336](#)
ncols0_, [333](#)
ncols_, [333](#)
nelems0_, [334](#)
nelems_, [333](#)
nrows0_, [333](#)
nrows_, [333](#)
originalColumn_, [335](#)
originalOffset_, [334](#)
originalRow_, [335](#)
rcosts_, [336](#)
rlo_, [335](#)
rowsBasic, [329](#)
rowStatusString, [330](#)
rowduals_, [336](#)
rowstat_, [336](#)
rup_, [335](#)
setArtificialStatus, [330](#)
setColLower, [330](#)
setColSolution, [330](#)
setColUpper, [330](#)
setColumnStatus, [329](#)
setColumnStatusUsingValue, [329](#)
setCost, [330](#)
setDualTolerance, [330](#)
setMessageHandler, [333](#)
setObjOffset, [330](#)
setObjSense, [330](#)
setPrimalTolerance, [330](#)
setReducedCost, [331](#)
setRowActivity, [331](#)
setRowLower, [331](#)
setRowPrice, [331](#)
setRowStatus, [329](#)
setRowStatusUsingValue, [329](#)
setRowUpper, [331](#)
setStatus, [330](#)
setStructuralStatus, [330](#)
sol_, [335](#)
Status, [328](#)
statusName, [333](#)
ztoldj_, [335](#)
ztolzb_, [335](#)
CoinPresolveAction, [337](#)
 ~CoinPresolveAction, [339](#)
 CoinPresolveAction, [339](#)
 CoinPresolveAction, [339](#)
 name, [339](#)
 next, [340](#)
 postsolve, [339](#)
 setNext, [339](#)
 throwCoinError, [339](#)
CoinPresolveDoubleton.hpp
 DOUBLETON, [534](#)
CoinPresolveDupcol.hpp
 DUPCOL, [535](#)
CoinPresolveEmpty.hpp
 DROP_COL, [535](#)
 DROP_ROW, [535](#)
CoinPresolveFixed.hpp
 FIXED_VARIABLE, [536](#)
CoinPresolveForcing.hpp
 IMPLIED_BOUND, [536](#)
CoinPresolveImpliedFree.hpp
 CoinPresolveImpliedFree_H, [537](#)
 IMPLIED_FREE, [537](#)
CoinPresolveImpliedFree_H
 CoinPresolveImpliedFree.hpp, [537](#)
CoinPresolveMatrix, [340](#)
 ~CoinPresolveMatrix, [346](#)
 addCol, [349](#)
 addRow, [351](#)
 anyInteger, [347](#)
 anyInteger_, [353](#)
 anyProhibited, [352](#)
 anyProhibited_, [355](#)
 assignPresolveToPostsolve, [352](#)
 change_bias, [347](#)
 clink_, [352](#)
 CoinPresolveMatrix, [346](#)
 CoinPresolveMatrix, [346](#)
 colChanged, [349](#)
 colChanged_, [354](#)

- colInfinite, [350](#)
- colProhibited, [349](#)
- colProhibited2, [349](#)
- colUsed, [350](#)
- colsToDo_, [354](#)
- countEmptyRows, [346](#)
- deleteStuff, [348](#)
- dobias_, [352](#)
- feasibilityTolerance, [348](#)
- feasibilityTolerance_, [353](#)
- getColIndicesByRow, [347](#)
- getElementsByRow, [347](#)
- getRowStarts, [347](#)
- hcol_, [353](#)
- hinrow_, [352](#)
- infiniteDown_, [356](#)
- infiniteUp_, [356](#)
- initColsToDo, [349](#)
- initRowsToDo, [350](#)
- initializeStuff, [348](#)
- integerType_, [353](#)
- isInteger, [347](#)
- maxSubstLevel_, [353](#)
- mrstrt_, [352](#)
- nextColsToDo_, [354](#)
- nextRowsToDo_, [355](#)
- numberColsToDo, [349](#)
- numberColsToDo_, [354](#)
- numberNextColsToDo_, [354](#)
- numberNextRowsToDo_, [355](#)
- numberRowsToDo, [350](#)
- numberRowsToDo_, [355](#)
- pass_, [353](#)
- presolveOptions, [347](#)
- presolveOptions_, [355](#)
- randomNumber_, [356](#)
- recomputeSums, [348](#)
- rlink_, [352](#)
- rowChanged, [350](#)
- rowChanged_, [354](#)
- rowProhibited, [351](#)
- rowProhibited2, [351](#)
- rowUsed, [351](#)
- rowels_, [352](#)
- rowsToDo_, [354](#)
- setAnyInteger, [347](#)
- setAnyProhibited, [352](#)
- setColChanged, [349](#)
- setColInfinite, [350](#)
- setColProhibited, [349](#)
- setColUsed, [350](#)
- setFeasibilityTolerance, [348](#)
- setMatrix, [346](#)
- setMaximumSubstitutionLevel, [348](#)

- setPass, [348](#)
- setPresolveOptions, [347](#)
- setRowChanged, [351](#)
- setRowProhibited, [351](#)
- setRowUsed, [351](#)
- setStatus, [348](#)
- setVariableType, [346](#), [347](#)
- startTime_, [353](#)
- statistics, [348](#)
- status, [348](#)
- status_, [353](#)
- stepColsToDo, [349](#)
- stepRowsToDo, [350](#)
- sumDown_, [356](#)
- sumUp_, [356](#)
- tuning_, [353](#)
- unsetColChanged, [349](#)
- unsetColInfinite, [350](#)
- unsetColUsed, [350](#)
- unsetRowChanged, [351](#)
- unsetRowUsed, [351](#)
- update_model, [346](#)
- usefulColumnDouble_, [356](#)
- usefulColumnInt_, [356](#)
- usefulRowDouble_, [356](#)
- usefulRowInt_, [355](#)
- CoinPresolveMatrix.hpp
 - DIE, [539](#)
 - deleteAction, [538](#)
 - NO_LINK, [539](#)
 - PRESOLVE_INF, [539](#)
 - PRESOLVE_STMT, [539](#)
 - PRESOLVEASSERT, [538](#)
 - PRESOLVEFINITE, [539](#)
 - ZTOLDP, [539](#)
 - ZTOLDP2, [539](#)
- CoinPresolveMonitor, [357](#)
 - checkAndTell, [358](#)
 - CoinPresolveMonitor, [357](#)
 - CoinPresolveMonitor, [357](#)
- CoinPresolveSingleton.hpp
 - SLACK_DOUBLETON, [540](#)
 - SLACK_SINGLETON, [540](#)
- CoinPresolveSubst.hpp
 - implied_bounds, [541](#)
 - SUBST_ROW, [541](#)
- CoinPresolveTighten.hpp
 - DO_TIGHTEN, [541](#)
 - tighten_zero_cost, [541](#)
- CoinPresolveTripleton.hpp
 - TRIPLETON, [542](#)
- CoinPresolveUseless.hpp
 - USELESS, [542](#)
- CoinPresolveZeros.hpp

- DROP_ZERO, 543
- drop_zero_coefficients, 543
- CoinReferencedObject
 - CoinSmartPtr.hpp, 546
- CoinRelFltEq, 358
 - ~CoinRelFltEq, 359
 - CoinRelFltEq, 359
 - CoinRelFltEq, 359
 - operator(), 359
 - operator=, 359
- CoinSearchTree
 - ~CoinSearchTree, 360
 - CoinSearchTree, 360
 - CoinSearchTree, 360
 - compName, 361
 - fixTop, 360
 - realpop, 360
 - realpush, 360
- CoinSearchTree< Comp >, 359
- CoinSearchTree.hpp
 - CoinNodeAction, 544
 - operator<, 544
- CoinSearchTreeBase, 361
 - ~CoinSearchTreeBase, 362
 - candidateList_, 363
 - CoinSearchTreeBase, 362
 - CoinSearchTreeBase, 362
 - compName, 362
 - empty, 362
 - fixTop, 362
 - getCandidates, 362
 - numInserted, 362
 - numInserted_, 363
 - pop, 363
 - push, 363
 - realpop, 362
 - realpush, 362
 - size, 362
 - size_, 363
 - top, 362
- CoinSearchTreeCompareBest, 363
 - name, 364
 - operator(), 364
- CoinSearchTreeCompareBreadth, 364
 - name, 364
 - operator(), 364
- CoinSearchTreeCompareDepth, 365
 - name, 365
 - operator(), 365
- CoinSearchTreeComparePreferred, 365
 - name, 366
 - operator(), 366
- CoinSearchTreeManager, 367
 - ~CoinSearchTreeManager, 367
- bestQuality, 367
- bestQualityCandidate, 367
- CoinSearchTreeManager, 367
- CoinSearchTreeManager, 367
- empty, 367
- getTree, 367
- newSolution, 368
- numInserted, 367
- pop, 367
- push, 367
- reevaluateSearchStrategy, 368
- setTree, 367
- size, 367
- top, 367
- CoinSeedRandom
 - CoinHelperFunctions.hpp, 512
- CoinSet, 368
 - ~CoinSet, 369
 - CoinSet, 369
 - CoinSet, 369
 - numberEntries, 369
 - numberEntries_, 370
 - operator=, 369
 - setType, 369
 - setType_, 370
 - weights, 370
 - weights_, 370
 - which, 369
 - which_, 370
- CoinShallowPackedVector, 370
 - ~CoinShallowPackedVector, 373
 - clear, 373
 - CoinShallowPackedVector, 372
 - CoinShallowPackedVectorUnitTest, 373
 - CoinShallowPackedVector, 372
 - getElements, 373
 - getIndices, 373
 - getNumElements, 373
 - operator=, 373
 - print, 373
 - setVector, 373
- CoinShallowPackedVector.hpp
 - CoinShallowPackedVectorUnitTest, 544
- CoinShallowPackedVectorUnitTest
 - CoinShallowPackedVector, 373
 - CoinShallowPackedVector.hpp, 544
- CoinShortSort_2
 - CoinSort.hpp, 549
- CoinSigHandler_t
 - CoinSignal.hpp, 545
- CoinSignal.hpp
 - CoinSigHandler_t, 545
- CoinSimpFactorization, 374
 - ~CoinSimpFactorization, 380

allocateSomeArrays, 384
allocateSpaceForU, 384
auxInd_, 386
auxVector_, 386
btran, 386
checkPivot, 386
clearArrays, 382
clone, 380
CoinSimpFactorization, 380
CoinSimpFactorizationUnitTest, 386
CoinSimpFactorization, 380
colOfU_, 390
colPosition_, 390
colSlack_, 390
copyLbyRows, 383
copyRowPermutations, 385
copyUbyColumns, 383
denseVector_, 386
doSuhlHeuristic_, 392
enlargeUcol, 384
enlargeUrow, 384
Eta_, 391
EtaInd_, 391
EtaLengths_, 391
EtaMaxCap_, 392
EtaPosition_, 391
EtaSize_, 391
EtaStarts_, 391
factor, 381
factorize, 383
findInColumn, 384
findInRow, 384
findMaxInRow, 384
findPivot, 383
findPivotShCol, 383
findPivotSimp, 383
findShortColumn, 383
findShortRow, 383
firstColInU_, 390
firstNumberSlacks_, 393
firstRowInU_, 389
ftran, 385
ftran2, 386
GaussEliminate, 383
getAreas, 381
gutsOfCopy, 383
gutsOfDestructor, 383
gutsOfInitialize, 383
Hxeqb, 385
Hxeqb2, 385
increaseColSize, 384
increaseLsize, 384
increaseRowSize, 384
indKeep_, 387
indVector_, 386
indices, 382
initialSomeNumbers, 385
invOfPivots_, 390
keepSize_, 387
LUupdate, 385
lastColInU_, 390
lastEtaRow_, 391
lastRowInU_, 389
LcolCap_, 388
LcolInd_, 388
LcolLengths_, 388
LcolSize_, 388
LcolStarts_, 387
Lcolumns_, 388
LrowCap_, 387
LrowInd_, 387
LrowLengths_, 387
LrowSize_, 387
LrowStarts_, 387
Lrows_, 387
Lxeqb, 385
Lxeqb2, 385
mainLoopFactor, 383
makeNonSingular, 381
maxA_, 392
maxEtaRows_, 392
maxGrowth_, 392
maxU_, 392
maximumCoefficient, 381
minIncrease_, 392
newEta, 385
nextColInU_, 390
nextRowInU_, 389
numberElements, 381
numberSlacks_, 392
operator=, 380
permute, 382
pivotCandLimit_, 392
pivoting, 384
postProcess, 381
preProcess, 381
prevColInU_, 389
prevRowInU_, 389
removeColumnFromActSet, 384
removeRowFromActSet, 384
replaceColumn, 381
rowOfU_, 390
rowPosition_, 391
secRowOfU_, 391
secRowPosition_, 391
UcolEnd_, 390
UcolInd_, 389
UcolLengths_, 389

- UcolMaxCap_, 390
- UcolStarts_, 389
- Ucolumns_, 389
- upColumn, 382
- upColumnTranspose, 382
- updateColumn, 382
- updateColumnFT, 381
- updateColumnTranspose, 382
- updateCurrentRow, 384
- updateToI_, 392
- updateTwoColumnsFT, 382
- UrowEnd_, 389
- UrowInd_, 388
- UrowLengths_, 388
- UrowMaxCap_, 388
- UrowStarts_, 388
- Urows_, 388
- Uxeqb, 385
- Uxeqb2, 385
- vecKeep_, 387
- vecLabels_, 386
- workArea2_, 386
- workArea3_, 386
- xHeqb, 385
- xLeqb, 385
- xUeqb, 385
- CoinSimpFactorizationUnitTest
 - CoinSimpFactorization, 386
- CoinSizeofAsInt
 - CoinHelperFunctions.hpp, 508
- CoinSmartPtr
 - CoinSmartPtr.hpp, 546
- CoinSmartPtr.hpp
 - CoinComparePointers, 546
 - CoinReferencedObject, 546
 - CoinSmartPtr, 546
 - dbg_smartptr_verbosity, 546
 - operator==, 546, 547
- CoinSnapshot, 393
 - ~CoinSnapshot, 396
 - CoinSnapshot, 396
 - CoinSnapshot, 396
 - createMatrixByRow, 401
 - createRightHandSide, 401
 - getColLower, 397
 - getColSolution, 399
 - getColType, 398
 - getColUpper, 397
 - getDoNotSeparateThis, 399
 - getDualTolerance, 399
 - getInfinity, 399
 - getIntegerLowerBound, 400
 - getIntegerTolerance, 400
 - getIntegerUpperBound, 400
 - getMatrixByCol, 398
 - getMatrixByRow, 398
 - getNumCols, 397
 - getNumElements, 397
 - getNumIntegers, 397
 - getNumRows, 397
 - getObjCoefficients, 397
 - getObjOffset, 399
 - getObjSense, 398
 - getObjValue, 399
 - getOriginalMatrixByCol, 398
 - getOriginalMatrixByRow, 398
 - getPrimalTolerance, 399
 - getReducedCost, 399
 - getRightHandSide, 397
 - getRowActivity, 399
 - getRowLower, 397
 - getRowPrice, 399
 - getRowUpper, 397
 - isBinary, 398
 - isContinuous, 398
 - isFreeBinary, 398
 - isInteger, 398
 - isIntegerNonBinary, 398
 - loadProblem, 400
 - operator=, 403
 - setColLower, 401
 - setColSolution, 402
 - setColType, 401
 - setColUpper, 401
 - setDoNotSeparateThis, 402
 - setDualTolerance, 402
 - setInfinity, 402
 - setIntegerLowerBound, 403
 - setIntegerTolerance, 403
 - setIntegerUpperBound, 403
 - setMatrixByCol, 401
 - setMatrixByRow, 401
 - setNumCols, 400
 - setNumElements, 400
 - setNumIntegers, 400
 - setNumRows, 400
 - setObjCoefficients, 401
 - setObjOffset, 402
 - setObjSense, 401
 - setObjValue, 402
 - setOriginalMatrixByCol, 402
 - setOriginalMatrixByRow, 402
 - setPrimalTolerance, 402
 - setReducedCost, 402
 - setRightHandSide, 401
 - setRowActivity, 402
 - setRowLower, 401
 - setRowPrice, 402

- setRowUpper, [401](#)
- CoinSort.hpp
 - CoinDecrSolutionOrdered, [549](#)
 - CoinIncrSolutionOrdered, [549](#)
 - CoinShortSort_2, [549](#)
 - CoinSort_2, [549](#)
 - CoinSort_2Std, [549](#)
 - CoinSort_3, [549](#)
- CoinSort_2
 - CoinSort.hpp, [549](#)
- CoinSort_2Std
 - CoinSort.hpp, [549](#)
- CoinSort_3
 - CoinSort.hpp, [549](#)
- CoinSosSet, [403](#)
 - ~CoinSosSet, [404](#)
 - CoinSosSet, [404](#)
 - CoinSosSet, [404](#)
- CoinStrNCaseCmp
 - CoinHelperFunctions.hpp, [512](#)
- CoinStrdup
 - CoinHelperFunctions.hpp, [510](#)
- CoinStrlenAsInt
 - CoinHelperFunctions.hpp, [512](#)
- CoinStructuredModel, [404](#)
 - ~CoinStructuredModel, [406](#)
 - addBlock, [406](#), [407](#)
 - addColumnBlock, [408](#)
 - addRowBlock, [408](#)
 - block, [409](#)
 - blockIndex, [409](#)
 - blockType, [408](#)
 - clone, [409](#)
 - coinBlock, [409](#)
 - coinModelBlock, [409](#)
 - CoinStructuredModel, [406](#)
 - CoinStructuredModel, [406](#)
 - columnBlock, [408](#)
 - decompose, [407](#)
 - getColumnBlock, [408](#)
 - getRowBlock, [408](#)
 - numberColumnBlocks, [407](#)
 - numberElementBlocks, [408](#)
 - numberElements, [408](#)
 - numberRowBlocks, [407](#)
 - operator=, [409](#)
 - optimizationDirection, [409](#)
 - refresh, [409](#)
 - rowBlock, [408](#)
 - setCoinModel, [409](#)
 - setColumnBlock, [408](#)
 - setOptimizationDirection, [409](#)
 - setRowBlock, [408](#)
 - writeMps, [407](#)
- CoinStructuredModel.hpp
 - CoinModelBlockInfo, [550](#)
- CoinSwap
 - CoinHelperFunctions.hpp, [512](#)
- CoinSysTime
 - CoinTime.hpp, [551](#)
- CoinThreadRandom, [410](#)
 - ~CoinThreadRandom, [411](#)
 - CoinThreadRandom, [411](#)
 - CoinThreadRandom, [411](#)
 - getSeed, [411](#)
 - operator=, [411](#)
 - randomDouble, [411](#)
 - randomize, [411](#)
 - seed_, [411](#)
 - setSeed, [411](#)
- CoinTime.hpp
 - CoinCpuTime, [551](#)
 - CoinCpuTimeJustChildren, [551](#)
 - CoinGetTimeOfDay, [551](#)
 - CoinSysTime, [551](#)
 - CoinWallclockTime, [551](#)
- CoinTimer, [412](#)
 - CoinTimer, [413](#)
 - CoinTimer, [413](#)
 - isExpired, [413](#)
 - isPast, [413](#)
 - isPastPercent, [413](#)
 - reset, [413](#)
 - restart, [413](#)
 - setLimit, [414](#)
 - timeElapsed, [413](#)
 - timeLeft, [413](#)
- CoinToFile
 - CoinHelperFunctions.hpp, [512](#)
- CoinTreeNode, [414](#)
 - ~CoinTreeNode, [415](#)
 - CoinTreeNode, [415](#)
 - CoinTreeNode, [415](#)
 - getDepth, [415](#)
 - getFractionality, [415](#)
 - getPreferred, [415](#)
 - getQuality, [415](#)
 - getTrueLB, [415](#)
 - operator=, [415](#)
 - setDepth, [415](#)
 - setFractionality, [415](#)
 - setPreferred, [416](#)
 - setQuality, [415](#)
 - setTrueLB, [415](#)
- CoinTreeSiblings, [416](#)
 - ~CoinTreeSiblings, [416](#)
 - advanceNode, [416](#)
 - CoinTreeSiblings, [416](#)

- CoinTreeSiblings, 416
- currentNode, 416
- printPref, 417
- size, 417
- toProcess, 417
- CoinTriple
 - CoinTriple, 417
 - CoinTriple, 417
 - first, 418
 - second, 418
 - third, 418
- CoinTriple< S, T, U >, 417
- CoinTypes.hpp
 - COIN_BIG_DOUBLE, 552
 - COIN_BIG_INDEX, 552
 - COIN_LONG_WORK, 552
 - CoinBigIndex, 552
 - CoinFactorizationDouble, 552
 - CoinInt64, 551
 - CoinIntPtr, 552
 - CoinUInt64, 551
 - CoinWorkDouble, 552
- CoinUInt64
 - CoinTypes.hpp, 551
- CoinUnsignedIntArrayWithLength, 418
 - array, 420
 - CoinUnsignedIntArrayWithLength, 419
 - CoinUnsignedIntArrayWithLength, 419
 - conditionalNew, 420
 - getSize, 420
 - operator=, 420
 - setSize, 420
- CoinUtility.hpp
 - CoinMakePair, 552
 - CoinMakeTriple, 552
- CoinVoidStarArrayWithLength, 420
 - array, 422
 - CoinVoidStarArrayWithLength, 421, 422
 - CoinVoidStarArrayWithLength, 421, 422
 - conditionalNew, 422
 - getSize, 422
 - operator=, 422
 - setSize, 422
- CoinWallclockTime
 - CoinTime.hpp, 551
- CoinWarmStart, 422
 - ~CoinWarmStart, 423
 - applyDiff, 423
 - clone, 423
 - generateDiff, 423
- CoinWarmStartBasis, 424
 - ~CoinWarmStartBasis, 427
 - applyDiff, 429
 - artificialStatus_, 431
 - assignBasisStatus, 430
 - clone, 430
 - CoinWarmStartBasis, 427
 - CoinWarmStartBasis, 427
 - compressRows, 429
 - deleteColumns, 430
 - deleteRows, 429
 - fixFullBasis, 431
 - fullBasis, 430
 - generateDiff, 429
 - getArtifStatus, 428
 - getArtificialStatus, 428, 429
 - getNumArtificial, 428
 - getNumStructural, 428
 - getStatus, 431
 - getStructStatus, 428
 - getStructuralStatus, 428
 - maxSize_, 431
 - mergeBasis, 430
 - numArtificial_, 431
 - numStructural_, 431
 - numberBasicStructurals, 428
 - operator=, 430
 - print, 430
 - resize, 429
 - setArtifStatus, 428
 - setSize, 429
 - setStatus, 431
 - setStructStatus, 428
 - Status, 427
 - statusName, 431
 - structuralStatus_, 431
 - XferEntry, 426
 - XferVec, 426
- CoinWarmStartBasis::applyDiff
 - CoinWarmStartBasisDiff, 434
- CoinWarmStartBasis::generateDiff
 - CoinWarmStartBasisDiff, 434
- CoinWarmStartBasisDiff, 432
 - ~CoinWarmStartBasisDiff, 433
 - clone, 433
 - CoinWarmStartBasis::applyDiff, 434
 - CoinWarmStartBasis::generateDiff, 434
 - CoinWarmStartBasisDiff, 433
 - CoinWarmStartBasisDiff, 433
 - operator=, 433
- CoinWarmStartDiff, 434
 - ~CoinWarmStartDiff, 434
 - clone, 435
- CoinWarmStartDual, 435
 - ~CoinWarmStartDual, 436
 - applyDiff, 437
 - assignDual, 436
 - clone, 436

- CoinWarmStartDual, [436](#)
- CoinWarmStartDual, [436](#)
- dual, [436](#)
- generateDiff, [436](#)
- operator=, [436](#)
- size, [436](#)
- CoinWarmStartDual::applyDiff
 - CoinWarmStartDualDiff, [439](#)
- CoinWarmStartDual::generateDiff
 - CoinWarmStartDualDiff, [439](#)
- CoinWarmStartDualDiff, [437](#)
 - ~CoinWarmStartDualDiff, [438](#)
 - clone, [438](#)
 - CoinWarmStartDual::applyDiff, [439](#)
 - CoinWarmStartDual::generateDiff, [439](#)
 - CoinWarmStartDualDiff, [438](#)
 - CoinWarmStartDualDiff, [438](#)
 - operator=, [438](#)
- CoinWarmStartPrimalDual, [439](#)
 - ~CoinWarmStartPrimalDual, [440](#)
 - applyDiff, [441](#)
 - assign, [441](#)
 - clear, [441](#)
 - clone, [441](#)
 - CoinWarmStartPrimalDual, [440](#)
 - CoinWarmStartPrimalDual, [440](#)
 - dual, [440](#)
 - dualSize, [440](#)
 - generateDiff, [441](#)
 - operator=, [441](#)
 - primal, [440](#)
 - primalSize, [440](#)
 - swap, [441](#)
- CoinWarmStartPrimalDual::applyDiff
 - CoinWarmStartPrimalDualDiff, [443](#)
- CoinWarmStartPrimalDual::generateDiff
 - CoinWarmStartPrimalDualDiff, [443](#)
- CoinWarmStartPrimalDualDiff, [442](#)
 - ~CoinWarmStartPrimalDualDiff, [443](#)
 - clear, [443](#)
 - clone, [443](#)
 - CoinWarmStartPrimalDual::applyDiff, [443](#)
 - CoinWarmStartPrimalDual::generateDiff, [443](#)
 - CoinWarmStartPrimalDualDiff, [443](#)
 - CoinWarmStartPrimalDualDiff, [443](#)
 - swap, [443](#)
- CoinWarmStartVector
 - ~CoinWarmStartVector, [445](#)
 - applyDiff, [446](#)
 - assignVector, [445](#)
 - clear, [446](#)
 - clone, [446](#)
 - CoinWarmStartVector, [445](#)
 - CoinWarmStartVector, [445](#)
- CoinWarmStartVectorDiff, [449](#)
 - generateDiff, [446](#)
 - gutsOfCopy, [445](#)
 - gutsOfDestructor, [445](#)
 - operator=, [446](#)
 - size, [445](#)
 - swap, [446](#)
 - values, [445](#)
- CoinWarmStartVector< T >, [444](#)
- CoinWarmStartVectorDiff
 - ~CoinWarmStartVectorDiff, [448](#)
 - clear, [448](#)
 - clone, [448](#)
 - CoinWarmStartVector, [449](#)
 - CoinWarmStartVectorDiff, [448](#)
 - CoinWarmStartVectorDiff, [448](#)
 - operator=, [448](#)
 - swap, [448](#)
- CoinWarmStartVectorDiff< T >, [446](#)
- CoinWarmStartVectorPair
 - ~CoinWarmStartVectorPair, [450](#)
 - applyDiff, [451](#)
 - assignVector0, [450](#)
 - assignVector1, [450](#)
 - clear, [451](#)
 - clone, [451](#)
 - CoinWarmStartVectorPair, [450](#)
 - CoinWarmStartVectorPair, [450](#)
 - CoinWarmStartVectorPairDiff, [453](#)
 - generateDiff, [451](#)
 - operator=, [450](#)
 - size0, [450](#)
 - size1, [450](#)
 - swap, [450](#)
 - values0, [450](#)
 - values1, [450](#)
- CoinWarmStartVectorPair< T, U >, [449](#)
- CoinWarmStartVectorPairDiff
 - ~CoinWarmStartVectorPairDiff, [452](#)
 - clear, [452](#)
 - clone, [452](#)
 - CoinWarmStartVectorPair, [453](#)
 - CoinWarmStartVectorPairDiff, [452](#)
 - CoinWarmStartVectorPairDiff, [452](#)
 - operator=, [452](#)
 - swap, [452](#)
- CoinWarmStartVectorPairDiff< T, U >, [451](#)
- CoinWorkDouble
 - CoinTypes.hpp, [552](#)
- CoinYacc, [453](#)
 - ~CoinYacc, [453](#)
 - CoinYacc, [453](#)
 - CoinYacc, [453](#)
 - length, [453](#)

- [symbuf](#), [453](#)
 - [symtable](#), [453](#)
 - [unsetValue](#), [454](#)
- [CoinZero](#)
 - [CoinHelperFunctions.hpp](#), [510](#)
- [CoinZeroN](#)
 - [CoinHelperFunctions.hpp](#), [510](#)
- [col](#)
 - [dropped_zero](#), [461](#)
 - [remove_fixed_action::action](#), [41](#)
- [colChanged](#)
 - [CoinPresolveMatrix](#), [349](#)
- [colChanged_](#)
 - [CoinPresolveMatrix](#), [354](#)
- [colInfinite](#)
 - [CoinPresolveMatrix](#), [350](#)
- [colOfU_](#)
 - [CoinSimpFactorization](#), [390](#)
- [colOrdered_](#)
 - [CoinPackedMatrix](#), [287](#)
- [colPosition_](#)
 - [CoinSimpFactorization](#), [390](#)
- [colProhibited](#)
 - [CoinPresolveMatrix](#), [349](#)
- [colProhibited2](#)
 - [CoinPresolveMatrix](#), [349](#)
- [colSlack_](#)
 - [CoinSimpFactorization](#), [390](#)
- [colUsed](#)
 - [CoinPresolveMatrix](#), [350](#)
- [colEl](#)
 - [doubleton_action::action](#), [40](#)
 - [tripleton_action::action](#), [44](#)
- [colElS_](#)
 - [CoinPrePostsolveMatrix](#), [334](#)
 - [remove_fixed_action](#), [480](#)
- [collectStatistics](#)
 - [CoinFactorization](#), [100](#)
- [collectStatistics_](#)
 - [CoinFactorization](#), [113](#)
- [collower_](#)
 - [CoinLpIO](#), [165](#)
 - [CoinMpsIO](#), [245](#)
- [colRows_](#)
 - [remove_fixed_action](#), [480](#)
- [colsToDo_](#)
 - [CoinPresolveMatrix](#), [354](#)
- [colstat_](#)
 - [CoinPrePostsolveMatrix](#), [336](#)
- [column](#)
 - [CoinBuild](#), [63](#)
 - [CoinModel](#), [202](#)
 - [CoinModelLink](#), [215](#)
 - [CoinModelTriple](#), [220](#)
- [columnBlock](#)
 - [CoinModelInfo2](#), [213](#)
 - [CoinStructuredModel](#), [408](#)
- [columnBlockName_](#)
 - [CoinBaseModel](#), [58](#)
- [columnIndex](#)
 - [CoinLpIO](#), [157](#)
 - [CoinMpsIO](#), [236](#)
- [columnIsBasic](#)
 - [CoinPrePostsolveMatrix](#), [329](#)
- [columnIsInteger](#)
 - [CoinModel](#), [200](#)
- [columnIsIntegerAsString](#)
 - [CoinModel](#), [202](#)
- [columnLower](#)
 - [CoinModel](#), [200](#)
- [columnLowerArray](#)
 - [CoinModel](#), [203](#)
- [columnLowerAsString](#)
 - [CoinModel](#), [202](#)
- [columnName](#)
 - [CoinLpIO](#), [157](#)
 - [CoinModel](#), [200](#)
 - [CoinModelInfo2](#), [213](#)
 - [CoinMpsCardReader](#), [224](#)
 - [CoinMpsIO](#), [236](#)
- [columnName_](#)
 - [CoinMpsCardReader](#), [226](#)
- [columnNames](#)
 - [CoinModel](#), [204](#)
- [columnObjective](#)
 - [CoinModel](#), [200](#)
- [columnObjectiveAsString](#)
 - [CoinModel](#), [202](#)
- [columnStatusString](#)
 - [CoinPrePostsolveMatrix](#), [330](#)
- [columnUpper](#)
 - [CoinModel](#), [200](#)
- [columnUpperArray](#)
 - [CoinModel](#), [203](#)
- [columnUpperAsString](#)
 - [CoinModel](#), [202](#)
- [colupper_](#)
 - [CoinLpIO](#), [165](#)
 - [CoinMpsIO](#), [245](#)
- [compName](#)
 - [CoinSearchTree](#), [361](#)
 - [CoinSearchTreeBase](#), [362](#)
- [compact](#)
 - [CoinPartitionedVector](#), [317](#)
- [compare](#)
 - [CoinPackedVectorBase](#), [301](#)
- [ComparePointers](#)
 - [Coin](#), [29](#)

- compress
 - CoinPackedMatrix, [279](#)
- compressRows
 - CoinWarmStartBasis, [429](#)
- Compression
 - CoinFileOutput, [124](#)
- compressionSupported
 - CoinFileOutput, [124](#)
- computeAssociated
 - CoinModel, [206](#)
- computeNumberElements
 - CoinPartitionedVector, [317](#)
- conditionNumber
 - CoinFactorization, [93](#)
 - CoinOslFactorization, [255](#)
- conditionalDelete
 - CoinArrayWithLength, [53](#)
- conditionalNew
 - CoinArbitraryArrayWithLength, [48](#)
 - CoinArrayWithLength, [53](#)
 - CoinBigIndexArrayWithLength, [60](#)
 - CoinDoubleArrayWithLength, [75](#)
 - CoinFactorizationDoubleArrayWithLength, [116](#)
 - CoinFactorizationLongDoubleArrayWithLength, [118](#)
 - CoinIntArrayWithLength, [147](#)
 - CoinUnsignedIntArrayWithLength, [420](#)
 - CoinVoidStarArrayWithLength, [422](#)
- config_coinutils_default.h
 - COIN_INT64_T, [555](#)
 - COIN_INTPTR_T, [555](#)
 - COIN_UINT64_T, [555](#)
- convertBoundToSense
 - CoinLpIO, [155](#)
 - CoinMpsIO, [242](#)
- convertMatrix
 - CoinModel, [197](#)
- convertObjective_
 - CoinMpsIO, [246](#)
- convertRowToColumnU_
 - CoinFactorization, [110](#)
- convertSenseToBound
 - CoinMpsIO, [242](#)
- copy
 - CoinArrayWithLength, [53](#)
 - CoinIndexedVector, [139](#)
- copyInIntegerInformation
 - CoinMpsIO, [237](#)
- copyLbyRows
 - CoinSimpFactorization, [383](#)
- copyMaxMinIndex
 - CoinPackedVectorBase, [302](#)
- copyOf
 - CoinPackedMatrix, [280](#)
- copyReuseArrays
 - CoinPackedMatrix, [280](#)
- copyRowPermutations
 - CoinSimpFactorization, [385](#)
- copyStringElements
 - CoinMpsIO, [241](#)
- copyUbyColumns
 - CoinSimpFactorization, [383](#)
- cost_
 - CoinPrePostsolveMatrix, [334](#)
- costx
 - doubleton_action::action, [39](#)
 - tripleton_action::action, [43](#)
- costy
 - doubleton_action::action, [40](#)
 - tripleton_action::action, [43](#)
- countEmptyCols
 - CoinPrePostsolveMatrix, [332](#)
- countEmptyRows
 - CoinPresolveMatrix, [346](#)
- countOrthoLength
 - CoinPackedMatrix, [281](#)
- countPlusMinusOne
 - CoinModel, [203](#)
- create
 - CoinFileInput, [120](#)
 - CoinFileOutput, [124](#)
 - CoinModelLinkedList, [218](#)
- createArrays
 - CoinModel, [203](#)
- createMatrixByRow
 - CoinSnapshot, [401](#)
- createOneUnpackedElement
 - CoinIndexedVector, [142](#)
- createPacked
 - CoinIndexedVector, [141](#)
- createPackedMatrix
 - CoinModel, [203](#)
- createPlusMinusOne
 - CoinModel, [203](#)
- createRightHandSide
 - CoinSnapshot, [401](#)
- createUnpacked
 - CoinIndexedVector, [142](#)
- cup_
 - CoinPrePostsolveMatrix, [335](#)
- cupx
 - doubleton_action::action, [39](#)
 - tripleton_action::action, [43](#)
- cupy
 - tripleton_action::action, [43](#)
- currentColumn
 - CoinBuild, [63](#), [64](#)
- currentMessage
 - CoinMessageHandler, [175](#)

- currentMessage_
 - CoinMessageHandler, [178](#)
- currentNode
 - CoinTreeSiblings, [416](#)
- currentRow
 - CoinBuild, [63](#)
- currentSource
 - CoinMessageHandler, [175](#)
- cutMarker
 - CoinModel, [204](#)
- cvec
 - Coin_C_defines.h, [497](#)
- DIE
 - CoinPresolveMatrix.hpp, [539](#)
- DO_TIGHTEN
 - CoinPresolveTighten.hpp, [541](#)
- DOUBLETON
 - CoinPresolveDoubleton.hpp, [534](#)
- DROP_COL
 - CoinPresolveEmpty.hpp, [535](#)
- DROP_ROW
 - CoinPresolveEmpty.hpp, [535](#)
- DROP_ZERO
 - CoinPresolveZeros.hpp, [543](#)
- DUPCOL
 - CoinPresolveDupcol.hpp, [535](#)
- dbg_smartptr_verbosity
 - CoinSmartPtr.hpp, [546](#)
- dblVal
 - CoinParam, [309](#)
- dealWithFileName
 - CoinMpsIO, [242](#)
- decimals_
 - CoinLpIO, [166](#)
- decodeString
 - CoinMpsIO, [243](#)
- decompose
 - CoinStructuredModel, [407](#)
- defaultBound_
 - CoinMpsIO, [246](#)
- defaultHandler_
 - CoinLpIO, [164](#)
 - CoinMpsIO, [246](#)
 - CoinPrePostsolveMatrix, [336](#)
- deleteAction
 - CoinPresolveMatrix.hpp, [538](#)
- deleteCol
 - CoinModel, [195](#)
- deleteCols
 - CoinPackedMatrix, [278](#)
- deleteColumn
 - CoinFactorization, [100](#)
 - CoinModel, [195](#)
- deleteColumns
 - CoinWarmStartBasis, [430](#)
- deleteElement
 - CoinModel, [195](#)
- deleteHash
 - CoinModelHash, [209](#)
 - CoinModelHash2, [211](#)
- deleteLink
 - CoinFactorization, [102](#)
- deleteMajorVectors
 - CoinPackedMatrix, [284](#)
- deleteMinorVectors
 - CoinPackedMatrix, [284](#)
- deleteRow
 - CoinFactorization, [100](#)
 - CoinModel, [195](#)
- deleteRowOne
 - CoinModelLinkedList, [219](#)
- deleteRows
 - CoinPackedMatrix, [278](#)
 - CoinWarmStartBasis, [429](#)
- deleteSame
 - CoinModelLinkedList, [219](#)
- deleteStuff
 - CoinPresolveMatrix, [348](#)
- deleteThisElement
 - CoinModel, [195](#)
- demark
 - _EKKfactinfo, [34](#)
- denseArea_
 - CoinFactorization, [111](#)
- densePermute_
 - CoinFactorization, [111](#)
- denseThreshold
 - CoinFactorization, [96](#)
- denseThreshold_
 - CoinFactorization, [112](#)
- denseVector
 - CoinIndexedVector, [138](#)
 - CoinPackedVectorBase, [300](#)
- denseVector_
 - CoinSimpFactorization, [386](#)
- detail
 - CoinMessageHandler, [173](#)
 - CoinOneMessage, [249](#)
- detail_
 - CoinOneMessage, [249](#)
- differentModel
 - CoinModel, [197](#)
- display
 - CoinParam, [311](#)
- do_tighten_action, [454](#)
 - ~do_tighten_action, [454](#)
 - name, [454](#)

- postsolve, 455
- presolve, 455
- doForrestTomlin_
 - CoinFactorization, 111
- doSuhlHeuristic_
 - CoinSimpFactorization, 392
- dobias_
 - CoinPresolveMatrix, 352
- dotProduct
 - CoinPackedVectorBase, 301
- doubleValue
 - CoinMessageHandler, 174
- doubleValue_
 - CoinMessageHandler, 177
- doubleton_action, 455
 - ~doubleton_action, 456
 - actions_, 456
 - nactions_, 456
 - name, 456
 - postsolve, 456
 - presolve, 456
- doubleton_action::action, 39
 - clox, 39
 - coeffx, 40
 - coeffy, 40
 - colcl, 40
 - costx, 39
 - costy, 40
 - cupx, 39
 - icolx, 40
 - icoly, 40
 - ncolx, 40
 - ncoly, 40
 - rlo, 40
 - row, 40
- drop_empty_cols_action, 456
 - ~drop_empty_cols_action, 457
 - name, 458
 - postsolve, 458
 - presolve, 458
- drop_empty_rows_action, 458
 - ~drop_empty_rows_action, 459
 - name, 459
 - postsolve, 459
 - presolve, 459
- drop_zero_coefficients
 - CoinPresolveZeros.hpp, 543
- drop_zero_coefficients_action, 459
 - ~drop_zero_coefficients_action, 460
 - name, 460
 - postsolve, 460
 - presolve, 460
- dropped_zero, 461
 - col, 461
 - row, 461
- drtpiv
 - _EKKfactinfo, 34
- dual
 - CoinWarmStartDual, 436
 - CoinWarmStartPrimalDual, 440
- dualSize
 - CoinWarmStartPrimalDual, 440
- dumpMatrix
 - CoinPackedMatrix, 282
- dupcol_action, 461
 - ~dupcol_action, 462
 - name, 462
 - postsolve, 462
 - presolve, 462
- duplicateIndex
 - CoinPackedVectorBase, 300
- duprow_action, 463
 - name, 463
 - postsolve, 464
 - presolve, 464
- dvec
 - Coin_C_defines.h, 496
- EKKHlink, 464
 - pre, 464
 - suc, 464
- EKKfactinfo
 - CoinOslFactorization.hpp, 528
- eightChar_
 - CoinMpsCardReader, 226
- element
 - CoinModelLink, 215
- element_
 - CoinPackedMatrix, 287
- elementByRowL
 - CoinFactorization, 94
- elementByRowL_
 - CoinFactorization, 114
- elementL_
 - CoinFactorization, 110
- elementR_
 - CoinFactorization, 111
- elementU
 - CoinFactorization, 98
- elementU_
 - CoinFactorization, 110
- elements
 - CoinModel, 198
 - CoinOslFactorization, 254
 - CoinOtherFactorization, 262
- elements_
 - CoinIndexedVector, 145
 - CoinOtherFactorization, 266

- eliminateDuplicates
 - CoinPackedMatrix, [279](#)
- empty
 - CoinIndexedVector, [139](#)
 - CoinSearchTreeBase, [362](#)
 - CoinSearchTreeManager, [367](#)
- emptyRows
 - CoinFactorization, [100](#)
- enlargeUcol
 - CoinSimpFactorization, [384](#)
- enlargeUrow
 - CoinSimpFactorization, [384](#)
- eol_
 - CoinMpsCardReader, [226](#)
- epsilon_
 - CoinLpIO, [166](#)
- Eta_
 - CoinSimpFactorization, [391](#)
- eta_size
 - _EKKfactinfo, [39](#)
- EtaInd_
 - CoinSimpFactorization, [391](#)
- EtaLengths_
 - CoinSimpFactorization, [391](#)
- EtaMaxCap_
 - CoinSimpFactorization, [392](#)
- EtaPosition_
 - CoinSimpFactorization, [391](#)
- EtaSize_
 - CoinSimpFactorization, [391](#)
- EtaStarts_
 - CoinSimpFactorization, [391](#)
- expand
 - CoinIndexedVector, [142](#)
- expandKnapsack
 - CoinModel, [206](#)
- extend
 - CoinArrayWithLength, [53](#)
- externalNumber
 - CoinOneMessage, [249](#)
- externalNumber_
 - CoinOneMessage, [249](#)
- extraGap_
 - CoinPackedMatrix, [287](#)
- extraMajor_
 - CoinPackedMatrix, [287](#)
- FIXED_VARIABLE
 - CoinPresolveFixed.hpp, [536](#)
- factInfo_
 - CoinOslFactorization, [256](#)
- factor
 - CoinDenseFactorization, [67](#)
 - CoinFactorization, [101](#)
 - CoinOslFactorization, [253](#)
 - CoinOtherFactorization, [264](#)
 - CoinSimpFactorization, [381](#)
- factorDense
 - CoinFactorization, [101](#)
- factorElements_
 - CoinFactorization, [107](#)
 - CoinOtherFactorization, [265](#)
- FactorPointers, [464](#)
 - ~FactorPointers, [465](#)
 - FactorPointers, [465](#)
 - FactorPointers, [465](#)
 - firstColKnonzeros, [465](#)
 - firstRowKnonzeros, [465](#)
 - newCols, [466](#)
 - nextColumn, [465](#)
 - nextRow, [465](#)
 - prevColumn, [465](#)
 - prevRow, [465](#)
 - rowMax, [465](#)
- factorSparse
 - CoinFactorization, [101](#)
- factorSparseLarge
 - CoinFactorization, [101](#)
- factorSparseSmall
 - CoinFactorization, [101](#)
- factorize
 - CoinFactorization, [92](#)
 - CoinOslFactorization, [253](#)
 - CoinSimpFactorization, [383](#)
- factorizePart1
 - CoinFactorization, [93](#)
- factorizePart2
 - CoinFactorization, [93](#)
- feasibilityTolerance
 - CoinPresolveMatrix, [348](#)
- feasibilityTolerance_
 - CoinPresolveMatrix, [353](#)
- fileAbsPath
 - CoinFileInput, [121](#)
- fileCoinReadable
 - CoinFileInput, [121](#)
- fileInput
 - CoinMpsCardReader, [225](#)
- fileName
 - CoinError, [77](#)
- fileName_
 - CoinLpIO, [166](#)
 - CoinMpsIO, [245](#)
- filePointer
 - CoinMessageHandler, [176](#)
- fill
 - CoinModelLinkedList, [220](#)
- find_obj

- CoinLpIO, 162
- findHash
 - CoinLpIO, 162
 - CoinMpsIO, 243
- findInColumn
 - CoinSimpFactorization, 384
- findInRow
 - CoinSimpFactorization, 384
- findIndex
 - CoinPackedVectorBase, 300
- findMaxInRow
 - CoinSimpFactorization, 384
- findMaxMinIndices
 - CoinPackedVectorBase, 301
- findPivot
 - CoinSimpFactorization, 383
- findPivotShCol
 - CoinSimpFactorization, 383
- findPivotSimp
 - CoinSimpFactorization, 383
- findShortColumn
 - CoinSimpFactorization, 383
- findShortRow
 - CoinSimpFactorization, 383
- finish
 - CoinMessageHandler, 177
- first
 - CoinModelLinkedList, 218
 - CoinPair, 303
 - CoinTriple, 418
- first_dense
 - _EKKfactinfo, 37
- first_is_number
 - CoinLpIO, 162
- firstColInU_
 - CoinSimpFactorization, 390
- firstColKnonzeros
 - FactorPointers, 465
- firstCount_
 - CoinFactorization, 108
- firstDoRow
 - _EKKfactinfo, 36
- firstFree
 - CoinModelLinkedList, 218
- firstInColumn
 - CoinModel, 198
- firstInQuadraticColumn
 - CoinModel, 199
- firstInRow
 - CoinModel, 198
- firstLRow
 - _EKKfactinfo, 36
- firstNonSlack
 - _EKKfactinfo, 38
- firstNumberSlacks_
 - CoinSimpFactorization, 393
- firstRowInU_
 - CoinSimpFactorization, 389
- firstRowKnonzeros
 - FactorPointers, 465
- fixFullBasis
 - CoinWarmStartBasis, 431
- fixTop
 - CoinSearchTree, 360
 - CoinSearchTreeBase, 362
- fnctptr
 - symrec, 491
- forcing_constraint_action, 466
 - ~forcing_constraint_action, 467
 - forcing_constraint_action, 467
 - forcing_constraint_action, 467
 - name, 467
 - postsolve, 467
 - presolve, 467
- forcing_constraint_action::action, 41
 - bounds, 42
 - nlo, 42
 - nup, 42
 - row, 42
 - rowcols, 42
- format_
 - CoinMessageHandler, 178
- forrestTomlin
 - CoinFactorization, 96
- fp_
 - CoinMessageHandler, 179
- free_list_
 - CoinPostsolveMatrix, 321
- freeAll
 - CoinLpIO, 155
 - CoinMpsIO, 242
- freeFormat
 - CoinMpsCardReader, 223
- freeFormat_
 - CoinMpsCardReader, 226
- freePreviousNames
 - CoinLpIO, 155
- fromCompact
 - CoinMessages, 182
- ftran
 - CoinSimpFactorization, 385
- ftran2
 - CoinSimpFactorization, 386
- ftranAverageAfterL_
 - CoinFactorization, 113
- ftranAverageAfterR_
 - CoinFactorization, 113
- ftranAverageAfterU_

- CoinFactorization, [113](#)
- ftranCountAfterL_
 - CoinFactorization, [112](#)
- ftranCountAfterR_
 - CoinFactorization, [112](#)
- ftranCountAfterU_
 - CoinFactorization, [112](#)
- ftranCountInput_
 - CoinFactorization, [112](#)
- fullBasis
 - CoinWarmStartBasis, [430](#)
- func_t
 - CoinModelUseful.hpp, [520](#)
- g_format_
 - CoinMessageHandler, [179](#)
- g_precision_
 - CoinMessageHandler, [179](#)
- GaussEliminate
 - CoinSimpFactorization, [383](#)
- generateDiff
 - CoinWarmStart, [423](#)
 - CoinWarmStartBasis, [429](#)
 - CoinWarmStartDual, [436](#)
 - CoinWarmStartPrimalDual, [441](#)
 - CoinWarmStartVector, [446](#)
 - CoinWarmStartVectorPair, [451](#)
- getAccuracyCheck
 - CoinFactorization, [95](#)
 - CoinOtherFactorization, [261](#)
- getAreas
 - CoinDenseFactorization, [66](#)
 - CoinFactorization, [101](#)
 - CoinOslFactorization, [253](#)
 - CoinOtherFactorization, [264](#)
 - CoinSimpFactorization, [381](#)
- getArray
 - CoinArrayWithLength, [53](#)
- getArtifStatus
 - CoinWarmStartBasis, [428](#)
- getArtificialStatus
 - CoinWarmStartBasis, [428](#), [429](#)
- getBoundName
 - CoinMpsIO, [237](#)
- getCandidates
 - CoinSearchTreeBase, [362](#)
- getCapacity
 - CoinArrayWithLength, [53](#)
- getClass
 - CoinMessages, [182](#)
- getCoefficient
 - CoinPackedMatrix, [279](#)
- getColIndicesByRow
 - CoinPresolveMatrix, [347](#)
- getColsInteger
 - CoinModel, [201](#)
- getColLengths
 - CoinPrePostsolveMatrix, [331](#)
- getColLower
 - CoinLpIO, [156](#)
 - CoinModel, [201](#)
 - CoinMpsIO, [234](#)
 - CoinPrePostsolveMatrix, [332](#)
 - CoinSnapshot, [397](#)
- getColName
 - CoinModel, [201](#)
- getColNames
 - CoinLpIO, [157](#)
- getColObjective
 - CoinModel, [201](#)
- getColSolution
 - CoinPrePostsolveMatrix, [332](#)
 - CoinSnapshot, [399](#)
- getColStarts
 - CoinPrePostsolveMatrix, [331](#)
- getColType
 - CoinSnapshot, [398](#)
- getColUpper
 - CoinLpIO, [156](#)
 - CoinModel, [201](#)
 - CoinMpsIO, [234](#)
 - CoinPrePostsolveMatrix, [332](#)
 - CoinSnapshot, [397](#)
- getColumn
 - CoinModel, [192](#)
- getColumnBlock
 - CoinBaseModel, [57](#)
 - CoinStructuredModel, [408](#)
- getColumnIsInteger
 - CoinModel, [200](#)
- getColumnIsIntegerAsString
 - CoinModel, [202](#)
- getColumnLower
 - CoinModel, [200](#)
- getColumnLowerAsString
 - CoinModel, [201](#)
- getColumnName
 - CoinModel, [200](#)
- getColumnObjective
 - CoinModel, [200](#)
- getColumnObjectiveAsString
 - CoinModel, [201](#)
- getColumnSpace
 - CoinFactorization, [101](#)
- getColumnSpaceltrate
 - CoinFactorization, [102](#)
- getColumnSpaceltrateR
 - CoinFactorization, [101](#)

- getColumnStatus
 - CoinPrePostsolveMatrix, [329](#)
- getColumnUpper
 - CoinModel, [200](#)
- getColumnUpperAsString
 - CoinModel, [201](#)
- getCommand
 - CoinParam, [312](#)
 - CoinParamUtils, [31](#)
- getCost
 - CoinPrePostsolveMatrix, [332](#)
- getDecimals
 - CoinLpIO, [158](#)
- getDefaultBound
 - CoinMpsIO, [238](#)
- getDepth
 - CoinTreeNode, [415](#)
- getDoNotSeparateThis
 - CoinSnapshot, [399](#)
- getDoubleField
 - CoinParam, [312](#)
 - CoinParamUtils, [30](#)
- getDoubleFromString
 - CoinModel.hpp, [519](#)
- getDualTolerance
 - CoinSnapshot, [399](#)
- getElement
 - CoinModel, [198](#)
- getElementAsString
 - CoinModel, [198](#)
- getElements
 - CoinDenseVector, [71](#)
 - CoinPackedMatrix, [276](#)
 - CoinPackedVector, [293](#), [294](#)
 - CoinPackedVectorBase, [299](#)
 - CoinShallowPackedVector, [373](#)
- getElementsByCol
 - CoinPrePostsolveMatrix, [331](#)
- getElementsByRow
 - CoinPresolveMatrix, [347](#)
- getEpsilon
 - CoinLpIO, [158](#)
- getExtraGap
 - CoinPackedMatrix, [275](#)
- getExtraMajor
 - CoinPackedMatrix, [275](#)
- getFileName
 - CoinFileIOBase, [123](#)
 - CoinMpsIO, [238](#)
- getFractionality
 - CoinTreeNode, [415](#)
- getFunctionValueFromString
 - CoinModel.hpp, [519](#)
- getIndices
 - CoinIndexedVector, [138](#)
 - CoinPackedMatrix, [276](#)
 - CoinPackedVector, [293](#)
 - CoinPackedVectorBase, [299](#)
 - CoinShallowPackedVector, [373](#)
- getInfinity
 - CoinLpIO, [158](#)
 - CoinMpsIO, [237](#)
 - CoinSnapshot, [399](#)
- getIntField
 - CoinParam, [312](#)
 - CoinParamUtils, [30](#)
- getIntegerLowerBound
 - CoinSnapshot, [400](#)
- getIntegerTolerance
 - CoinSnapshot, [400](#)
- getIntegerUpperBound
 - CoinSnapshot, [400](#)
- getMajorDim
 - CoinPackedMatrix, [281](#)
- getMajorIndices
 - CoinPackedMatrix, [277](#)
- getMatrixByCol
 - CoinLpIO, [157](#)
 - CoinMpsIO, [235](#)
 - CoinSnapshot, [398](#)
- getMatrixByRow
 - CoinLpIO, [157](#)
 - CoinMpsIO, [235](#)
 - CoinSnapshot, [398](#)
- getMaxIndex
 - CoinIndexedVector, [143](#)
 - CoinPackedVectorBase, [300](#)
- getMaxMajorDim
 - CoinPackedMatrix, [282](#)
- getMinIndex
 - CoinIndexedVector, [143](#)
 - CoinPackedVectorBase, [300](#)
- getMinorDim
 - CoinPackedMatrix, [282](#)
- getMutableElements
 - CoinPackedMatrix, [285](#)
- getMutableIndices
 - CoinPackedMatrix, [285](#)
- getMutableVectorLengths
 - CoinPackedMatrix, [285](#)
- getMutableVectorStarts
 - CoinPackedMatrix, [285](#)
- getName
 - CoinModelHash, [209](#)
- getNumArtificial
 - CoinWarmStartBasis, [428](#)
- getNumCols
 - CoinLpIO, [155](#)

- CoinMpsIO, 234
- CoinPackedMatrix, 275
- CoinPrePostsolveMatrix, 331
- CoinSnapshot, 397
- getNumElements
 - CoinDenseVector, 71
 - CoinIndexedVector, 138
 - CoinLpIO, 156
 - CoinMpsIO, 234
 - CoinPackedMatrix, 275
 - CoinPackedVector, 293
 - CoinPackedVectorBase, 299
 - CoinPartitionedVector, 316
 - CoinShallowPackedVector, 373
 - CoinSnapshot, 397
- getNumElems
 - CoinPrePostsolveMatrix, 331
- getNumIntegers
 - CoinSnapshot, 397
- getNumPartitions
 - CoinPartitionedVector, 316
- getNumRows
 - CoinLpIO, 155
 - CoinMpsIO, 234
 - CoinPackedMatrix, 275
 - CoinPrePostsolveMatrix, 331
 - CoinSnapshot, 397
- getNumStructural
 - CoinWarmStartBasis, 428
- getNumberAcross
 - CoinLpIO, 158
- getObjCoefficients
 - CoinLpIO, 157
 - CoinMpsIO, 235
 - CoinSnapshot, 397
- getObjName
 - CoinLpIO, 157
- getObjOffset
 - CoinSnapshot, 399
- getObjSense
 - CoinSnapshot, 398
- getObjValue
 - CoinSnapshot, 399
- getObjectiveName
 - CoinMpsIO, 236
- getOriginalMatrixByCol
 - CoinSnapshot, 398
- getOriginalMatrixByRow
 - CoinSnapshot, 398
- getOriginalPosition
 - CoinPackedVector, 294
- getPosition
 - CoinMpsCardReader, 225
- getPreferred
 - CoinTreeNode, 415
- getPreviousColNames
 - CoinLpIO, 157
- getPreviousRowNames
 - CoinLpIO, 157
- getPrimalTolerance
 - CoinSnapshot, 399
- getProblemName
 - CoinBaseModel, 57
 - CoinLpIO, 155
 - CoinMpsIO, 236
- getQuadraticElement
 - CoinModel, 198
- getQuality
 - CoinTreeNode, 415
- getRangeName
 - CoinMpsIO, 236
- GetRawPtr
 - Coin::SmartPtr, 487
- getReadType
 - CoinFileIOBase, 123
- getReducedCost
 - CoinPrePostsolveMatrix, 332
 - CoinSnapshot, 399
- getRhsName
 - CoinMpsIO, 236
- getRightHandSide
 - CoinLpIO, 156
 - CoinMpsIO, 235
 - CoinSnapshot, 397
- getRow
 - CoinModel, 192
- getRowActivity
 - CoinPrePostsolveMatrix, 332
 - CoinSnapshot, 399
- getRowBlock
 - CoinBaseModel, 57
 - CoinStructuredModel, 408
- getRowIndicesByCol
 - CoinPrePostsolveMatrix, 331
- getRowLower
 - CoinLpIO, 156
 - CoinModel, 199
 - CoinMpsIO, 235
 - CoinPrePostsolveMatrix, 332
 - CoinSnapshot, 397
- getRowLowerAsString
 - CoinModel, 201
- getRowName
 - CoinModel, 199
- getRowNames
 - CoinLpIO, 157
- getRowPrice
 - CoinPrePostsolveMatrix, 332

- CoinSnapshot, 399
- getRowRange
 - CoinLpIO, 156
 - CoinMpsIO, 235
- getRowSense
 - CoinLpIO, 156
 - CoinMpsIO, 234
- getRowSpace
 - CoinFactorization, 102
- getRowSpacelterate
 - CoinFactorization, 102
- getRowStarts
 - CoinPresolveMatrix, 347
- getRowStatus
 - CoinPrePostsolveMatrix, 329
- getRowUpper
 - CoinLpIO, 156
 - CoinModel, 199
 - CoinMpsIO, 235
 - CoinPrePostsolveMatrix, 332
 - CoinSnapshot, 397
- getRowUpperAsString
 - CoinModel, 201
- getSeed
 - CoinThreadRandom, 411
- getSize
 - CoinArbitraryArrayWithLength, 48
 - CoinArrayWithLength, 52
 - CoinBigIndexArrayWithLength, 60
 - CoinDoubleArrayWithLength, 75
 - CoinFactorizationDoubleArrayWithLength, 116
 - CoinFactorizationLongDoubleArrayWithLength, 118
 - CoinIntArrayWithLength, 147
 - CoinUnsignedIntArrayWithLength, 420
 - CoinVoidStarArrayWithLength, 422
- getSizeVectorLengths
 - CoinPackedMatrix, 276
- getSizeVectorStarts
 - CoinPackedMatrix, 276
- getSmallElementValue
 - CoinMpsIO, 238
- getStatus
 - CoinPrePostsolveMatrix, 330
 - CoinWarmStartBasis, 431
- getStringField
 - CoinParam, 311
 - CoinParamUtils, 30
- getStructStatus
 - CoinWarmStartBasis, 428
- getStructuralStatus
 - CoinWarmStartBasis, 428
- getTree
 - CoinSearchTreeManager, 367
- getTrueLB
 - CoinTreeNode, 415
- getVector
 - CoinPackedMatrix, 277
- getVectorElements
 - CoinPackedVector, 294
- getVectorFirst
 - CoinPackedMatrix, 276
- getVectorIndices
 - CoinPackedVector, 294
- getVectorLast
 - CoinPackedMatrix, 276
- getVectorLengths
 - CoinPackedMatrix, 276
- getVectorNumElements
 - CoinPackedVector, 294
- getVectorSize
 - CoinPackedMatrix, 277
- getVectorStarts
 - CoinPackedMatrix, 276
- gets
 - CoinFileInput, 121
- goSparse
 - CoinFactorization, 99
- gubrow_action, 467
 - name, 468
 - postsolve, 468
 - presolve, 468
- gutsOfCopy
 - CoinDenseFactorization, 68
 - CoinFactorization, 100
 - CoinLpIO, 155
 - CoinMpsIO, 242
 - CoinOslFactorization, 256
 - CoinSimpFactorization, 383
 - CoinWarmStartVector, 445
- gutsOfCopyOf
 - CoinPackedMatrix, 286
- gutsOfCopyOfNoGaps
 - CoinPackedMatrix, 286
- gutsOfDestructor
 - CoinDenseFactorization, 68
 - CoinFactorization, 100
 - CoinLpIO, 155
 - CoinMpsIO, 242
 - CoinOslFactorization, 256
 - CoinPackedMatrix, 286
 - CoinSimpFactorization, 383
 - CoinWarmStartVector, 445
- gutsOfInitialize
 - CoinDenseFactorization, 68
 - CoinFactorization, 100
 - CoinOslFactorization, 256
 - CoinSimpFactorization, 383
- gutsOfOpEqual

- CoinPackedMatrix, [286](#)
- handler_
 - CoinLpIO, [164](#)
 - CoinMpsCardReader, [227](#)
 - CoinMpsIO, [246](#)
 - CoinPrePostsolveMatrix, [336](#)
- hasGaps
 - CoinPackedMatrix, [275](#)
- hash
 - CoinModelHash, [209](#)
 - CoinModelHash2, [211](#)
- hash_
 - CoinLpIO, [167](#)
 - CoinMpsIO, [245](#)
- haveBzip2Support
 - CoinFileInput, [120](#)
- haveGzipSupport
 - CoinFileInput, [120](#)
- hcol_
 - CoinPresolveMatrix, [353](#)
- highestNumber
 - CoinMessageHandler, [175](#)
- highestNumber_
 - CoinMessageHandler, [179](#)
- hincol_
 - CoinPrePostsolveMatrix, [334](#)
- hinrow_
 - CoinPresolveMatrix, [352](#)
- hpivcoR
 - _EKKfactinfo, [36](#)
- hrow_
 - CoinPrePostsolveMatrix, [334](#)
- Hxeb
 - CoinSimpFactorization, [385](#)
- Hxeb2
 - CoinSimpFactorization, [385](#)
- IMPLIED_BOUND
 - CoinPresolveForcing.hpp, [536](#)
- IMPLIED_FREE
 - CoinPresolveImpliedFree.hpp, [537](#)
- icolx
 - doubleton_action::action, [40](#)
 - tripleton_action::action, [43](#)
- icoly
 - doubleton_action::action, [40](#)
 - tripleton_action::action, [43](#)
- icolz
 - tripleton_action::action, [43](#)
- ieeeFormat_
 - CoinMpsCardReader, [226](#)
- if_sparse_update
 - _EKKfactinfo, [38](#)
- ifvsol
 - _EKKfactinfo, [38](#)
- implied_bounds
 - CoinPresolveSubst.hpp, [541](#)
- implied_free_action, [468](#)
 - ~implied_free_action, [469](#)
 - name, [469](#)
 - postsolve, [469](#)
 - presolve, [469](#)
- increaseColSize
 - CoinSimpFactorization, [384](#)
- increaseLsize
 - CoinSimpFactorization, [384](#)
- increaseRowSize
 - CoinSimpFactorization, [384](#)
- indKeep_
 - CoinSimpFactorization, [387](#)
- indVector_
 - CoinSimpFactorization, [386](#)
- index
 - CoinLpIO::CoinHashLink, [131](#)
 - CoinModelHashLink, [212](#)
 - CoinMpsIO::CoinHashLink, [132](#)
- index_
 - CoinPackedMatrix, [287](#)
- indexColumnL
 - CoinFactorization, [94](#)
- indexColumnL_
 - CoinFactorization, [114](#)
- indexColumnU_
 - CoinFactorization, [109](#)
- indexRowL
 - CoinFactorization, [94](#)
- indexRowL_
 - CoinFactorization, [110](#)
- indexRowR_
 - CoinFactorization, [111](#)
- indexRowU
 - CoinFactorization, [98](#)
- indexRowU_
 - CoinFactorization, [110](#)
- indexSet
 - CoinPackedVectorBase, [301](#)
- indices
 - CoinDenseFactorization, [68](#)
 - CoinOslFactorization, [256](#)
 - CoinOtherFactorization, [263](#)
 - CoinSimpFactorization, [382](#)
- indices_
 - CoinIndexedVector, [145](#)
- infNorm
 - CoinDenseVector, [72](#)
 - CoinPackedVectorBase, [301](#)
- infiniteDown_
 - CoinPresolveMatrix, [356](#)

- infiniteUp_
 - CoinPresolveMatrix, 356
- infinity_
 - CoinLpIO, 166
 - CoinMpsIO, 246
- initColsToDo
 - CoinPresolveMatrix, 349
- initRowsToDo
 - CoinPresolveMatrix, 350
- initialSomeNumbers
 - CoinSimpFactorization, 385
- initializeStuff
 - CoinPresolveMatrix, 348
- input_
 - CoinMpsCardReader, 226
- insert
 - CoinIndexedVector, 140
 - CoinPackedVector, 295
- insertHash
 - CoinLpIO, 162
- intVal
 - CoinParam, 309
- intValue
 - CoinMessageHandler, 174
- intWorkArea
 - CoinOslFactorization, 254
 - CoinOtherFactorization, 262
- integer
 - CoinModelInfo2, 213
- integerColumns
 - CoinLpIO, 158
 - CoinMpsIO, 236
- integerType_
 - CoinLpIO, 166
 - CoinMpsIO, 245
 - CoinPresolveMatrix, 353
- integerTypeArray
 - CoinModel, 204
- internalNumber_
 - CoinMessageHandler, 178
- invOfPivots_
 - CoinSimpFactorization, 390
- invok
 - _EKKfactinfo, 37
- isFree
 - CoinPrePostsolveMatrix, 328
 - CoinWarmStartBasis, 427
- is_comment
 - CoinLpIO, 162
- is_free
 - CoinLpIO, 162
- is_inf
 - CoinLpIO, 163
- is_invalid_name
 - CoinLpIO, 159
- is_keyword
 - CoinLpIO, 163
- is_sense
 - CoinLpIO, 163
- is_subject_to
 - CoinLpIO, 162
- isApproximatelyEqual
 - CoinIndexedVector, 143
- isBinary
 - CoinSnapshot, 398
- isColOrdered
 - CoinPackedMatrix, 275
- isCommandLine
 - CoinParam, 311
 - CoinParamUtils, 30
- isContinuous
 - CoinMpsIO, 235
 - CoinSnapshot, 398
- isEquivalent
 - CoinPackedMatrix, 284
 - CoinPackedVectorBase, 301
- isEquivalent2
 - CoinPackedMatrix, 284
- isExistingIndex
 - CoinPackedVectorBase, 300
- isExpired
 - CoinTimer, 413
- isFreeBinary
 - CoinSnapshot, 398
- isInteger
 - CoinLpIO, 158
 - CoinModel, 200
 - CoinMpsIO, 235
 - CoinPresolveMatrix, 347
 - CoinSnapshot, 398
- isIntegerAsString
 - CoinModel, 202
- isIntegerNonBinary
 - CoinSnapshot, 398
- isInteractive
 - CoinParam, 311
 - CoinParamUtils, 30
- IsNull
 - Coin::SmartPtr, 487
- isPast
 - CoinTimer, 413
- isPastPercent
 - CoinTimer, 413
- IsValid
 - Coin::SmartPtr, 487
- isolated_constraint_action, 470
 - ~isolated_constraint_action, 470
 - name, 470

- postsolve, [470](#)
 - presolve, [470](#)
- it
 - CoinMessages, [181](#)
- iter0
 - _EKKfactinfo, [37](#)
- iterin
 - _EKKfactinfo, [37](#)
- iterno
 - _EKKfactinfo, [37](#)
- ivec
 - Coin_C_defines.h, [497](#)
- kadrpm
 - _EKKfactinfo, [35](#)
- kcpadr
 - _EKKfactinfo, [35](#)
- keepSize_
 - CoinSimpFactorization, [387](#)
- kmxeta
 - _EKKfactinfo, [37](#)
- kp1adr
 - _EKKfactinfo, [36](#)
- kp2adr
 - _EKKfactinfo, [36](#)
- krpadr
 - _EKKfactinfo, [35](#)
- kw1adr
 - _EKKfactinfo, [36](#)
- kw2adr
 - _EKKfactinfo, [36](#)
- kw3adr
 - _EKKfactinfo, [36](#)
- kwdIndex
 - CoinParam, [308](#)
- kwdVal
 - CoinParam, [308](#)
- LUUpdate
 - CoinSimpFactorization, [385](#)
- Language
 - CoinMessages, [181](#)
- language
 - CoinMessages, [181](#)
- language_
 - CoinMessages, [182](#)
- last
 - CoinModelLinkedList, [219](#)
- last_dense
 - _EKKfactinfo, [37](#)
- last_eta_size
 - _EKKfactinfo, [39](#)
- lastCollnU_
 - CoinSimpFactorization, [390](#)
- lastColumn_
 - CoinFactorization, [108](#)
- lastCount_
 - CoinFactorization, [108](#)
- lastEtaCount
 - _EKKfactinfo, [38](#)
- lastEtaRow_
 - CoinSimpFactorization, [391](#)
- lastFree
 - CoinModelLinkedList, [218](#)
- lastInColumn
 - CoinModel, [199](#)
- lastInQuadraticColumn
 - CoinModel, [199](#)
- lastInRow
 - CoinModel, [198](#)
- lastRow_
 - CoinFactorization, [108](#)
- lastRowInU_
 - CoinSimpFactorization, [389](#)
- lastSlack
 - _EKKfactinfo, [38](#)
- LcolCap_
 - CoinSimpFactorization, [388](#)
- LcolInd_
 - CoinSimpFactorization, [388](#)
- LcolLengths_
 - CoinSimpFactorization, [388](#)
- LcolSize_
 - CoinSimpFactorization, [388](#)
- LcolStarts_
 - CoinSimpFactorization, [387](#)
- Lcolumns_
 - CoinSimpFactorization, [388](#)
- length
 - CoinYacc, [453](#)
- length_
 - CoinPackedMatrix, [288](#)
- lengthAreaL
 - CoinFactorization, [97](#)
- lengthAreaL_
 - CoinFactorization, [110](#)
- lengthAreaR_
 - CoinFactorization, [111](#)
- lengthAreaU
 - CoinFactorization, [97](#)
- lengthAreaU_
 - CoinFactorization, [109](#)
- lengthInBytes_
 - CoinArbitraryArrayWithLength, [49](#)
- lengthL_
 - CoinFactorization, [110](#)
- lengthMessages_
 - CoinMessages, [182](#)
- lengthR_
 - CoinFactorization, [108](#)

- CoinFactorization, 111
- lengthU_
 - CoinFactorization, 109
- lineNumber
 - CoinError, 77
- link_
 - CoinPostsolveMatrix, 321
- loadBlock
 - CoinModel, 205, 206
- loadProblem
 - CoinSnapshot, 400
- logLevel
 - CoinBaseModel, 57
 - CoinMessageHandler, 173, 174
- logLevel_
 - CoinBaseModel, 58
 - CoinMessageHandler, 178
- logLevels_
 - CoinMessageHandler, 178
- longHelp
 - CoinParam, 310
- longValue_
 - CoinMessageHandler, 177
- lookupParam
 - CoinParam, 312
 - CoinParamUtils, 31
- LrowCap_
 - CoinSimpFactorization, 387
- LrowInd_
 - CoinSimpFactorization, 387
- LrowLengths_
 - CoinSimpFactorization, 387
- LrowSize_
 - CoinSimpFactorization, 387
- LrowStarts_
 - CoinSimpFactorization, 387
- Lrows_
 - CoinSimpFactorization, 387
- Istart
 - _EKKfactinfo, 38
- Lxeqb
 - CoinSimpFactorization, 385
- Lxeqb2
 - CoinSimpFactorization, 385
- MAX_CARD_LENGTH
 - CoinMpsIO.hpp, 521
- mainLoopFactor
 - CoinSimpFactorization, 383
- majorAppendOrthoOrdered
 - CoinPackedMatrix, 283
- majorAppendSameOrdered
 - CoinPackedMatrix, 283
- majorDim_
 - CoinPackedMatrix, 288
- make_fixed
 - make_fixed_action, 472
- make_fixed_action, 471
 - ~make_fixed_action, 472
 - make_fixed, 472
 - name, 472
 - postsolve, 472
 - presolve, 472
 - transferCosts, 472
- makeNonSingular
 - CoinDenseFactorization, 67
 - CoinOslFactorization, 253
 - CoinOtherFactorization, 264
 - CoinSimpFactorization, 381
- markRow_
 - CoinFactorization, 108
- matchName
 - CoinParam, 310
- matchParam
 - CoinParam, 312
 - CoinParamUtils, 30
- matches
 - CoinParam, 310
- matrix
 - CoinModelInfo2, 213
- matrixByColumn_
 - CoinLpIO, 165
 - CoinMpsIO, 244
- matrixByRow_
 - CoinLpIO, 165
 - CoinMpsIO, 244
- maxA_
 - CoinSimpFactorization, 392
- maxEtaRows_
 - CoinSimpFactorization, 392
- maxGrowth_
 - CoinSimpFactorization, 392
- maxHash_
 - CoinLpIO, 167
- maxMajorDim_
 - CoinPackedMatrix, 288
- maxNNetas
 - _EKKfactinfo, 39
- maxSize_
 - CoinPackedMatrix, 288
 - CoinWarmStartBasis, 431
- maxSubstLevel_
 - CoinPresolveMatrix, 353
- maxU_
 - CoinSimpFactorization, 392
- maximumCoefficient
 - CoinDenseFactorization, 67
 - CoinFactorization, 96

- CoinOslFactorization, [255](#)
- CoinSimpFactorization, [381](#)
- maximumColumnsExtra
 - CoinFactorization, [98](#)
- maximumColumnsExtra_
 - CoinFactorization, [106](#)
- maximumElements
 - CoinModelLinkedList, [218](#)
- maximumItems
 - CoinModelHash, [209](#)
 - CoinModelHash2, [211](#)
- maximumMajor
 - CoinModelLinkedList, [218](#)
- maximumPivots
 - CoinFactorization, [96](#)
 - CoinOslFactorization, [255](#)
 - CoinOtherFactorization, [261](#)
- maximumPivots_
 - CoinFactorization, [106](#)
 - CoinOtherFactorization, [266](#)
- maximumRows_
 - CoinOtherFactorization, [266](#)
- maximumRowsExtra
 - CoinFactorization, [95](#)
- maximumRowsExtra_
 - CoinFactorization, [106](#)
- maximumSpace_
 - CoinOtherFactorization, [266](#)
- maximumStringElements_
 - CoinMpsIO, [246](#)
- maximumU_
 - CoinFactorization, [109](#)
- maxinv
 - _EKKfactinfo, [36](#)
- maxlink_
 - CoinPostsolveMatrix, [321](#)
- maxmin_
 - CoinPrePostsolveMatrix, [335](#)
- mcstrt_
 - CoinPrePostsolveMatrix, [334](#)
- mergeBasis
 - CoinWarmStartBasis, [430](#)
- message
 - CoinError, [77](#)
 - CoinMessageHandler, [176](#)
 - CoinOneMessage, [249](#)
- message_
 - CoinMessages, [183](#)
 - CoinOneMessage, [250](#)
- messageBuffer
 - CoinMessageHandler, [175](#)
- messageBuffer_
 - CoinMessageHandler, [178](#)
- messageHandler
 - CoinLpIO, [161](#)
 - CoinMpsIO, [241](#)
 - CoinPrePostsolveMatrix, [333](#)
- messageLevel
 - CoinFactorization, [96](#)
- messageLevel_
 - CoinFactorization, [109](#)
- messageOut_
 - CoinMessageHandler, [178](#)
- messages
 - CoinLpIO, [161](#)
 - CoinMpsIO, [241](#)
 - CoinPrePostsolveMatrix, [333](#)
- messages_
 - CoinLpIO, [164](#)
 - CoinMpsCardReader, [227](#)
 - CoinMpsIO, [246](#)
 - CoinPrePostsolveMatrix, [336](#)
- messagesPointer
 - CoinLpIO, [161](#)
 - CoinMpsIO, [241](#)
- methodName
 - CoinError, [77](#)
- minIncrease_
 - CoinSimpFactorization, [392](#)
- minorAppendOrthoOrdered
 - CoinPackedMatrix, [283](#)
- minorAppendSameOrdered
 - CoinPackedMatrix, [283](#)
- minorDim_
 - CoinPackedMatrix, [288](#)
- modifyCoefficient
 - CoinPackedMatrix, [279](#)
- moreInfo
 - CoinModel, [204](#)
- mpermu
 - _EKKfactinfo, [35](#)
- mpsType
 - CoinMpsCardReader, [224](#)
- mpsType_
 - CoinMpsCardReader, [226](#)
- mrstrt_
 - CoinPresolveMatrix, [352](#)
- msgno
 - Coin_C_defines.h, [496](#)
- mutableCard
 - CoinMpsCardReader, [224](#)
- nElements_
 - CoinIndexedVector, [145](#)
- NO_LINK
 - CoinPresolveMatrix.hpp, [539](#)
- NO_SHIFT
 - CoinOslC.h, [525](#)

NOT_ZERO
 CoinOsIC.h, 526
 nR_etas
 _EKKfactinfo, 38
 nactions_
 doubleton_action, 456
 remove_fixed_action, 480
 tripleton_action, 492
 name
 CoinModelHash, 209
 CoinParam, 310
 CoinPresolveAction, 339
 CoinSearchTreeCompareBest, 364
 CoinSearchTreeCompareBreadth, 364
 CoinSearchTreeCompareDepth, 365
 CoinSearchTreeComparePreferred, 366
 do_tighten_action, 454
 doubleton_action, 456
 drop_empty_cols_action, 458
 drop_empty_rows_action, 459
 drop_zero_coefficients_action, 460
 dupcol_action, 462
 duprow_action, 463
 forcing_constraint_action, 467
 gubrow_action, 468
 implied_free_action, 469
 isolated_constraint_action, 470
 make_fixed_action, 472
 remove_dual_action, 478
 remove_fixed_action, 480
 slack_doubleton_action, 482
 slack_singleton_action, 483
 subst_constraint_action, 489
 symrec, 490
 tripleton_action, 492
 twoxtwo_action, 493
 useless_constraint_action, 495
 names
 CoinModelHash, 209
 names_
 CoinLpIO, 167
 CoinMpsIO, 245
 nbfinv
 _EKKfactinfo, 37
 nchar
 Coin_C_defines.h, 497
 ncols0_
 CoinPrePostsolveMatrix, 333
 ncols_
 CoinPrePostsolveMatrix, 333
 ncolx
 doubleton_action::action, 40
 tripleton_action::action, 44
 ncoly
 doubleton_action::action, 40
 tripleton_action::action, 44
 ndenuc
 _EKKfactinfo, 37
 ndouble
 Coin_C_defines.h, 496
 nelems0_
 CoinPrePostsolveMatrix, 334
 nelems_
 CoinPrePostsolveMatrix, 333
 newCols
 FactorPointers, 466
 newEta
 CoinSimpFactorization, 385
 newLanguage
 CoinLpIO, 161
 CoinMpsIO, 241
 newSolution
 CoinSearchTreeManager, 368
 next
 CoinLpIO::CoinHashLink, 131
 CoinModel, 199
 CoinModelHashLink, 212
 CoinModelLinkedList, 219
 CoinMpsIO::CoinHashLink, 132
 CoinPresolveAction, 340
 symrec, 491
 nextBlankOr
 CoinMpsCardReader, 225
 nextCollnU_
 CoinSimpFactorization, 390
 nextColsToDo_
 CoinPresolveMatrix, 354
 nextColumn
 FactorPointers, 465
 nextColumn_
 CoinFactorization, 108
 nextCount_
 CoinFactorization, 108
 nextField
 CoinMpsCardReader, 223
 nextGmsField
 CoinMpsCardReader, 223
 nextRow
 FactorPointers, 465
 nextRow_
 CoinFactorization, 108
 nextRowInU_
 CoinSimpFactorization, 389
 nextRowsToDo_
 CoinPresolveMatrix, 355
 nint
 Coin_C_defines.h, 496
 nlo

- forcing_constraint_action::action, 42
- nnentl
 - _EKKfactinfo, 37
- nnentu
 - _EKKfactinfo, 37
- nnetas
 - _EKKfactinfo, 36
- nonzero
 - _EKKfactinfo, 35
- normSquare
 - CoinPackedVectorBase, 301
- npivots
 - _EKKfactinfo, 37
- nrow
 - _EKKfactinfo, 36
- nrowmx
 - _EKKfactinfo, 36
- nrows0_
 - CoinPrePostsolveMatrix, 333
- nrows_
 - CoinPrePostsolveMatrix, 333
- nullElementArray
 - CoinPackedMatrix, 285
- nullIndexArray
 - CoinPackedMatrix, 286
- nullLengthArray
 - CoinPackedMatrix, 285
- nullStartArray
 - CoinPackedMatrix, 285
- num_resets
 - _EKKfactinfo, 37
- numArtificial_
 - CoinWarmStartBasis, 431
- numInserted
 - CoinSearchTreeBase, 362
 - CoinSearchTreeManager, 367
- numInserted_
 - CoinSearchTreeBase, 363
- numStructural_
 - CoinWarmStartBasis, 431
- numberAcross_
 - CoinLpIO, 166
- numberBasicStructurals
 - CoinWarmStartBasis, 428
- numberBtranCounts_
 - CoinFactorization, 113
- numberCharFields
 - CoinMessageHandler, 175
- numberColsToDo
 - CoinPresolveMatrix, 349
- numberColsToDo_
 - CoinPresolveMatrix, 354
- numberColumnBlocks
 - CoinStructuredModel, 407
- numberColumns
 - CoinBaseModel, 56
 - CoinBuild, 63
 - CoinFactorization, 95
 - CoinOtherFactorization, 261
- numberColumns_
 - CoinBaseModel, 58
 - CoinFactorization, 106
 - CoinLpIO, 164
 - CoinMpsIO, 244
 - CoinOtherFactorization, 265
- numberColumnsExtra_
 - CoinFactorization, 106
- numberCompressions
 - CoinFactorization, 97
- numberCompressions_
 - CoinFactorization, 112
- numberDense
 - CoinFactorization, 97
- numberDense_
 - CoinFactorization, 112
- numberDoubleFields
 - CoinMessageHandler, 174
- numberElementBlocks
 - CoinStructuredModel, 408
- numberElements
 - CoinBaseModel, 56
 - CoinBuild, 63
 - CoinDenseFactorization, 67
 - CoinFactorization, 95
 - CoinModel, 197
 - CoinModelLinkedList, 218
 - CoinOslFactorization, 253
 - CoinOtherFactorization, 263
 - CoinSimpFactorization, 381
 - CoinStructuredModel, 408
- numberElements_
 - CoinLpIO, 165
 - CoinMpsIO, 244
- numberElementsL
 - CoinFactorization, 97
- numberElementsPartition_
 - CoinPartitionedVector, 318
- numberElementsR
 - CoinFactorization, 97
- numberElementsU
 - CoinFactorization, 97
- numberEntries
 - CoinSet, 369
- numberEntries_
 - CoinSet, 370
- numberForrestTomlin
 - CoinFactorization, 95
- numberFtranCounts_

- CoinFactorization, [113](#)
- numberGoodColumns
 - CoinFactorization, [95](#)
 - CoinOtherFactorization, [261](#)
- numberGoodL_
 - CoinFactorization, [106](#)
- numberGoodU_
 - CoinFactorization, [106](#)
 - CoinOtherFactorization, [265](#)
- numberHash_
 - CoinLpIO, [167](#)
 - CoinMpsIO, [245](#)
- numberInColumn
 - CoinFactorization, [98](#)
 - CoinOslFactorization, [254](#)
 - CoinOtherFactorization, [262](#)
- numberInColumn_
 - CoinFactorization, [107](#)
- numberInColumnPlus_
 - CoinFactorization, [108](#)
- numberInRow
 - CoinFactorization, [98](#)
 - CoinOslFactorization, [254](#)
 - CoinOtherFactorization, [262](#)
- numberInRow_
 - CoinFactorization, [107](#)
- numberIntFields
 - CoinMessageHandler, [175](#)
- numberItems
 - CoinModelHash, [209](#)
 - CoinModelHash2, [211](#)
- numberL
 - CoinFactorization, [94](#)
- numberL_
 - CoinFactorization, [110](#)
- numberMajor
 - CoinModelLinkedList, [218](#)
- numberMessages_
 - CoinMessages, [182](#)
- numberNextColsToDo_
 - CoinPresolveMatrix, [354](#)
- numberNextRowsToDo_
 - CoinPresolveMatrix, [355](#)
- numberPartitions_
 - CoinPartitionedVector, [318](#)
- numberPivots_
 - CoinFactorization, [106](#)
 - CoinOtherFactorization, [266](#)
- numberR_
 - CoinFactorization, [111](#)
- numberRowBlocks
 - CoinStructuredModel, [407](#)
- numberRows
 - CoinBaseModel, [56](#)
- CoinBuild, [63](#)
- CoinFactorization, [94](#)
- CoinOtherFactorization, [261](#)
- numberRows_
 - CoinBaseModel, [58](#)
 - CoinFactorization, [106](#)
 - CoinLpIO, [164](#)
 - CoinMpsIO, [244](#)
 - CoinOtherFactorization, [265](#)
- numberRowsExtra
 - CoinFactorization, [94](#)
- numberRowsExtra_
 - CoinFactorization, [106](#)
- numberRowsToDo
 - CoinPresolveMatrix, [350](#)
- numberRowsToDo_
 - CoinPresolveMatrix, [355](#)
- numberSlacks
 - _EKKfactinfo, [38](#)
- numberSlacks_
 - CoinFactorization, [109](#)
 - CoinSimpFactorization, [392](#)
- numberStringElements
 - CoinMpsIO, [237](#)
- numberStringElements_
 - CoinMpsIO, [247](#)
- numberStringFields
 - CoinMessageHandler, [175](#)
- numberTrials_
 - CoinFactorization, [107](#)
- numberU_
 - CoinFactorization, [109](#)
- nup
 - forcing_constraint_action::action, [42](#)
- nuspike
 - _EKKfactinfo, [38](#)
- objName_
 - CoinLpIO, [166](#)
- objective
 - CoinModel, [200](#)
- objective_
 - CoinLpIO, [166](#)
 - CoinMpsIO, [245](#)
- objectiveArray
 - CoinModel, [204](#)
- objectiveAsString
 - CoinModel, [202](#)
- objectiveName_
 - CoinMpsIO, [243](#)
- objectiveOffset
 - CoinBaseModel, [56](#)
 - CoinLpIO, [158](#)
 - CoinMpsIO, [236](#)

objectiveOffset_
 CoinBaseModel, 58
 CoinLpIO, 166
 CoinMpsIO, 245

offset_
 CoinArrayWithLength, 54
 CoinIndexedVector, 145

onRow
 CoinModelLink, 215

oneNorm
 CoinDenseVector, 72
 CoinPackedVectorBase, 301

operator<
 BitVector128, 45
 CoinSearchTree.hpp, 544

operator<<
 CoinMessageHandler, 176, 177
 CoinParam, 311
 CoinParam.hpp, 534

operator*
 Coin::SmartPtr, 487
 CoinDenseVector.hpp, 500, 501
 CoinIndexedVector, 143
 CoinPackedVector.hpp, 531, 532

operator*=
 CoinDenseVector, 73
 CoinIndexedVector, 142, 144
 CoinPackedVector, 295

operator()
 CoinAbsFltEq, 46
 CoinExternalVectorFirstGreater_2, 79
 CoinExternalVectorFirstGreater_3, 79
 CoinExternalVectorFirstLess_2, 80
 CoinExternalVectorFirstLess_3, 81
 CoinFirstAbsGreater_2, 126
 CoinFirstAbsGreater_3, 126
 CoinFirstAbsLess_2, 127
 CoinFirstAbsLess_3, 128
 CoinFirstGreater_2, 128
 CoinFirstGreater_3, 129
 CoinFirstLess_2, 130
 CoinFirstLess_3, 130
 CoinModel, 192, 198
 CoinRelFltEq, 359
 CoinSearchTreeCompareBest, 364
 CoinSearchTreeCompareBreadth, 364
 CoinSearchTreeCompareDepth, 365
 CoinSearchTreeComparePreferred, 366

operator+
 CoinDenseVector.hpp, 499, 500
 CoinIndexedVector, 143
 CoinPackedVector.hpp, 530–532

operator+=
 CoinDenseVector, 73
 CoinIndexedVector, 142, 144
 CoinPackedVector, 295

operator->
 Coin::SmartPtr, 487

operator=
 CoinDenseVector, 73
 CoinIndexedVector, 142, 144
 CoinPackedVector, 295

operator/
 CoinDenseVector.hpp, 500, 501
 CoinIndexedVector, 144
 CoinPackedVector.hpp, 531, 532

operator/=
 CoinDenseVector, 73
 CoinIndexedVector, 142, 144
 CoinPackedVector, 295

operator=
 Coin::SmartPtr, 487
 CoinAbsFltEq, 46
 CoinArbitraryArrayWithLength, 48
 CoinArrayWithLength, 53
 CoinBaseModel, 56
 CoinBigIndexArrayWithLength, 60
 CoinBuild, 64
 CoinDenseFactorization, 66
 CoinDenseVector, 72
 CoinDoubleArrayWithLength, 75
 CoinError, 77
 CoinFactorization, 92
 CoinFactorizationDoubleArrayWithLength, 116
 CoinFactorizationLongDoubleArrayWithLength, 118
 CoinIndexedVector, 139
 CoinIntArrayWithLength, 147
 CoinLpIO, 155
 CoinMessageHandler, 173
 CoinMessages, 181
 CoinModel, 206
 CoinModelHash, 208
 CoinModelHash2, 211
 CoinModelLink, 215
 CoinModelLinkedList, 218
 CoinMpsIO, 241
 CoinOneMessage, 248
 CoinOslFactorization, 253
 CoinOtherFactorization, 260
 CoinPackedMatrix, 280
 CoinPackedVector, 294
 CoinParam, 308
 CoinPartitionedVector, 318
 CoinRelFltEq, 359

- CoinSet, [369](#)
- CoinShallowPackedVector, [373](#)
- CoinSimpFactorization, [380](#)
- CoinSnapshot, [403](#)
- CoinStructuredModel, [409](#)
- CoinThreadRandom, [411](#)
- CoinTreeNode, [415](#)
- CoinUnsignedIntArrayWithLength, [420](#)
- CoinVoidStarArrayWithLength, [422](#)
- CoinWarmStartBasis, [430](#)
- CoinWarmStartBasisDiff, [433](#)
- CoinWarmStartDual, [436](#)
- CoinWarmStartDualDiff, [438](#)
- CoinWarmStartPrimalDual, [441](#)
- CoinWarmStartVector, [446](#)
- CoinWarmStartVectorDiff, [448](#)
- CoinWarmStartVectorPair, [450](#)
- CoinWarmStartVectorPairDiff, [452](#)
- operator==
 - Coin::SmartPtr, [488](#)
 - CoinIndexedVector, [142](#), [143](#)
 - CoinPackedVectorBase, [300](#)
 - CoinSmartPtr.hpp, [546](#), [547](#)
- optimizationDirection
 - CoinBaseModel, [57](#)
 - CoinModel, [204](#)
 - CoinStructuredModel, [409](#)
- optimizationDirection_
 - CoinBaseModel, [58](#)
- orderMatrix
 - CoinPackedMatrix, [279](#)
- originalColumn_
 - CoinPrePostsolveMatrix, [335](#)
- originalColumns
 - CoinModel, [197](#)
- originalOffset_
 - CoinPrePostsolveMatrix, [334](#)
- originalRow_
 - CoinPrePostsolveMatrix, [335](#)
- originalRows
 - CoinModel, [197](#)
- osi_strtod
 - CoinMpsCardReader, [225](#)
- out_coeff
 - CoinLpIO, [162](#)
- PRESOLVE_INF
 - CoinPresolveMatrix.hpp, [539](#)
- PRESOLVE_INSERT_LINK
 - presolvehlink, [473](#)
- PRESOLVE_MOVE_LINK
 - presolvehlink, [474](#)
- PRESOLVE_REMOVE_LINK
 - presolvehlink, [473](#)
- PRESOLVE_SMALL_INF
 - CoinPresolveMatrix.hpp, [539](#)
- PRESOLVE_STMT
 - CoinPresolveMatrix.hpp, [539](#)
- PRESOLVEASSERT
 - CoinPresolveMatrix.hpp, [538](#)
- PRESOLVEFINITE
 - CoinPresolveMatrix.hpp, [539](#)
- pack
 - CoinModel, [196](#)
- packCols
 - CoinModel, [195](#)
- packColumns
 - CoinModel, [195](#)
- packRows
 - CoinModel, [195](#)
- packedMatrix
 - CoinModel, [197](#)
- packedMode
 - _EKKfactinfo, [38](#)
 - CoinIndexedVector, [144](#)
- packedMode_
 - CoinIndexedVector, [145](#)
- pass_
 - CoinPresolveMatrix, [353](#)
- passInMatrix
 - CoinModel, [197](#)
- passInMessageHandler
 - CoinLpIO, [161](#)
 - CoinMpsIO, [241](#)
- permute
 - CoinDenseFactorization, [68](#)
 - CoinFactorization, [93](#)
 - CoinOslFactorization, [256](#)
 - CoinOtherFactorization, [263](#)
 - CoinSimpFactorization, [382](#)
- permute_
 - CoinFactorization, [107](#)
- permuteBack
 - CoinFactorization, [94](#), [103](#)
 - CoinOslFactorization, [254](#)
 - CoinOtherFactorization, [263](#)
- permuteBack_
 - CoinFactorization, [107](#)
- persistenceFlag
 - CoinFactorization, [98](#)
- persistenceFlag_
 - CoinFactorization, [114](#)
- pivot
 - CoinFactorization, [105](#)
- pivotCandLimit_
 - CoinSimpFactorization, [392](#)
- pivotColumn
 - CoinFactorization, [93](#)

- pivotColumn_
 - CoinFactorization, [107](#)
- pivotColumnBack
 - CoinFactorization, [94](#)
- pivotColumnBack_
 - CoinFactorization, [107](#)
- pivotColumnSingleton
 - CoinFactorization, [101](#)
- pivotOneOtherRow
 - CoinFactorization, [101](#)
- pivotRegion
 - CoinFactorization, [93](#)
- pivotRegion_
 - CoinFactorization, [109](#)
- pivotRow
 - CoinOslFactorization, [254](#)
 - CoinOtherFactorization, [262](#)
- pivotRow_
 - CoinOtherFactorization, [266](#)
- pivotRowL_
 - CoinFactorization, [109](#)
- pivotRowSingleton
 - CoinFactorization, [101](#)
- pivotTolerance
 - CoinFactorization, [96](#)
 - CoinOtherFactorization, [262](#)
- pivotTolerance_
 - CoinFactorization, [105](#)
 - CoinOtherFactorization, [265](#)
- pivoting
 - CoinSimpFactorization, [384](#)
- pivots
 - CoinFactorization, [93](#)
 - CoinOtherFactorization, [261](#)
- pointer
 - CoinModel, [198](#)
- pop
 - CoinSearchTreeBase, [363](#)
 - CoinSearchTreeManager, [367](#)
- position
 - CoinModel, [198](#)
 - CoinModelLink, [215](#)
- position_
 - CoinMpsCardReader, [225](#)
- postProcess
 - CoinDenseFactorization, [67](#)
 - CoinOslFactorization, [253](#)
 - CoinOtherFactorization, [264](#)
 - CoinSimpFactorization, [381](#)
- postsolve
 - CoinPresolveAction, [339](#)
 - do_tighten_action, [455](#)
 - doubleton_action, [456](#)
 - drop_empty_cols_action, [458](#)
 - drop_empty_rows_action, [459](#)
 - drop_zero_coefficients_action, [460](#)
 - dupcol_action, [462](#)
 - duprow_action, [464](#)
 - forcing_constraint_action, [467](#)
 - gubrow_action, [468](#)
 - implied_free_action, [469](#)
 - isolated_constraint_action, [470](#)
 - make_fixed_action, [472](#)
 - remove_dual_action, [478](#)
 - remove_fixed_action, [480](#)
 - slack_doubleton_action, [482](#)
 - slack_singleton_action, [483](#)
 - subst_constraint_action, [490](#)
 - tripleton_action, [492](#)
 - twoxtwo_action, [493](#)
 - useless_constraint_action, [495](#)
- drop_empty_rows_action, [459](#)
- drop_zero_coefficients_action, [460](#)
- dupcol_action, [462](#)
- duprow_action, [464](#)
- forcing_constraint_action, [467](#)
- gubrow_action, [468](#)
- implied_free_action, [469](#)
- isolated_constraint_action, [470](#)
- make_fixed_action, [472](#)
- remove_dual_action, [478](#)
- remove_fixed_action, [480](#)
- slack_doubleton_action, [482](#)
- slack_singleton_action, [483](#)
- subst_constraint_action, [490](#)
- tripleton_action, [492](#)
- twoxtwo_action, [493](#)
- useless_constraint_action, [495](#)
- pre
 - EKKHlink, [464](#)
 - presolvehlink, [474](#)
- preProcess
 - CoinDenseFactorization, [67](#)
 - CoinFactorization, [101](#)
 - CoinOslFactorization, [253](#)
 - CoinOtherFactorization, [264](#)
 - CoinSimpFactorization, [381](#)
- precision
 - CoinMessageHandler, [174](#)
- prefix
 - CoinMessageHandler, [174](#)
- prefix_
 - CoinMessageHandler, [178](#)
- presolve
 - do_tighten_action, [455](#)
 - doubleton_action, [456](#)
 - drop_empty_cols_action, [458](#)
 - drop_empty_rows_action, [459](#)
 - drop_zero_coefficients_action, [460](#)
 - dupcol_action, [462](#)
 - duprow_action, [464](#)
 - forcing_constraint_action, [467](#)
 - gubrow_action, [468](#)
 - implied_free_action, [469](#)
 - isolated_constraint_action, [470](#)
 - make_fixed_action, [472](#)
 - remove_dual_action, [478](#)
 - remove_fixed_action, [480](#)
 - slack_doubleton_action, [482](#)
 - slack_singleton_action, [483](#)
 - subst_constraint_action, [490](#)
 - tripleton_action, [492](#)
 - twoxtwo_action, [493](#)
 - useless_constraint_action, [495](#)
- Presolve Debug Functions, [26](#)

- presolve_check_duals, [27](#)
- presolve_check_free_list, [27](#)
- presolve_check_nbasic, [28](#)
- presolve_check_reduced_costs, [27](#)
- presolve_check_sol, [27](#), [28](#)
- presolve_check_threads, [27](#)
- presolve_consistent, [27](#)
- presolve_links_ok, [27](#)
- presolve_no_dups, [27](#)
- presolve_no_zeros, [27](#)
- Presolve Matrix Manipulation Functions, [19](#)
 - presolve_delete_from_col, [23](#)
 - presolve_delete_from_col2, [23](#)
 - presolve_delete_from_major, [22](#)
 - presolve_delete_from_major2, [23](#)
 - presolve_delete_from_row, [23](#)
 - presolve_delete_many_from_major, [23](#)
 - presolve_expand_col, [21](#)
 - presolve_expand_major, [21](#)
 - presolve_expand_row, [21](#)
 - presolve_find_col, [21](#)
 - presolve_find_col1, [22](#)
 - presolve_find_minor, [21](#)
 - presolve_find_minor1, [21](#)
 - presolve_find_minor2, [22](#)
 - presolve_find_minor3, [22](#)
 - presolve_find_row, [21](#)
 - presolve_find_row1, [22](#)
 - presolve_find_row2, [22](#)
 - presolve_find_row3, [22](#)
 - presolve_make_memlists, [21](#)
- Presolve Utility Functions, [25](#)
 - coin_init_random_vec, [25](#)
 - presolve_dupmajor, [25](#)
- presolve_check_duals
 - Presolve Debug Functions, [27](#)
- presolve_check_free_list
 - Presolve Debug Functions, [27](#)
- presolve_check_nbasic
 - Presolve Debug Functions, [28](#)
- presolve_check_reduced_costs
 - Presolve Debug Functions, [27](#)
- presolve_check_sol
 - Presolve Debug Functions, [27](#), [28](#)
- presolve_check_threads
 - Presolve Debug Functions, [27](#)
- presolve_consistent
 - Presolve Debug Functions, [27](#)
- presolve_delete_from_col
 - Presolve Matrix Manipulation Functions, [23](#)
- presolve_delete_from_col2
 - Presolve Matrix Manipulation Functions, [23](#)
- presolve_delete_from_major
 - Presolve Matrix Manipulation Functions, [22](#)
- presolve_delete_from_major2
 - Presolve Matrix Manipulation Functions, [23](#)
- presolve_delete_from_row
 - Presolve Matrix Manipulation Functions, [23](#)
- presolve_delete_many_from_major
 - Presolve Matrix Manipulation Functions, [23](#)
- presolve_dupmajor
 - Presolve Utility Functions, [25](#)
- presolve_expand_col
 - Presolve Matrix Manipulation Functions, [21](#)
- presolve_expand_major
 - Presolve Matrix Manipulation Functions, [21](#)
- presolve_expand_row
 - Presolve Matrix Manipulation Functions, [21](#)
- presolve_find_col
 - Presolve Matrix Manipulation Functions, [21](#)
- presolve_find_col1
 - Presolve Matrix Manipulation Functions, [22](#)
- presolve_find_minor
 - Presolve Matrix Manipulation Functions, [21](#)
- presolve_find_minor1
 - Presolve Matrix Manipulation Functions, [21](#)
- presolve_find_minor2
 - Presolve Matrix Manipulation Functions, [22](#)
- presolve_find_minor3
 - Presolve Matrix Manipulation Functions, [22](#)
- presolve_find_row
 - Presolve Matrix Manipulation Functions, [21](#)
- presolve_find_row1
 - Presolve Matrix Manipulation Functions, [22](#)
- presolve_find_row2
 - Presolve Matrix Manipulation Functions, [22](#)
- presolve_find_row3
 - Presolve Matrix Manipulation Functions, [22](#)
- presolve_links_ok
 - Presolve Debug Functions, [27](#)
- presolve_make_memlists
 - Presolve Matrix Manipulation Functions, [21](#)
- presolve_no_dups
 - Presolve Debug Functions, [27](#)
- presolve_no_zeros
 - Presolve Debug Functions, [27](#)
- presolveOptions
 - CoinPresolveMatrix, [347](#)
- presolveOptions_
 - CoinPresolveMatrix, [355](#)
- presolveX
 - subst_constraint_action, [490](#)
- presolvehlink, [473](#)
 - PRESOLVE_INSERT_LINK, [473](#)
 - PRESOLVE_MOVE_LINK, [474](#)
 - PRESOLVE_REMOVE_LINK, [473](#)
- pre, [474](#)
- suc, [474](#)

- prevColInU_
 - CoinSimpFactorization, [389](#)
- prevColumn
 - FactorPointers, [465](#)
- prevRow
 - FactorPointers, [465](#)
- prevRowInU_
 - CoinSimpFactorization, [389](#)
- previous
 - CoinModel, [199](#)
 - CoinModelLinkedList, [219](#)
- previous_names_
 - CoinLpIO, [166](#)
- primal
 - CoinWarmStartPrimalDual, [440](#)
- primalSize
 - CoinWarmStartPrimalDual, [440](#)
- print
 - CoinError, [77](#)
 - CoinIndexedVector, [142](#)
 - CoinLpIO, [161](#)
 - CoinMessageHandler, [173](#)
 - CoinPartitionedVector, [318](#)
 - CoinShallowPackedVector, [373](#)
 - CoinWarmStartBasis, [430](#)
- printErrors_
 - CoinError, [78](#)
- printGenericHelp
 - CoinParam, [313](#)
 - CoinParamUtils, [32](#)
- printHelp
 - CoinParam, [314](#)
 - CoinParamUtils, [32](#)
- printIt
 - CoinParam, [313](#)
 - CoinParamUtils, [32](#)
- printKwds
 - CoinParam, [309](#)
- printLongHelp
 - CoinParam, [310](#)
- printMatrixElement
 - CoinPackedMatrix, [282](#)
- printPref
 - CoinTreeSiblings, [417](#)
- printStatus_
 - CoinMessageHandler, [178](#)
- printing
 - CoinMessageHandler, [177](#)
- priorities
 - CoinModel, [207](#)
- problemName_
 - CoinBaseModel, [58](#)
 - CoinLpIO, [164](#)
 - CoinMpsIO, [243](#)
- pullFunc
 - CoinParam, [311](#)
- push
 - CoinSearchTreeBase, [363](#)
 - CoinSearchTreeManager, [367](#)
- pushFunc
 - CoinParam, [311](#)
- puts
 - CoinFileOutput, [125](#)
- quadraticRow
 - CoinModel, [206](#)
- quickAdd
 - CoinIndexedVector, [140](#)
- quickAddNonZero
 - CoinIndexedVector, [140](#)
- quickInsert
 - CoinIndexedVector, [140](#)
- R_etas_element
 - _EKKfactinfo, [35](#)
- R_etas_index
 - _EKKfactinfo, [35](#)
- R_etas_start
 - _EKKfactinfo, [35](#)
- randomDouble
 - CoinThreadRandom, [411](#)
- randomNumber_
 - CoinPresolveMatrix, [356](#)
- randomize
 - CoinThreadRandom, [411](#)
- rangeName_
 - CoinMpsIO, [243](#)
- rawSize
 - CoinArrayWithLength, [52](#)
- rcosts_
 - CoinPrePostsolveMatrix, [336](#)
- rdone_
 - CoinPostsolveMatrix, [322](#)
- read
 - CoinFileInput, [120](#)
- read_monom_obj
 - CoinLpIO, [163](#)
- read_monom_row
 - CoinLpIO, [163](#)
- read_row
 - CoinLpIO, [163](#)
- readBasis
 - CoinMpsIO, [239](#)
- readConicMps
 - CoinMpsIO, [240](#)
- readGMPL
 - CoinMpsIO, [239](#)
- readGms
 - CoinMpsIO, [239](#)

- readLp
 - CoinLpIO, 160, 161
- readMps
 - CoinMpsIO, 238
- readQuadraticMps
 - CoinMpsIO, 240
- readToNextSection
 - CoinMpsCardReader, 223
- readType_
 - CoinFileIOBase, 123
- reader
 - CoinMpsIO, 240
- reader_
 - CoinMpsCardReader, 227
- realloc_coeff
 - CoinLpIO, 163
- realloc_col
 - CoinLpIO, 163
- realloc_row
 - CoinLpIO, 163
- reallyFreeArray
 - CoinArrayWithLength, 53
- realpop
 - CoinSearchTree, 360
 - CoinSearchTreeBase, 362
- realpush
 - CoinSearchTree, 360
 - CoinSearchTreeBase, 362
- recomputeSums
 - CoinPresolveMatrix, 348
- reevaluateSearchStrategy
 - CoinSearchTreeManager, 368
- ReferenceCount
 - Coin::ReferencedObject, 476
- ReferencedObject
 - Coin::ReferencedObject, 476
- refresh
 - CoinStructuredModel, 409
- relaxAccuracyCheck
 - CoinFactorization, 95
 - CoinOtherFactorization, 261
- relaxCheck_
 - CoinFactorization, 105
 - CoinOtherFactorization, 265
- releaseColumnInformation
 - CoinMpsIO, 241
- releaseColumnNames
 - CoinMpsIO, 242
- releaseIntegerInformation
 - CoinMpsIO, 241
- releaseMatrixInformation
 - CoinMpsIO, 242
- releaseRedundantInformation
 - CoinMpsIO, 241
- ReleaseRef
 - Coin::ReferencedObject, 476
- releaseRowInformation
 - CoinMpsIO, 241
- releaseRowNames
 - CoinMpsIO, 242
- remove_dual_action, 477
 - ~remove_dual_action, 478
 - name, 478
 - postsolve, 478
 - presolve, 478
- remove_fixed
 - remove_fixed_action, 480
- remove_fixed_action, 478
 - ~remove_fixed_action, 480
 - actions_, 481
 - cols_, 480
 - colrows_, 480
 - nactions_, 480
 - name, 480
 - postsolve, 480
 - presolve, 480
 - remove_fixed, 480
- remove_fixed_action::action, 40
 - col, 41
 - sol, 41
 - start, 41
- removeColumnFromActSet
 - CoinSimpFactorization, 384
- removeGaps
 - CoinPackedMatrix, 279
- removeRowFromActSet
 - CoinSimpFactorization, 384
- reorder
 - CoinModel, 206
- reorderU
 - CoinFactorization, 101
- replaceColumn
 - CoinDenseFactorization, 67
 - CoinFactorization, 98
 - CoinOslFactorization, 255
 - CoinOtherFactorization, 264
 - CoinSimpFactorization, 381
- replaceColumnPFI
 - CoinFactorization, 105
- replaceColumnU
 - CoinFactorization, 99
- replaceMessage
 - CoinMessages, 181
 - CoinOneMessage, 248
- replaceQuadraticRow
 - CoinModel, 206
- replaceRow
 - CoinFactorization, 100

- replaceVector
 - CoinPackedMatrix, [279](#)
- reserve
 - CoinIndexedVector, [144](#)
 - CoinPackedMatrix, [275](#)
 - CoinPackedVector, [296](#)
 - CoinPartitionedVector, [317](#)
- reset
 - CoinTimer, [413](#)
- resetStatistics
 - CoinFactorization, [101](#)
- resize
 - CoinDenseVector, [72](#)
 - CoinModelHash, [208](#)
 - CoinModelHash2, [211](#)
 - CoinModelLinkedList, [218](#)
 - CoinWarmStartBasis, [429](#)
- resizeForAddingMajorVectors
 - CoinPackedMatrix, [286](#)
- resizeForAddingMinorVectors
 - CoinPackedMatrix, [286](#)
- restart
 - CoinTimer, [413](#)
- restoreFactorization
 - CoinFactorization, [92](#)
- returnVector
 - CoinIndexedVector, [139](#)
- reverseOrderedCopyOf
 - CoinPackedMatrix, [280](#)
- reverseOrdering
 - CoinPackedMatrix, [280](#)
- rhs
 - CoinModelInfo2, [213](#)
- rhs_
 - CoinLpIO, [165](#)
 - CoinMpsIO, [244](#)
- rhsName_
 - CoinMpsIO, [243](#)
- rightAppendPackedMatrix
 - CoinPackedMatrix, [278](#)
- rlink_
 - CoinPresolveMatrix, [352](#)
- rlo
 - doubleton_action::action, [40](#)
 - tripleton_action::action, [43](#)
- rlo_
 - CoinPrePostsolveMatrix, [335](#)
- row
 - CoinBuild, [63](#)
 - CoinModel, [202](#)
 - CoinModelLink, [215](#)
 - CoinModelTriple, [220](#)
 - doubleton_action::action, [40](#)
 - dropped_zero, [461](#)
 - forcing_constraint_action::action, [42](#)
 - tripleton_action::action, [43](#)
- rowBlock
 - CoinModelInfo2, [213](#)
 - CoinStructuredModel, [408](#)
- rowBlockName_
 - CoinBaseModel, [58](#)
- rowChanged
 - CoinPresolveMatrix, [350](#)
- rowChanged_
 - CoinPresolveMatrix, [354](#)
- rowInTriple
 - CoinModelUseful.hpp, [520](#)
- rowIndex
 - CoinLpIO, [157](#)
 - CoinMpsIO, [236](#)
- rowsBasic
 - CoinPrePostsolveMatrix, [329](#)
- rowLower
 - CoinModel, [199](#)
- rowLowerArray
 - CoinModel, [203](#)
- rowLowerAsString
 - CoinModel, [201](#)
- rowMax
 - FactorPointers, [465](#)
- rowName
 - CoinLpIO, [157](#)
 - CoinModel, [199](#)
 - CoinModelInfo2, [213](#)
 - CoinMpsCardReader, [224](#)
 - CoinMpsIO, [236](#)
- rowName_
 - CoinMpsCardReader, [226](#)
- rowNames
 - CoinModel, [204](#)
- rowOfU_
 - CoinSimpFactorization, [390](#)
- rowPosition_
 - CoinSimpFactorization, [391](#)
- rowProhibited
 - CoinPresolveMatrix, [351](#)
- rowProhibited2
 - CoinPresolveMatrix, [351](#)
- rowStatusString
 - CoinPrePostsolveMatrix, [330](#)
- rowUpper
 - CoinModel, [199](#)
- rowUpperArray
 - CoinModel, [203](#)
- rowUpperAsString
 - CoinModel, [201](#)
- rowUsed
 - CoinPresolveMatrix, [351](#)

- rowcols
 - forcing_constraint_action::action, 42
- rowduals_
 - CoinPrePostsolveMatrix, 336
- rowels_
 - CoinPresolveMatrix, 352
- rowlower_
 - CoinLpIO, 165
 - CoinMpsIO, 244
- rowrange_
 - CoinLpIO, 165
 - CoinMpsIO, 244
- rows_ok
 - _EKKfactinfo, 38
- rowsToDo_
 - CoinPresolveMatrix, 354
- rowsense_
 - CoinLpIO, 165
 - CoinMpsIO, 244
- rowstat_
 - CoinPrePostsolveMatrix, 336
- rowupper_
 - CoinLpIO, 165
 - CoinMpsIO, 245
- rup
 - tripleton_action::action, 43
- rup_
 - CoinPrePostsolveMatrix, 335
- SHIFT_INDEX
 - CoinOsIC.h, 526
- SHIFT_REF
 - CoinOsIC.h, 526
- SLACK_DOUBLETON
 - CoinPresolveSingleton.hpp, 540
- SLACK_SINGLETON
 - CoinPresolveSingleton.hpp, 540
- SLACK_VALUE
 - CoinOsIC.h, 525
- SPARSE_UPDATE
 - CoinOsIC.h, 525
- SUBST_ROW
 - CoinPresolveSubst.hpp, 541
- SWAP
 - CoinOsIC.h, 526
- saveColumn_
 - CoinFactorization, 108
- saveFactorization
 - CoinFactorization, 92
- Sbb_Model
 - Coin_C_defines.h, 497
- scale
 - CoinDenseVector, 73
- scan
 - CoinIndexedVector, 141
 - CoinPartitionedVector, 317
- scan_next
 - CoinLpIO, 162
- scanAndPack
 - CoinIndexedVector, 141
- secRowOfU_
 - CoinSimpFactorization, 391
- secRowPosition_
 - CoinSimpFactorization, 391
- second
 - CoinPair, 303
 - CoinTriple, 418
- section_
 - CoinMpsCardReader, 226
- seed_
 - CoinThreadRandom, 411
- separateLinks
 - CoinFactorization, 102
- set
 - BitVector128, 45
- setAllowStringElements
 - CoinMpsIO, 238
- setAnyInteger
 - CoinPresolveMatrix, 347
- setAnyProhibited
 - CoinPresolveMatrix, 352
- setArtifStatus
 - CoinWarmStartBasis, 428
- setArtificialStatus
 - CoinPrePostsolveMatrix, 330
- setBiasLU
 - CoinFactorization, 98
- setBit
 - BitVector128, 45
- setCapacity
 - CoinArrayWithLength, 52
- setCoinModel
 - CoinStructuredModel, 409
- setColBounds
 - CoinModel, 194
- setColChanged
 - CoinPresolveMatrix, 349
- setColInfinite
 - CoinPresolveMatrix, 350
- setCollsInteger
 - CoinModel, 194
- setColLower
 - CoinModel, 193, 196
 - CoinPrePostsolveMatrix, 330
 - CoinSnapshot, 401
- setColName
 - CoinModel, 194
- setColObjective

- CoinModel, 194
- setColProhibited
 - CoinPresolveMatrix, 349
- setColSolution
 - CoinPrePostsolveMatrix, 330
 - CoinSnapshot, 402
- setColType
 - CoinSnapshot, 401
- setColUpper
 - CoinModel, 193, 196
 - CoinPrePostsolveMatrix, 330
 - CoinSnapshot, 401
- setColUsed
 - CoinPresolveMatrix, 350
- setCollectStatistics
 - CoinFactorization, 100
- setColumn
 - CoinModelLink, 215
- setColumnBlock
 - CoinBaseModel, 57
 - CoinStructuredModel, 408
- setColumnBounds
 - CoinModel, 193
- setColumnIsInteger
 - CoinModel, 193, 194
- setColumnLower
 - CoinModel, 193, 194, 196
- setColumnName
 - CoinModel, 193
- setColumnObjective
 - CoinModel, 193, 194
- setColumnStatus
 - CoinPrePostsolveMatrix, 329
- setColumnStatusUsingValue
 - CoinPrePostsolveMatrix, 329
- setColumnUpper
 - CoinModel, 193, 194, 196
- setConstant
 - CoinDenseVector, 72
 - CoinIndexedVector, 140
 - CoinPackedVector, 295
- setContinuous
 - CoinModel, 193
- setConvertObjective
 - CoinMpsIO, 240
- setCost
 - CoinPrePostsolveMatrix, 330
- setCurrentColumn
 - CoinBuild, 63
- setCurrentRow
 - CoinBuild, 63
- setCutMarker
 - CoinModel, 207
- setDbIVal
 - CoinParam, 309
- setDecimals
 - CoinLpIO, 158
- setDefaultBound
 - CoinMpsIO, 237
- setDefaultColNames
 - CoinLpIO, 159
- setDefaultRowNames
 - CoinLpIO, 159
- setDenseThreshold
 - CoinFactorization, 96
- setDenseVector
 - CoinIndexedVector, 138
- setDepth
 - CoinTreeNode, 415
- setDetail
 - CoinOneMessage, 249
- setDetailMessage
 - CoinMessages, 182
- setDetailMessages
 - CoinMessages, 182
- setDimensions
 - CoinPackedMatrix, 277
- setDisplay
 - CoinParam, 310
- setDoNotSeparateThis
 - CoinSnapshot, 402
- setDualTolerance
 - CoinPrePostsolveMatrix, 330
 - CoinSnapshot, 402
- setElement
 - CoinDenseVector, 72
 - CoinIndexedVector, 140
 - CoinModel, 192
 - CoinModelLink, 216
 - CoinPackedVector, 295
- setEpsilon
 - CoinLpIO, 158
- setExternalNumber
 - CoinOneMessage, 249
- setExtraGap
 - CoinPackedMatrix, 277
- setExtraMajor
 - CoinPackedMatrix, 277
- setFeasibilityTolerance
 - CoinPresolveMatrix, 348
- setFileName
 - CoinMpsIO, 238
- setFilePointer
 - CoinMessageHandler, 176
- setForrestTomlin
 - CoinFactorization, 97
- setFractionality
 - CoinTreeNode, 415

- setFreeFormat
 - CoinMpsCardReader, [224](#)
- setFull
 - CoinIndexedVector, [140](#)
 - CoinPackedVector, [295](#)
- setFullNonZero
 - CoinPackedVector, [295](#)
- setIndexVector
 - CoinIndexedVector, [139](#)
- setInfinity
 - CoinLpIO, [158](#)
 - CoinMpsIO, [237](#)
 - CoinSnapshot, [402](#)
- setInputSrc
 - CoinParam, [311](#)
 - CoinParamUtils, [30](#)
- setIntVal
 - CoinParam, [309](#)
- setInteger
 - CoinModel, [193](#)
- setIntegerLowerBound
 - CoinSnapshot, [403](#)
- setIntegerTolerance
 - CoinSnapshot, [403](#)
- setIntegerUpperBound
 - CoinSnapshot, [403](#)
- setIsInteger
 - CoinModel, [193](#), [195](#)
- setKwdVal
 - CoinParam, [309](#)
- setLanguage
 - CoinLpIO, [161](#)
 - CoinMessages, [181](#)
 - CoinMpsIO, [241](#)
- setLimit
 - CoinTimer, [414](#)
- setLogLevel
 - CoinBaseModel, [57](#)
 - CoinMessageHandler, [173](#), [174](#)
- setLongHelp
 - CoinParam, [309](#)
- setLpDataRowAndColNames
 - CoinLpIO, [159](#)
- setLpDataWithoutRowAndColNames
 - CoinLpIO, [159](#)
- setMajorDim
 - CoinPackedMatrix, [282](#)
- setMatrix
 - CoinPresolveMatrix, [346](#)
- setMatrixByCol
 - CoinSnapshot, [401](#)
- setMatrixByRow
 - CoinSnapshot, [401](#)
- setMaximumSubstitutionLevel
 - CoinPresolveMatrix, [348](#)
- setMessageHandler
 - CoinPrePostsolveMatrix, [333](#)
- setMinorDim
 - CoinPackedMatrix, [282](#)
- setMoreInfo
 - CoinModel, [205](#)
- setMpsData
 - CoinMpsIO, [237](#)
- setMpsDataColAndRowNames
 - CoinMpsIO, [242](#)
- setMpsDataWithoutRowAndColNames
 - CoinMpsIO, [242](#)
- setName
 - CoinModelHash, [209](#)
 - CoinParam, [310](#)
- setNext
 - CoinPresolveAction, [339](#)
- setNumCols
 - CoinSnapshot, [400](#)
- setNumElements
 - CoinIndexedVector, [139](#)
 - CoinPackedMatrix, [285](#)
 - CoinSnapshot, [400](#)
- setNumElementsPartition
 - CoinPartitionedVector, [317](#)
- setNumIntegers
 - CoinSnapshot, [400](#)
- setNumRows
 - CoinSnapshot, [400](#)
- setNumberAcross
 - CoinLpIO, [158](#)
- setNumberElementsU
 - CoinFactorization, [97](#)
- setNumberItems
 - CoinModelHash, [209](#)
 - CoinModelHash2, [211](#)
- setNumberRows
 - CoinFactorization, [94](#)
 - CoinOtherFactorization, [261](#)
- setObjCoefficients
 - CoinSnapshot, [401](#)
- setObjOffset
 - CoinPrePostsolveMatrix, [330](#)
 - CoinSnapshot, [402](#)
- setObjSense
 - CoinPrePostsolveMatrix, [330](#)
 - CoinSnapshot, [401](#)
- setObjValue
 - CoinSnapshot, [402](#)
- setObjective
 - CoinModel, [193](#), [194](#), [196](#)
- setObjectiveName
 - CoinMpsIO, [237](#)

- setObjectiveOffset
 - CoinBaseModel, 56
 - CoinLpIO, 158
 - CoinMpsIO, 236
- setOnRow
 - CoinModelLink, 216
- setOptimizationDirection
 - CoinBaseModel, 57
 - CoinModel, 204
 - CoinStructuredModel, 409
- setOriginalIndices
 - CoinModel, 207
- setOriginalMatrixByCol
 - CoinSnapshot, 402
- setOriginalMatrixByRow
 - CoinSnapshot, 402
- setPacked
 - CoinIndexedVector, 141
- setPackedMode
 - CoinIndexedVector, 144
- setPartitions
 - CoinPartitionedVector, 317
- setPass
 - CoinPresolveMatrix, 348
- setPersistence
 - CoinArrayWithLength, 53
- setPersistenceFlag
 - CoinFactorization, 98
- setPivots
 - CoinFactorization, 93
 - CoinOtherFactorization, 261
- setPosition
 - CoinModelLink, 216
 - CoinMpsCardReader, 224
- setPrecision
 - CoinMessageHandler, 174
- setPreferred
 - CoinTreeNode, 416
- setPrefix
 - CoinMessageHandler, 174
- setPresolveOptions
 - CoinPresolveMatrix, 347
- setPrimalTolerance
 - CoinPrePostsolveMatrix, 330
 - CoinSnapshot, 402
- setPriorities
 - CoinModel, 207
- setProblemName
 - CoinBaseModel, 57
 - CoinLpIO, 155
 - CoinMpsIO, 237
- setPullFunc
 - CoinParam, 311
- setPushFunc
 - CoinParam, 311
- setQuadraticElement
 - CoinModel, 192
- setQuality
 - CoinTreeNode, 415
- setReducedCost
 - CoinPrePostsolveMatrix, 331
 - CoinSnapshot, 402
- setRightHandSide
 - CoinSnapshot, 401
- setRow
 - CoinModelLink, 215
- setRowActivity
 - CoinPrePostsolveMatrix, 331
 - CoinSnapshot, 402
- setRowAndStringInTriple
 - CoinModelUseful.hpp, 520
- setRowBlock
 - CoinBaseModel, 57
 - CoinStructuredModel, 408
- setRowBounds
 - CoinModel, 192
- setRowChanged
 - CoinPresolveMatrix, 351
- setRowInTriple
 - CoinModelUseful.hpp, 520
- setRowLower
 - CoinModel, 192, 194, 196
 - CoinPrePostsolveMatrix, 331
 - CoinSnapshot, 401
- setRowName
 - CoinModel, 192
- setRowPrice
 - CoinPrePostsolveMatrix, 331
 - CoinSnapshot, 402
- setRowProhibited
 - CoinPresolveMatrix, 351
- setRowStatus
 - CoinPrePostsolveMatrix, 329
- setRowStatusUsingValue
 - CoinPrePostsolveMatrix, 329
- setRowUpper
 - CoinModel, 192, 194, 196
 - CoinPrePostsolveMatrix, 331
 - CoinSnapshot, 401
- setRowUsed
 - CoinPresolveMatrix, 351
- setSeed
 - CoinThreadRandom, 411
- setShortHelp
 - CoinParam, 309
- setSize
 - CoinArbitraryArrayWithLength, 48
 - CoinArrayWithLength, 52

- CoinBigIndexArrayWithLength, 60
- CoinDoubleArrayWithLength, 75
- CoinFactorizationDoubleArrayWithLength, 116
- CoinFactorizationLongDoubleArrayWithLength, 118
- CoinIntArrayWithLength, 147
- CoinUnsignedIntArrayWithLength, 420
- CoinVoidStarArrayWithLength, 422
- CoinWarmStartBasis, 429
- setSmallElementValue
 - CoinMpsIO, 238
- setSolveMode
 - CoinOtherFactorization, 263
- setStatus
 - CoinFactorization, 93
 - CoinOtherFactorization, 260
 - CoinPrePostsolveMatrix, 330
 - CoinPresolveMatrix, 348
 - CoinWarmStartBasis, 431
- setStrVal
 - CoinParam, 309
- setStringInTriple
 - CoinModelUseful.hpp, 520
- setStringsAllowed
 - CoinMpsCardReader, 225
- setStructStatus
 - CoinWarmStartBasis, 428
- setStructuralStatus
 - CoinPrePostsolveMatrix, 330
- setTempNumElementsPartition
 - CoinPartitionedVector, 317
- setTestForDuplicateIndex
 - CoinPackedVectorBase, 299
- setTestForDuplicateIndexWhenTrue
 - CoinPackedVectorBase, 299
- setTestsOff
 - CoinPackedVectorBase, 300
- setTree
 - CoinSearchTreeManager, 367
- setTrueLB
 - CoinTreeNode, 415
- setType
 - CoinParam, 310
 - CoinSet, 369
- setType_
 - CoinSet, 370
- setUsefulInformation
 - CoinOslFactorization, 254
 - CoinOtherFactorization, 263
- setValue
 - CoinModelLink, 215
- setVariableType
 - CoinPresolveMatrix, 346, 347
- setVector
 - CoinDenseVector, 72
- CoinIndexedVector, 139, 140
- CoinPackedVector, 294
- CoinShallowPackedVector, 373
- setWhichSection
 - CoinMpsCardReader, 223
- severity
 - CoinOneMessage, 249
- severity_
 - CoinOneMessage, 249
- shortHelp
 - CoinParam, 309
- shortOrHelpMany
 - CoinParam, 313
 - CoinParamUtils, 32
- shortOrHelpOne
 - CoinParam, 313
 - CoinParamUtils, 32
- show_self
 - CoinFactorization, 92
- size
 - CoinDenseVector, 71
 - CoinSearchTreeBase, 362
 - CoinSearchTreeManager, 367
 - CoinTreeSiblings, 417
 - CoinWarmStartDual, 436
 - CoinWarmStartVector, 445
- size0
 - CoinWarmStartVectorPair, 450
- size1
 - CoinWarmStartVectorPair, 450
- size_
 - CoinArrayWithLength, 54
 - CoinPackedMatrix, 288
 - CoinSearchTreeBase, 363
- skip_comment
 - CoinLpIO, 162
- slack_doubleton_action, 481
 - ~slack_doubleton_action, 482
 - name, 482
 - postsolve, 482
 - presolve, 482
- slack_singleton_action, 482
 - ~slack_singleton_action, 483
 - name, 483
 - postsolve, 483
 - presolve, 483
- slackValue
 - CoinFactorization, 96
 - CoinOtherFactorization, 262
- slackValue_
 - CoinFactorization, 105
 - CoinOtherFactorization, 265
- smallElement_
 - CoinMpsIO, 246

SmartPtr
 Coin::SmartPtr, 486
sol
 remove_fixed_action::action, 41
sol_
 CoinPrePostsolveMatrix, 335
solveMode
 CoinOtherFactorization, 263
solveMode_
 CoinOtherFactorization, 266
sort
 CoinFactorization, 92
 CoinIndexedVector, 143
 CoinPackedVector, 296
 CoinPartitionedVector, 318
sortDecrElement
 CoinIndexedVector, 143
 CoinPackedVector, 296
sortDecrIndex
 CoinIndexedVector, 143
 CoinPackedVector, 296
sortIncrElement
 CoinIndexedVector, 143
 CoinPackedVector, 296
sortIncrIndex
 CoinIndexedVector, 143
 CoinPackedVector, 296
sortOriginalOrder
 CoinPackedVector, 296
sortPacked
 CoinIndexedVector, 143
sortedEta
 _EKKfactinfo, 38
sortedSparseDotProduct
 CoinPackedVector.hpp, 531
source_
 CoinMessageHandler, 178
 CoinMessages, 182
spaceForForrestTomlin
 CoinFactorization, 97
sparse_
 CoinFactorization, 114
sparseDotProduct
 CoinPackedVector.hpp, 531
sparseThreshold
 CoinFactorization, 99
sparseThreshold2_
 CoinFactorization, 113
sparseThreshold_
 CoinFactorization, 113
start
 remove_fixed_action::action, 41
start_
 CoinPackedMatrix, 288
startColumnL
 CoinFactorization, 94
startColumnL_
 CoinFactorization, 111
startColumnR_
 CoinFactorization, 111
startColumnU
 CoinFactorization, 98
startColumnU_
 CoinFactorization, 110
startHash
 CoinLpIO, 161
 CoinMpsIO, 243
startPartition
 CoinPartitionedVector, 316
startPartition_
 CoinPartitionedVector, 318
startPartitions
 CoinPartitionedVector, 317
startRowL
 CoinFactorization, 94
startRowL_
 CoinFactorization, 114
startRowU_
 CoinFactorization, 107
startTime_
 CoinPresolveMatrix, 353
starts
 CoinOslFactorization, 254
 CoinOtherFactorization, 262
statistics
 CoinPresolveMatrix, 348
Status
 CoinPrePostsolveMatrix, 328
 CoinWarmStartBasis, 427
status
 CoinFactorization, 93
 CoinOtherFactorization, 260
 CoinPresolveMatrix, 348
status_
 CoinFactorization, 107
 CoinOtherFactorization, 266
 CoinPresolveMatrix, 353
statusName
 CoinPrePostsolveMatrix, 333
 CoinWarmStartBasis, 431
stepColsToDo
 CoinPresolveMatrix, 349
stepRowsToDo
 CoinPresolveMatrix, 350
stopHash
 CoinLpIO, 161
 CoinMpsIO, 243
str

- BitVector128, [45](#)
- strVal
 - CoinParam, [309](#)
- strcpyAndCompress
 - CoinMpsCardReader, [225](#)
- stringArray
 - CoinModel, [203](#)
- stringElement
 - CoinMpsIO, [237](#)
- stringElements_
 - CoinMpsIO, [247](#)
- stringInTriple
 - CoinModelUseful.hpp, [520](#)
- stringValue
 - CoinMessageHandler, [175](#)
- stringValue_
 - CoinMessageHandler, [177](#)
- stringsAllowed_
 - CoinMpsCardReader, [227](#)
- stringsExist
 - CoinModel, [203](#)
- structuralStatus_
 - CoinWarmStartBasis, [431](#)
- submatrixOf
 - CoinPackedMatrix, [279](#)
- submatrixOfWithDuplicates
 - CoinPackedMatrix, [279](#)
- subst_constraint_action, [488](#)
 - ~subst_constraint_action, [489](#)
 - name, [489](#)
 - postsolve, [490](#)
 - presolve, [490](#)
 - presolveX, [490](#)
- suc
 - EKKHlink, [464](#)
 - presolvehlink, [474](#)
- sum
 - CoinDenseVector, [72](#)
 - CoinPackedVectorBase, [301](#)
- sumDown_
 - CoinPresolveMatrix, [356](#)
- sumUp_
 - CoinPresolveMatrix, [356](#)
- superBasic
 - CoinPrePostsolveMatrix, [328](#)
- swap
 - CoinArrayWithLength, [53](#)
 - CoinIndexedVector, [142](#)
 - CoinPackedMatrix, [281](#)
 - CoinPackedVector, [295](#)
 - CoinWarmStartPrimalDual, [441](#)
 - CoinWarmStartPrimalDualDiff, [443](#)
 - CoinWarmStartVector, [446](#)
 - CoinWarmStartVectorDiff, [448](#)
 - CoinWarmStartVectorPair, [450](#)
 - CoinWarmStartVectorPairDiff, [452](#)
- switch_off_sparse_update
 - _EKKfactinfo, [38](#)
- switchOff
 - CoinArrayWithLength, [52](#)
- switchOn
 - CoinArrayWithLength, [53](#)
- switchedOn
 - CoinArrayWithLength, [52](#)
- symbuf
 - CoinYacc, [453](#)
- symrec, [490](#)
 - CoinModelUseful.hpp, [520](#)
 - fnctptr, [491](#)
 - name, [490](#)
 - next, [491](#)
 - type, [490](#)
 - value, [491](#)
 - var, [490](#)
- symtable
 - CoinYacc, [453](#)
- synchronize
 - CoinModelLinkedList, [220](#)
- TRIPLETON
 - CoinPresolveTripletion.hpp, [542](#)
- testForDuplicateIndex
 - CoinPackedVectorBase, [300](#)
- testRedundant
 - useless_constraint_action, [495](#)
- third
 - CoinTriple, [418](#)
- throwCoinError
 - CoinPresolveAction, [339](#)
- tighten_zero_cost
 - CoinPresolveTighten.hpp, [541](#)
- timeElapsed
 - CoinTimer, [413](#)
- timeLeft
 - CoinTimer, [413](#)
- times
 - CoinPackedMatrix, [281](#)
- timesMajor
 - CoinPackedMatrix, [284](#)
- timesMinor
 - CoinPackedMatrix, [284](#)
- toCompact
 - CoinMessages, [182](#)
- toProcess
 - CoinTreeSiblings, [417](#)
- top
 - CoinSearchTreeBase, [362](#)
 - CoinSearchTreeManager, [367](#)

- totalElements_
 - CoinFactorization, [106](#)
- transferCosts
 - make_fixed_action, [472](#)
- transpose
 - CoinPackedMatrix, [281](#)
- transposeTimes
 - CoinPackedMatrix, [281](#)
- tripleton_action, [491](#)
 - ~tripleton_action, [492](#)
 - actions_, [492](#)
 - nactions_, [492](#)
 - name, [492](#)
 - postsolve, [492](#)
 - presolve, [492](#)
- tripleton_action::action, [42](#)
 - clox, [43](#)
 - cloy, [43](#)
 - coeffx, [43](#)
 - coeffy, [43](#)
 - coeffz, [44](#)
 - colel, [44](#)
 - costx, [43](#)
 - costy, [43](#)
 - cupx, [43](#)
 - cupy, [43](#)
 - icolx, [43](#)
 - icoly, [43](#)
 - icolz, [43](#)
 - ncolx, [44](#)
 - ncoly, [44](#)
 - rlo, [43](#)
 - row, [43](#)
 - rup, [43](#)
- trueStart
 - _EKKfactinfo, [35](#)
- truncate
 - CoinIndexedVector, [142](#)
 - CoinPackedVector, [295](#)
- tuning_
 - CoinPresolveMatrix, [353](#)
- twoNorm
 - CoinDenseVector, [72](#)
 - CoinPackedVectorBase, [301](#)
- twoxtwo_action, [492](#)
 - ~twoxtwo_action, [493](#)
 - name, [493](#)
 - postsolve, [493](#)
 - presolve, [493](#)
- type
 - CoinBuild, [64](#)
 - CoinModel, [202](#)
 - CoinParam, [310](#)
 - symrec, [490](#)
- UNROLL_LOOP_BODY1
 - CoinOslC.h, [526](#)
- UNROLL_LOOP_BODY2
 - CoinOslC.h, [526](#)
- UNROLL_LOOP_BODY4
 - CoinOslC.h, [526](#)
- UNSHIFT_INDEX
 - CoinOslC.h, [526](#)
- USELESS
 - CoinPresolveUseless.hpp, [542](#)
- UcolEnd_
 - CoinSimpFactorization, [390](#)
- UcolInd_
 - CoinSimpFactorization, [389](#)
- UcolLengths_
 - CoinSimpFactorization, [389](#)
- UcolMaxCap_
 - CoinSimpFactorization, [390](#)
- UcolStarts_
 - CoinSimpFactorization, [389](#)
- Ucolumns_
 - CoinSimpFactorization, [389](#)
- uk_en
 - CoinMessages, [181](#)
- unsetColChanged
 - CoinPresolveMatrix, [349](#)
- unsetColInfinite
 - CoinPresolveMatrix, [350](#)
- unsetColUsed
 - CoinPresolveMatrix, [350](#)
- unsetRowChanged
 - CoinPresolveMatrix, [351](#)
- unsetRowUsed
 - CoinPresolveMatrix, [351](#)
- unsetValue
 - CoinModel, [202](#)
 - CoinYacc, [454](#)
- upColumn
 - CoinSimpFactorization, [382](#)
- upColumnTranspose
 - CoinSimpFactorization, [382](#)
- update_model
 - CoinPresolveMatrix, [346](#)
- updateColumn
 - CoinDenseFactorization, [68](#)
 - CoinFactorization, [99](#)
 - CoinOslFactorization, [255](#)
 - CoinOtherFactorization, [264](#)
 - CoinSimpFactorization, [382](#)
- updateColumnFT
 - CoinDenseFactorization, [67](#)
 - CoinFactorization, [99](#)
 - CoinOslFactorization, [255](#)
 - CoinOtherFactorization, [264](#)

- CoinSimpFactorization, [381](#)
- updateColumnL
 - CoinFactorization, [102](#)
- updateColumnLDensish
 - CoinFactorization, [102](#)
- updateColumnLSparse
 - CoinFactorization, [102](#)
- updateColumnLSparsish
 - CoinFactorization, [103](#)
- updateColumnPFI
 - CoinFactorization, [103](#)
- updateColumnR
 - CoinFactorization, [103](#)
- updateColumnRFT
 - CoinFactorization, [103](#)
- updateColumnTranspose
 - CoinDenseFactorization, [68](#)
 - CoinFactorization, [99](#)
 - CoinOslFactorization, [255](#)
 - CoinOtherFactorization, [265](#)
 - CoinSimpFactorization, [382](#)
- updateColumnTransposeL
 - CoinFactorization, [104](#)
- updateColumnTransposeLByRow
 - CoinFactorization, [104](#)
- updateColumnTransposeLDensish
 - CoinFactorization, [104](#)
- updateColumnTransposeLSparse
 - CoinFactorization, [105](#)
- updateColumnTransposeLSparsish
 - CoinFactorization, [104](#)
- updateColumnTransposePFI
 - CoinFactorization, [103](#)
- updateColumnTransposeR
 - CoinFactorization, [104](#)
- updateColumnTransposeRDensish
 - CoinFactorization, [104](#)
- updateColumnTransposeRSparse
 - CoinFactorization, [104](#)
- updateColumnTransposeU
 - CoinFactorization, [103](#)
- updateColumnTransposeUByColumn
 - CoinFactorization, [104](#)
- updateColumnTransposeUDensish
 - CoinFactorization, [104](#)
- updateColumnTransposeUSparse
 - CoinFactorization, [104](#)
- updateColumnTransposeUSparsish
 - CoinFactorization, [104](#)
- updateColumnU
 - CoinFactorization, [103](#)
- updateColumnUDensish
 - CoinFactorization, [103](#)
- updateColumnUSparse
 - CoinFactorization, [103](#)
- updateColumnUSparsish
 - CoinFactorization, [103](#)
- updateCurrentRow
 - CoinSimpFactorization, [384](#)
- updateDeleted
 - CoinModelLinkedList, [219](#)
- updateDeletedOne
 - CoinModelLinkedList, [220](#)
- updateToI_
 - CoinSimpFactorization, [392](#)
- updateTwoColumnsFT
 - CoinDenseFactorization, [68](#)
 - CoinFactorization, [99](#)
 - CoinOslFactorization, [255](#)
 - CoinOtherFactorization, [264](#)
 - CoinSimpFactorization, [382](#)
- updateTwoColumnsUDensish
 - CoinFactorization, [103](#)
- UrowEnd_
 - CoinSimpFactorization, [389](#)
- UrowInd_
 - CoinSimpFactorization, [388](#)
- UrowLengths_
 - CoinSimpFactorization, [388](#)
- UrowMaxCap_
 - CoinSimpFactorization, [388](#)
- UrowStarts_
 - CoinSimpFactorization, [388](#)
- Urows_
 - CoinSimpFactorization, [388](#)
- us_en
 - CoinMessages, [181](#)
- usefulColumnDouble_
 - CoinPresolveMatrix, [356](#)
- usefulColumnInt_
 - CoinPresolveMatrix, [356](#)
- usefulRowDouble_
 - CoinPresolveMatrix, [356](#)
- usefulRowInt_
 - CoinPresolveMatrix, [355](#)
- useless_constraint_action, [494](#)
 - ~useless_constraint_action, [494](#)
 - name, [495](#)
 - postsolve, [495](#)
 - presolve, [495](#)
 - testRedundant, [495](#)
- Uxeqb
 - CoinSimpFactorization, [385](#)
- Uxeqb2
 - CoinSimpFactorization, [385](#)
- validateHash
 - CoinModelHash, [209](#)

- validateLinks
 - CoinModel, 206
 - CoinModelLinkedList, 220
- value
 - CoinModelLink, 215
 - CoinModelTriple, 220
 - CoinMpsCardReader, 224
 - symrec, 491
- value_
 - CoinMpsCardReader, 225
- valueString
 - CoinMpsCardReader, 224
- valueString_
 - CoinMpsCardReader, 227
- values
 - CoinWarmStartVector, 445
- values0
 - CoinWarmStartVectorPair, 450
- values1
 - CoinWarmStartVectorPair, 450
- var
 - symrec, 490
- vecKeep_
 - CoinSimpFactorization, 387
- vecLabels_
 - CoinSimpFactorization, 386
- verifyMtx
 - CoinPackedMatrix, 286
- void
 - Coin_C_defines.h, 497
- wantsTableauColumn
 - CoinOslFactorization, 254
 - CoinOtherFactorization, 263
- weights
 - CoinSet, 370
- weights_
 - CoinSet, 370
- whatIsSet
 - CoinModel, 205
- which
 - CoinSet, 369
- which_
 - CoinSet, 370
- whichSection
 - CoinMpsCardReader, 223
- WindowsErrorPopupBlocker
 - CoinError.hpp, 503
- workArea
 - CoinOslFactorization, 254
 - CoinOtherFactorization, 262
- workArea2_
 - CoinFactorization, 112
 - CoinSimpFactorization, 386
- workArea3_
 - CoinSimpFactorization, 386
- workArea_
 - CoinFactorization, 112
 - CoinOtherFactorization, 266
- write
 - CoinFileOutput, 125
- writeLp
 - CoinLpIO, 160
- writeMps
 - CoinModel, 196
 - CoinMpsIO, 239
 - CoinStructuredModel, 407
- xHeqb
 - CoinSimpFactorization, 385
- xLeqb
 - CoinSimpFactorization, 385
- xUeqb
 - CoinSimpFactorization, 385
- xcnadr
 - _EKKfactinfo, 35
- xcsadr
 - _EKKfactinfo, 34
- xe2adr
 - _EKKfactinfo, 36
- xecadr
 - _EKKfactinfo, 35
- xeeadr
 - _EKKfactinfo, 36
- xeradr
 - _EKKfactinfo, 35
- XferEntry
 - CoinWarmStartBasis, 426
- XferVec
 - CoinWarmStartBasis, 426
- xnetal
 - _EKKfactinfo, 37
- xnetalval
 - _EKKfactinfo, 38
- xrnadr
 - _EKKfactinfo, 34
- xrsadr
 - _EKKfactinfo, 34
- ZTOLDP
 - CoinPresolveMatrix.hpp, 539
- ZTOLDP2
 - CoinPresolveMatrix.hpp, 539
- zapColumnNames
 - CoinModel, 204
- zapRowNames
 - CoinModel, 204
- zero
 - CoinIndexedVector, 140

zeroTolerance
 _EKKfactinfo, [34](#)
 CoinFactorization, [96](#)
 CoinOtherFactorization, [262](#)
zeroTolerance_
 CoinFactorization, [105](#)
 CoinOtherFactorization, [265](#)
zpivlu
 _EKKfactinfo, [34](#)
ztoldj_
 CoinPrePostsolveMatrix, [335](#)
ztolzb_
 CoinPrePostsolveMatrix, [335](#)