

CoinUtils

2.9

Generated by Doxygen 1.7.1

Mon Nov 25 2013 17:36:29

Contents

1	Module Index	1
1.1	Modules	1
2	Namespace Index	1
2.1	Namespace List	1
3	Class Index	1
3.1	Class Hierarchy	1
4	Class Index	7
4.1	Class List	7
5	File Index	13
5.1	File List	13
6	Module Documentation	16
6.1	Presolve Matrix Manipulation Functions	16
6.1.1	Detailed Description	18
6.1.2	Function Documentation	18
6.2	Presolve Utility Functions	23
6.2.1	Detailed Description	23
6.2.2	Function Documentation	23
6.3	Presolve Debug Functions	24
6.3.1	Detailed Description	25
6.3.2	Function Documentation	26
7	Namespace Documentation	28
7.1	CoinParamUtils Namespace Reference	28
7.1.1	Detailed Description	29
7.1.2	Function Documentation	29
8	Class Documentation	35

8.1	_EKKfactinfo Struct Reference	35
8.1.1	Detailed Description	36
8.2	remove_fixed_action::action Struct Reference	36
8.2.1	Detailed Description	37
8.3	forcing_constraint_action::action Struct Reference	37
8.3.1	Detailed Description	37
8.4	tripleton_action::action Struct Reference	38
8.4.1	Detailed Description	38
8.5	doubleton_action::action Struct Reference	39
8.5.1	Detailed Description	39
8.6	BitVector128 Class Reference	40
8.6.1	Detailed Description	40
8.7	CoinAbsFltEq Class Reference	40
8.7.1	Detailed Description	42
8.7.2	Constructor & Destructor Documentation	42
8.8	CoinArbitraryArrayWithLength Class Reference	42
8.8.1	Detailed Description	44
8.8.2	Constructor & Destructor Documentation	45
8.8.3	Member Function Documentation	45
8.9	CoinArrayWithLength Class Reference	45
8.9.1	Detailed Description	50
8.9.2	Constructor & Destructor Documentation	50
8.9.3	Member Function Documentation	50
8.10	CoinBaseModel Class Reference	51
8.10.1	Detailed Description	54
8.10.2	Member Data Documentation	54
8.11	CoinBigIndexArrayWithLength Class Reference	54
8.11.1	Detailed Description	57
8.11.2	Constructor & Destructor Documentation	57
8.11.3	Member Function Documentation	57
8.12	CoinBuild Class Reference	57

8.12.1 Detailed Description	60
8.12.2 Constructor & Destructor Documentation	60
8.13 CoinDenseFactorization Class Reference	61
8.13.1 Detailed Description	65
8.13.2 Member Function Documentation	65
8.14 CoinDenseVector< T > Class Template Reference	66
8.14.1 Detailed Description	69
8.14.2 Member Function Documentation	69
8.15 CoinDoubleArrayWithLength Class Reference	70
8.15.1 Detailed Description	72
8.15.2 Constructor & Destructor Documentation	72
8.15.3 Member Function Documentation	73
8.16 CoinError Class Reference	73
8.16.1 Detailed Description	75
8.16.2 Friends And Related Function Documentation	76
8.17 CoinExternalVectorFirstGreater_2< S, T, V > Class Template Reference	76
8.17.1 Detailed Description	77
8.18 CoinExternalVectorFirstGreater_3< S, T, U, V > Class Template Reference	77
8.18.1 Detailed Description	78
8.19 CoinExternalVectorFirstLess_2< S, T, V > Class Template Reference	78
8.19.1 Detailed Description	79
8.20 CoinExternalVectorFirstLess_3< S, T, U, V > Class Template Reference	79
8.20.1 Detailed Description	80
8.21 CoinFactorization Class Reference	80
8.21.1 Detailed Description	99
8.21.2 Member Function Documentation	99
8.21.3 Member Data Documentation	104
8.22 CoinFactorizationDoubleArrayWithLength Class Reference	104
8.22.1 Detailed Description	107
8.22.2 Constructor & Destructor Documentation	107

8.22.3	Member Function Documentation	107
8.23	CoinFactorizationLongDoubleArrayWithLength Class Reference	107
8.23.1	Detailed Description	110
8.23.2	Constructor & Destructor Documentation	110
8.23.3	Member Function Documentation	110
8.24	CoinFileInput Class Reference	110
8.24.1	Detailed Description	113
8.24.2	Constructor & Destructor Documentation	113
8.24.3	Member Function Documentation	114
8.24.4	Friends And Related Function Documentation	115
8.25	CoinFileIOBase Class Reference	116
8.25.1	Detailed Description	117
8.25.2	Constructor & Destructor Documentation	118
8.26	CoinFileOutput Class Reference	118
8.26.1	Detailed Description	120
8.26.2	Member Enumeration Documentation	120
8.26.3	Constructor & Destructor Documentation	121
8.26.4	Member Function Documentation	121
8.27	CoinFirstAbsGreater_2< S, T > Class Template Reference	122
8.27.1	Detailed Description	123
8.28	CoinFirstAbsGreater_3< S, T, U > Class Template Reference	123
8.28.1	Detailed Description	123
8.29	CoinFirstAbsLess_2< S, T > Class Template Reference	123
8.29.1	Detailed Description	124
8.30	CoinFirstAbsLess_3< S, T, U > Class Template Reference	124
8.30.1	Detailed Description	124
8.31	CoinFirstGreater_2< S, T > Class Template Reference	125
8.31.1	Detailed Description	125
8.32	CoinFirstGreater_3< S, T, U > Class Template Reference	125
8.32.1	Detailed Description	125
8.33	CoinFirstLess_2< S, T > Class Template Reference	126

8.33.1 Detailed Description	126
8.34 CoinFirstLess_3< S, T, U > Class Template Reference	126
8.34.1 Detailed Description	127
8.35 CoinMpsIO::CoinHashLink Struct Reference	127
8.35.1 Detailed Description	127
8.36 CoinLpIO::CoinHashLink Struct Reference	128
8.36.1 Detailed Description	128
8.37 CoinIndexedVector Class Reference	128
8.37.1 Detailed Description	137
8.37.2 Constructor & Destructor Documentation	138
8.37.3 Member Function Documentation	138
8.37.4 Friends And Related Function Documentation	142
8.38 CoinIntArrayWithLength Class Reference	142
8.38.1 Detailed Description	145
8.38.2 Constructor & Destructor Documentation	145
8.38.3 Member Function Documentation	145
8.39 CoinLpIO Class Reference	145
8.39.1 Detailed Description	156
8.39.2 Member Function Documentation	158
8.39.3 Member Data Documentation	169
8.40 CoinMessage Class Reference	170
8.40.1 Detailed Description	172
8.40.2 Constructor & Destructor Documentation	173
8.41 CoinMessageHandler Class Reference	173
8.41.1 Detailed Description	179
8.41.2 Member Function Documentation	181
8.41.3 Friends And Related Function Documentation	186
8.41.4 Member Data Documentation	186
8.42 CoinMessages Class Reference	187
8.42.1 Detailed Description	190
8.42.2 Member Enumeration Documentation	190

8.42.3	Constructor & Destructor Documentation	190
8.42.4	Member Function Documentation	191
8.42.5	Member Data Documentation	192
8.43	CoinModel Class Reference	192
8.43.1	Detailed Description	206
8.43.2	Constructor & Destructor Documentation	207
8.43.3	Member Function Documentation	208
8.44	CoinModelHash Class Reference	216
8.44.1	Detailed Description	218
8.44.2	Constructor & Destructor Documentation	218
8.45	CoinModelHash2 Class Reference	218
8.45.1	Detailed Description	220
8.45.2	Constructor & Destructor Documentation	220
8.46	CoinModelHashLink Struct Reference	221
8.46.1	Detailed Description	221
8.47	CoinModelInfo2 Struct Reference	222
8.47.1	Detailed Description	222
8.48	CoinModelLink Class Reference	222
8.48.1	Detailed Description	224
8.48.2	Constructor & Destructor Documentation	224
8.49	CoinModelLinkedList Class Reference	225
8.49.1	Detailed Description	228
8.49.2	Constructor & Destructor Documentation	228
8.49.3	Member Function Documentation	228
8.50	CoinModelTriple Struct Reference	229
8.50.1	Detailed Description	230
8.51	CoinMpsCardReader Class Reference	230
8.51.1	Detailed Description	235
8.51.2	Member Function Documentation	235
8.52	CoinMpsIO Class Reference	235
8.52.1	Detailed Description	247

8.52.2	Member Function Documentation	247
8.52.3	Friends And Related Function Documentation	254
8.52.4	Member Data Documentation	254
8.53	CoinOneMessage Class Reference	254
8.53.1	Detailed Description	256
8.53.2	Constructor & Destructor Documentation	257
8.53.3	Member Function Documentation	257
8.54	CoinOsIFactorization Class Reference	258
8.54.1	Detailed Description	263
8.54.2	Member Function Documentation	263
8.55	CoinOtherFactorization Class Reference	264
8.55.1	Detailed Description	271
8.55.2	Member Function Documentation	271
8.55.3	Member Data Documentation	272
8.56	CoinPackedMatrix Class Reference	272
8.56.1	Detailed Description	283
8.56.2	Constructor & Destructor Documentation	284
8.56.3	Member Function Documentation	285
8.56.4	Friends And Related Function Documentation	305
8.56.5	Member Data Documentation	306
8.57	CoinPackedVector Class Reference	307
8.57.1	Detailed Description	312
8.57.2	Constructor & Destructor Documentation	314
8.57.3	Member Function Documentation	315
8.57.4	Friends And Related Function Documentation	317
8.58	CoinPackedVectorBase Class Reference	317
8.58.1	Detailed Description	321
8.58.2	Constructor & Destructor Documentation	321
8.58.3	Member Function Documentation	322
8.59	CoinPair< S, T > Struct Template Reference	324
8.59.1	Detailed Description	325

8.60	CoinParam Class Reference	325
8.60.1	Detailed Description	331
8.60.2	Member Typedef Documentation	332
8.60.3	Member Enumeration Documentation	332
8.60.4	Constructor & Destructor Documentation	333
8.60.5	Member Function Documentation	334
8.60.6	Friends And Related Function Documentation	335
8.61	CoinPartitionedVector Class Reference	340
8.61.1	Detailed Description	344
8.61.2	Constructor & Destructor Documentation	344
8.61.3	Member Function Documentation	344
8.62	CoinPostsolveMatrix Class Reference	345
8.62.1	Detailed Description	349
8.62.2	Constructor & Destructor Documentation	349
8.62.3	Member Function Documentation	350
8.62.4	Member Data Documentation	350
8.63	CoinPrePostsolveMatrix Class Reference	351
8.63.1	Detailed Description	361
8.63.2	Member Enumeration Documentation	362
8.63.3	Constructor & Destructor Documentation	362
8.63.4	Member Function Documentation	363
8.63.5	Member Data Documentation	363
8.64	CoinPresolveAction Class Reference	366
8.64.1	Detailed Description	369
8.64.2	Constructor & Destructor Documentation	370
8.64.3	Member Function Documentation	370
8.64.4	Member Data Documentation	371
8.65	CoinPresolveMatrix Class Reference	371
8.65.1	Detailed Description	381
8.65.2	Constructor & Destructor Documentation	381
8.65.3	Member Function Documentation	382

8.65.4	Friends And Related Function Documentation	385
8.65.5	Member Data Documentation	386
8.66	CoinPresolveMonitor Class Reference	388
8.66.1	Detailed Description	390
8.66.2	Constructor & Destructor Documentation	390
8.66.3	Member Function Documentation	391
8.67	CoinRelFltEq Class Reference	391
8.67.1	Detailed Description	393
8.67.2	Constructor & Destructor Documentation	393
8.68	CoinSearchTree< Comp > Class Template Reference	393
8.68.1	Detailed Description	395
8.69	CoinSearchTreeBase Class Reference	395
8.69.1	Detailed Description	397
8.69.2	Member Function Documentation	397
8.70	CoinSearchTreeCompareBest Struct Reference	397
8.70.1	Detailed Description	397
8.71	CoinSearchTreeCompareBreadth Struct Reference	397
8.71.1	Detailed Description	397
8.72	CoinSearchTreeCompareDepth Struct Reference	398
8.72.1	Detailed Description	398
8.73	CoinSearchTreeComparePreferred Struct Reference	398
8.73.1	Detailed Description	398
8.74	CoinSearchTreeManager Class Reference	399
8.74.1	Detailed Description	400
8.75	CoinSet Class Reference	400
8.75.1	Detailed Description	402
8.76	CoinShallowPackedVector Class Reference	402
8.76.1	Detailed Description	406
8.76.2	Constructor & Destructor Documentation	407
8.76.3	Member Function Documentation	408
8.76.4	Friends And Related Function Documentation	408

8.77	CoinSimpFactorization Class Reference	409
8.77.1	Detailed Description	421
8.77.2	Member Function Documentation	421
8.78	CoinSnapshot Class Reference	422
8.78.1	Detailed Description	428
8.78.2	Member Function Documentation	428
8.79	CoinSosSet Class Reference	429
8.79.1	Detailed Description	431
8.80	CoinStructuredModel Class Reference	431
8.80.1	Detailed Description	435
8.80.2	Constructor & Destructor Documentation	436
8.80.3	Member Function Documentation	436
8.81	CoinThreadRandom Class Reference	437
8.81.1	Detailed Description	439
8.81.2	Constructor & Destructor Documentation	439
8.81.3	Member Function Documentation	439
8.82	CoinTimer Class Reference	440
8.82.1	Detailed Description	441
8.83	CoinTreeNode Class Reference	442
8.83.1	Detailed Description	442
8.84	CoinTreeSiblings Class Reference	443
8.84.1	Detailed Description	444
8.85	CoinTriple< S, T, U > Class Template Reference	444
8.85.1	Detailed Description	445
8.86	CoinUnsignedIntArrayWithLength Class Reference	445
8.86.1	Detailed Description	447
8.86.2	Constructor & Destructor Documentation	447
8.86.3	Member Function Documentation	448
8.87	CoinVoidStarArrayWithLength Class Reference	448
8.87.1	Detailed Description	450
8.87.2	Constructor & Destructor Documentation	450

8.87.3	Member Function Documentation	451
8.88	CoinWarmStart Class Reference	451
8.88.1	Detailed Description	452
8.89	CoinWarmStartBasis Class Reference	452
8.89.1	Detailed Description	457
8.89.2	Member Enumeration Documentation	457
8.89.3	Constructor & Destructor Documentation	458
8.89.4	Member Function Documentation	458
8.89.5	Member Data Documentation	461
8.90	CoinWarmStartBasisDiff Class Reference	462
8.90.1	Detailed Description	464
8.90.2	Constructor & Destructor Documentation	464
8.91	CoinWarmStartDiff Class Reference	465
8.91.1	Detailed Description	466
8.92	CoinWarmStartDual Class Reference	466
8.92.1	Detailed Description	468
8.92.2	Member Function Documentation	468
8.93	CoinWarmStartDualDiff Class Reference	469
8.93.1	Detailed Description	471
8.93.2	Constructor & Destructor Documentation	471
8.94	CoinWarmStartPrimalDual Class Reference	472
8.94.1	Detailed Description	474
8.94.2	Member Function Documentation	474
8.95	CoinWarmStartPrimalDualDiff Class Reference	475
8.95.1	Detailed Description	478
8.95.2	Constructor & Destructor Documentation	478
8.95.3	Member Function Documentation	479
8.96	CoinWarmStartVector< T > Class Template Reference	479
8.96.1	Detailed Description	481
8.96.2	Member Function Documentation	481
8.97	CoinWarmStartVectorDiff< T > Class Template Reference	483

8.97.1 Detailed Description	485
8.97.2 Constructor & Destructor Documentation	485
8.97.3 Member Function Documentation	485
8.98 CoinWarmStartVectorPair< T, U > Class Template Reference	486
8.98.1 Detailed Description	487
8.99 CoinWarmStartVectorPairDiff< T, U > Class Template Reference	487
8.99.1 Detailed Description	488
8.100 CoinYacc Class Reference	489
8.100.1 Detailed Description	489
8.101 do_tighten_action Class Reference	490
8.101.1 Detailed Description	491
8.101.2 Member Function Documentation	492
8.102 doubleton_action Class Reference	492
8.102.1 Detailed Description	494
8.102.2 Member Function Documentation	494
8.103 drop_empty_cols_action Class Reference	494
8.103.1 Detailed Description	496
8.103.2 Member Function Documentation	496
8.104 drop_empty_rows_action Class Reference	497
8.104.1 Detailed Description	498
8.104.2 Member Function Documentation	498
8.105 drop_zero_coefficients_action Class Reference	499
8.105.1 Detailed Description	500
8.105.2 Member Function Documentation	501
8.106 dropped_zero Struct Reference	501
8.106.1 Detailed Description	501
8.107 dupcol_action Class Reference	502
8.107.1 Detailed Description	504
8.107.2 Member Function Documentation	504
8.108 duprow_action Class Reference	504
8.108.1 Detailed Description	506

8.108.2 Member Function Documentation	506
8.109EKKHlink Struct Reference	506
8.109.1 Detailed Description	507
8.110FactorPointers Class Reference	507
8.110.1 Detailed Description	508
8.111forcing_constraint_action Class Reference	508
8.111.1 Detailed Description	510
8.111.2 Member Function Documentation	510
8.112gubrow_action Class Reference	510
8.112.1 Detailed Description	512
8.112.2 Member Function Documentation	512
8.113implied_free_action Class Reference	512
8.113.1 Detailed Description	514
8.113.2 Member Function Documentation	514
8.114isolated_constraint_action Class Reference	515
8.114.1 Detailed Description	516
8.114.2 Member Function Documentation	516
8.115make_fixed_action Class Reference	516
8.115.1 Detailed Description	518
8.115.2 Member Function Documentation	519
8.115.3 Friends And Related Function Documentation	519
8.116presolvehlink Class Reference	520
8.116.1 Detailed Description	521
8.116.2 Friends And Related Function Documentation	521
8.117Coin::ReferencedObject Class Reference	522
8.117.1 Detailed Description	523
8.118remove_dual_action Class Reference	525
8.118.1 Detailed Description	527
8.118.2 Member Function Documentation	527
8.119remove_fixed_action Class Reference	528
8.119.1 Detailed Description	530

8.119.2 Member Function Documentation	531
8.119.3 Friends And Related Function Documentation	531
8.120slack_doubleton_action Class Reference	531
8.120.1 Detailed Description	533
8.120.2 Member Function Documentation	533
8.121slack_singleton_action Class Reference	534
8.121.1 Detailed Description	535
8.121.2 Member Function Documentation	536
8.122Coin::SmartPtr< T > Class Template Reference	536
8.122.1 Detailed Description	538
8.122.2 Constructor & Destructor Documentation	541
8.122.3 Member Function Documentation	541
8.122.4 Friends And Related Function Documentation	542
8.123subst_constraint_action Class Reference	543
8.123.1 Detailed Description	544
8.123.2 Member Function Documentation	545
8.124symrec Struct Reference	545
8.124.1 Detailed Description	546
8.125tripleton_action Class Reference	546
8.125.1 Detailed Description	549
8.125.2 Member Function Documentation	549
8.126twoxtwo_action Class Reference	549
8.126.1 Detailed Description	551
8.126.2 Member Function Documentation	551
8.127useless_constraint_action Class Reference	552
8.127.1 Detailed Description	553
8.127.2 Member Function Documentation	553
8.127.3 Friends And Related Function Documentation	553
9 File Documentation	554
9.1 CoinFloatEqual.hpp File Reference	554

9.1.1	Detailed Description	555
9.2	CoinMessage.hpp File Reference	556
9.2.1	Detailed Description	557
9.3	CoinMessageHandler.hpp File Reference	557
9.3.1	Detailed Description	559
9.3.2	Define Documentation	559
9.3.3	Function Documentation	560
9.4	CoinParam.hpp File Reference	560
9.4.1	Detailed Description	562
9.5	CoinPresolveDupcol.hpp File Reference	562
9.5.1	Detailed Description	564
9.6	CoinPresolveEmpty.hpp File Reference	564
9.6.1	Detailed Description	564
9.7	CoinPresolveForcing.hpp File Reference	564
9.7.1	Detailed Description	565
9.8	CoinPresolveImpliedFree.hpp File Reference	565
9.8.1	Detailed Description	566
9.9	CoinPresolveMatrix.hpp File Reference	566
9.9.1	Detailed Description	569
9.9.2	Variable Documentation	569
9.10	CoinPresolveSingleton.hpp File Reference	569
9.10.1	Detailed Description	569
9.11	CoinPresolveSubst.hpp File Reference	570
9.11.1	Detailed Description	570
9.12	CoinPresolveZeros.hpp File Reference	571
9.12.1	Detailed Description	571
9.13	CoinWarmStart.hpp File Reference	571
9.13.1	Detailed Description	572

1 Module Index

1.1 Modules

Here is a list of all modules:

Presolve Matrix Manipulation Functions	16
Presolve Utility Functions	23
Presolve Debug Functions	24

2 Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

CoinParamUtils (Utility functions for processing CoinParam parameters)	28
--	-----------

3 Class Index

3.1 Class Hierarchy

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

_EKKfactinfo	35
remove_fixed_action::action	36
forcing_constraint_action::action	37
tripleton_action::action	38
doubleton_action::action	39
<code>std::basic_fstream< char ></code>	
<code>std::basic_fstream< wchar_t ></code>	
<code>std::basic_ifstream< char ></code>	
<code>std::basic_ifstream< wchar_t ></code>	
<code>std::basic_ios< char ></code>	
<code>std::basic_ios< wchar_t ></code>	
<code>std::basic_iostream< char ></code>	
<code>std::basic_iostream< wchar_t ></code>	

std::basic_istream< char >	
std::basic_istream< wchar_t >	
std::basic_istringstream< char >	
std::basic_istringstream< wchar_t >	
std::basic_ofstream< char >	
std::basic_ofstream< wchar_t >	
std::basic_ostream< char >	
std::basic_ostream< wchar_t >	
std::basic_ostringstream< char >	
std::basic_ostringstream< wchar_t >	
std::basic_string< char >	
std::basic_string< wchar_t >	
std::basic_stringstream< char >	
std::basic_stringstream< wchar_t >	
BitVector128	40
CoinAbsFltEq	40
CoinArrayWithLength	45
CoinArbitraryArrayWithLength	42
CoinBigIndexArrayWithLength	54
CoinDoubleArrayWithLength	70
CoinFactorizationDoubleArrayWithLength	104
CoinFactorizationLongDoubleArrayWithLength	107
CoinIntArrayWithLength	142
CoinUnsignedIntArrayWithLength	445
CoinVoidStarArrayWithLength	448
CoinBaseModel	51
CoinModel	192
CoinStructuredModel	431
CoinBuild	57
CoinDenseVector< T >	66
CoinError	73

CoinExternalVectorFirstGreater_2 < S, T, V >	76
CoinExternalVectorFirstGreater_3 < S, T, U, V >	77
CoinExternalVectorFirstLess_2 < S, T, V >	78
CoinExternalVectorFirstLess_3 < S, T, U, V >	79
CoinFactorization	80
CoinFileIOBase	116
CoinFileInput	110
CoinFileOutput	118
CoinFirstAbsGreater_2 < S, T >	122
CoinFirstAbsGreater_3 < S, T, U >	123
CoinFirstAbsLess_2 < S, T >	123
CoinFirstAbsLess_3 < S, T, U >	124
CoinFirstGreater_2 < S, T >	125
CoinFirstGreater_3 < S, T, U >	125
CoinFirstLess_2 < S, T >	126
CoinFirstLess_3 < S, T, U >	126
CoinMpsIO::CoinHashLink	127
CoinLpIO::CoinHashLink	128
CoinIndexedVector	128
CoinPartitionedVector	340
CoinLpIO	145
CoinMessageHandler	173
CoinMessages	187
CoinMessage	170
CoinModelHash	216

CoinModelHash2	218
CoinModelHashLink	221
CoinModelInfo2	222
CoinModelLink	222
CoinModelLinkedList	225
CoinModelTriple	229
CoinMpsCardReader	230
CoinMpsIO	235
CoinOneMessage	254
CoinOtherFactorization	264
CoinDenseFactorization	61
CoinOslFactorization	258
CoinSimpFactorization	409
CoinPackedMatrix	272
CoinPackedVectorBase	317
CoinPackedVector	307
CoinShallowPackedVector	402
CoinPair< S, T >	324
CoinParam	325
CoinPrePostsolveMatrix	351
CoinPostsolveMatrix	345
CoinPresolveMatrix	371
CoinPresolveAction	366
do_tighten_action	490
doubleton_action	492

drop_empty_cols_action	494
drop_empty_rows_action	497
drop_zero_coefficients_action	499
dupcol_action	502
duprow_action	504
forcing_constraint_action	508
gubrow_action	510
implied_free_action	512
isolated_constraint_action	515
make_fixed_action	516
remove_dual_action	525
remove_fixed_action	528
slack_doubleton_action	531
slack_singleton_action	534
subst_constraint_action	543
tripleton_action	546
twoxtwo_action	549
useless_constraint_action	552
CoinPresolveMonitor	388
CoinRelFltEq	391
CoinSearchTreeBase	395
CoinSearchTree< Comp >	393
CoinSearchTreeCompareBest	397
CoinSearchTreeCompareBreadth	397
CoinSearchTreeCompareDepth	398

CoinSearchTreeComparePreferred	398
CoinSearchTreeManager	399
CoinSet	400
CoinSosSet	429
CoinSnapshot	422
CoinThreadRandom	437
CoinTimer	440
CoinTreeNode	442
CoinTreeSiblings	443
CoinTriple< S, T, U >	444
CoinWarmStart	451
CoinWarmStartBasis	452
CoinWarmStartDual	466
CoinWarmStartPrimalDual	472
CoinWarmStartVector< T >	479
CoinWarmStartVectorPair< T, U >	486
CoinWarmStartDiff	465
CoinWarmStartBasisDiff	462
CoinWarmStartDualDiff	469
CoinWarmStartPrimalDualDiff	475
CoinWarmStartVectorDiff< T >	483
CoinWarmStartVectorPairDiff< T, U >	487
CoinYacc	489
dropped_zero	501
EKKHlink	506

FactorPointers	507
presolvehlink	520
Coin::ReferencedObject	522
Coin::SmartPtr< T >	536
symrec	545

4 Class Index

4.1 Class List

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

_EKKfactinfo	35
remove_fixed_action::action (Structure to hold information necessary to reintroduce a column into the problem representation)	36
forcing_constraint_action::action	37
tripleton_action::action	38
doubleton_action::action	39
BitVector128	40
CoinAbsFltEq (Equality to an absolute tolerance)	40
CoinArbitraryArrayWithLength (Arbitrary version)	42
CoinArrayWithLength (Pointer with length in bytes)	45
CoinBaseModel	51
CoinBigIndexArrayWithLength (CoinBigIndex * version)	54
CoinBuild (In many cases it is natural to build a model by adding one row at a time)	57
CoinDenseFactorization (This deals with Factorization and Updates This is a simple dense version so other people can write a better one)	61
CoinDenseVector< T > (Dense Vector)	66

CoinDoubleArrayWithLength (Double * version)	70
CoinError (Error Class thrown by an exception)	73
CoinExternalVectorFirstGreater_2 < S, T, V > (Function operator)	76
CoinExternalVectorFirstGreater_3 < S, T, U, V > (Function operator)	77
CoinExternalVectorFirstLess_2 < S, T, V > (Function operator)	78
CoinExternalVectorFirstLess_3 < S, T, U, V > (Function operator)	79
CoinFactorization (This deals with Factorization and Updates)	80
CoinFactorizationDoubleArrayWithLength (CoinFactorizationDouble * version)	104
CoinFactorizationLongDoubleArrayWithLength (CoinFactorization- LongDouble * version)	107
CoinFileInput (Abstract base class for file input classes)	110
CoinFileIOBase (Base class for FileIO classes)	116
CoinFileOutput (Abstract base class for file output classes)	118
CoinFirstAbsGreater_2 < S, T > (Function operator)	122
CoinFirstAbsGreater_3 < S, T, U > (Function operator)	123
CoinFirstAbsLess_2 < S, T > (Function operator)	123
CoinFirstAbsLess_3 < S, T, U > (Function operator)	124
CoinFirstGreater_2 < S, T > (Function operator)	125
CoinFirstGreater_3 < S, T, U > (Function operator)	125
CoinFirstLess_2 < S, T > (Function operator)	126
CoinFirstLess_3 < S, T, U > (Function operator)	126
CoinMpsIO::CoinHashLink	127
CoinLpIO::CoinHashLink	128
CoinIndexedVector (Indexed Vector)	128
CoinIntArrayWithLength (Int * version)	142

CoinLpIO (Class to read and write Lp files)	145
CoinMessage (The standard set of Coin messages)	170
CoinMessageHandler (Base class for message handling)	173
CoinMessages (Class to hold and manipulate an array of massaged messages)	187
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)	192
CoinModelHash	216
CoinModelHash2 (For int,int hashing)	218
CoinModelHashLink (For names and hashing)	221
CoinModelInfo2 (This is a model which is made up of Coin(Structured)Model blocks)	222
CoinModelLink (This is for various structures/classes needed by Coin-Model)	222
CoinModelLinkedList	225
CoinModelTriple (For linked lists)	229
CoinMpsCardReader (Very simple code for reading MPS data)	230
CoinMpsIO (MPS IO Interface)	235
CoinOneMessage (Class for one massaged message)	254
CoinOslFactorization	258
CoinOtherFactorization (Abstract base class which also has some scalars so can be used from Dense or Simp)	264
CoinPackedMatrix (Sparse Matrix Base Class)	272
CoinPackedVector (Sparse Vector)	307
CoinPackedVectorBase (Abstract base class for various sparse vectors)	317
CoinPair< S, T > (An ordered pair)	324
CoinParam (A base class for 'keyword value' command line parameters)	325

CoinPartitionedVector	340
CoinPostsolveMatrix (Augments CoinPrePostsolveMatrix with information about the problem that is only needed during postsolve)	345
CoinPrePostsolveMatrix (Collects all the information about the problem that is needed in both presolve and postsolve)	351
CoinPresolveAction (Abstract base class of all presolve routines)	366
CoinPresolveMatrix (Augments CoinPrePostsolveMatrix with information about the problem that is only needed during presolve)	371
CoinPresolveMonitor (Monitor a row or column for modification)	388
CoinRelFltEq (Equality to a scaled tolerance)	391
CoinSearchTree< Comp >	393
CoinSearchTreeBase	395
CoinSearchTreeCompareBest (Best first search)	397
CoinSearchTreeCompareBreadth	397
CoinSearchTreeCompareDepth (Depth First Search)	398
CoinSearchTreeComparePreferred (Function objects to compare search tree nodes)	398
CoinSearchTreeManager	399
CoinSet (Very simple class for containing data on set)	400
CoinShallowPackedVector (Shallow Sparse Vector)	402
CoinSimpFactorization	409
CoinSnapshot (NON Abstract Base Class for interfacing with cut generators or branching code or)	422
CoinSosSet (Very simple class for containing SOS set)	429
CoinStructuredModel	431
CoinThreadRandom (Class for thread specific random numbers)	437
CoinTimer (This class implements a timer that also implements a tracing functionality)	440

CoinTreeNode (A class from which the real tree nodes should be derived from)	442
CoinTreeSiblings	443
CoinTriple< S, T, U >	444
CoinUnsignedIntArrayWithLength (Unsigned int * version)	445
CoinVoidStarArrayWithLength (Void * version)	448
CoinWarmStart (Abstract base class for warm start information)	451
CoinWarmStartBasis (The default COIN simplex (basis-oriented) warm start class)	452
CoinWarmStartBasisDiff (A ‘diff’ between two CoinWarmStartBasis objects)	462
CoinWarmStartDiff (Abstract base class for warm start ‘diff’ objects)	465
CoinWarmStartDual (WarmStart information that is only a dual vector)	466
CoinWarmStartDualDiff (A ‘diff’ between two CoinWarmStartDual objects)	469
CoinWarmStartPrimalDual (WarmStart information that is only a dual vector)	472
CoinWarmStartPrimalDualDiff (A ‘diff’ between two CoinWarmStart-PrimalDual objects)	475
CoinWarmStartVector< T > (WarmStart information that is only a vector)	479
CoinWarmStartVectorDiff< T > (A ‘diff’ between two CoinWarmStartVector objects)	483
CoinWarmStartVectorPair< T, U >	486
CoinWarmStartVectorPairDiff< T, U >	487
CoinYacc	489
do_tighten_action	490
doubleton_action (Solve $ax+by=c$ for y and substitute y out of the problem)	492

drop_empty_cols_action (Physically removes empty columns in presolve, and reinserts empty columns in postsolve)	494
drop_empty_rows_action (Physically removes empty rows in presolve, and reinserts empty rows in postsolve)	497
drop_zero_coefficients_action (Removal of explicit zeros)	499
dropped_zero (Tracking information for an explicit zero coefficient)	501
dupcol_action (Detect and remove duplicate columns)	502
duprow_action (Detect and remove duplicate rows)	504
EKKHlink (This deals with Factorization and Updates This is ripped off from OSL!!!!!!!!!!)	506
FactorPointers (Pointers used during factorization)	507
forcing_constraint_action (Detect and process forcing constraints and use-less constraints)	508
gubrow_action (Detect and remove entries whose sum is known)	510
implied_free_action (Detect and process implied free variables)	512
isolated_constraint_action	515
make_fixed_action (Fix a variable at a specified bound)	516
presolvehlink (Links to aid in packed matrix modification)	520
Coin::ReferencedObject (ReferencedObject class)	522
remove_dual_action (Attempt to fix variables by bounding reduced costs)	525
remove_fixed_action (Excise fixed variables from the model)	528
slack_doubleton_action (Convert an explicit bound constraint to a column bound)	531
slack_singleton_action (For variables with one entry)	534
Coin::SmartPtr< T > (Template class for Smart Pointers)	536
subst_constraint_action (Detect and process implied free variables)	543
symrec (For string evaluation)	545

tripleton_action (We are only going to do this if it does not increase number of elements?)	546
twoxtwo_action (Detect interesting 2 by 2 blocks)	549
useless_constraint_action	552

5 File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Coin_C_defines.h	??
CoinAlloc.hpp	??
CoinBuild.hpp	??
CoinDenseFactorization.hpp	??
CoinDenseVector.hpp	??
CoinDistance.hpp	??
CoinError.hpp	??
CoinFactorization.hpp	??
CoinFileIO.hpp	??
CoinFinite.hpp	??
CoinFloatEqual.hpp (Function objects for testing equality of real numbers)	554
CoinHelperFunctions.hpp	??
CoinIndexedVector.hpp	??
CoinLpIO.hpp	??
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)	556
CoinMessageHandler.hpp (This is a first attempt at a message handler)	557

CoinModel.hpp	??
CoinModelUseful.hpp	??
CoinMpsIO.hpp	??
CoinOslC.h	??
CoinOslFactorization.hpp	??
CoinPackedMatrix.hpp	??
CoinPackedVector.hpp	??
CoinPackedVectorBase.hpp	??
CoinParam.hpp (Declaration of a class for command line parameters)	560
CoinPragma.hpp	??
CoinPresolveDoubleton.hpp	??
CoinPresolveDual.hpp	??
CoinPresolveDupcol.hpp	562
CoinPresolveEmpty.hpp (Drop/reinsert empty rows/columns)	564
CoinPresolveFixed.hpp	??
CoinPresolveForcing.hpp	564
CoinPresolveImpliedFree.hpp	565
CoinPresolveIsolated.hpp	??
CoinPresolveMatrix.hpp (Declarations for CoinPresolveMatrix and Coin- PostsolveMatrix and their common base class CoinPrePostsolveMatrix)	566
CoinPresolveMonitor.hpp	??
CoinPresolvePsdebug.hpp	??
CoinPresolveSingleton.hpp	569
CoinPresolveSubst.hpp	570
CoinPresolveTighten.hpp	??

CoinPresolveTripletion.hpp	??
CoinPresolveUseless.hpp	??
CoinPresolveZeros.hpp (Drop/reintroduce explicit zeros)	571
CoinSearchTree.hpp	??
CoinShallowPackedVector.hpp	??
CoinSignal.hpp	??
CoinSimpFactorization.hpp	??
CoinSmartPtr.hpp	??
CoinSnapshot.hpp	??
CoinSort.hpp	??
CoinStructuredModel.hpp	??
CoinTime.hpp	??
CoinTypes.hpp	??
CoinUtility.hpp	??
CoinUtilsConfig.h	??
CoinWarmStart.hpp (Copyright (C) 2000 -- 2003, International Business Machines Corporation and others)	571
CoinWarmStartBasis.hpp	??
CoinWarmStartDual.hpp	??
CoinWarmStartPrimalDual.hpp	??
CoinWarmStartVector.hpp	??
config_coinutils_default.h	??
config_default.h	??

6 Module Documentation

6.1 Presolve Matrix Manipulation Functions

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

Functions

- void `CoinPrePostsolveMatrix::presolve_make_memlists` (int *lengths, `presolvevlink` *link, int n)
Initialise linked list for major vector order in bulk storage.
- bool `CoinPrePostsolveMatrix::presolve_expand_major` (CoinBigIndex *majstrts, double *majels, int *minndx, int *majlens, `presolvevlink` *majlinks, int nmaj, int k)
Make sure a major-dimension vector k has room for one more coefficient.
- bool `CoinPrePostsolveMatrix::presolve_expand_col` (CoinBigIndex *mcstrt, double *colels, int *hrow, int *hincol, `presolvevlink` *clink, int ncols, int colx)
Make sure a column (colx) in a column-major matrix has room for one more coefficient.
- bool `CoinPrePostsolveMatrix::presolve_expand_row` (CoinBigIndex *mrstrt, double *rowels, int *hcol, int *hinrow, `presolvevlink` *rlink, int nrows, int rowx)
Make sure a row (rowx) in a row-major matrix has room for one more coefficient.
- CoinBigIndex `CoinPrePostsolveMatrix::presolve_find_minor` (int tgt, CoinBigIndex ks, CoinBigIndex ke, const int *minndx)
Find position of a minor index in a major vector.
- CoinBigIndex `CoinPrePostsolveMatrix::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 `CoinPostsolveMatrix::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 `CoinPrePostsolveMatrix::presolve_find_minor1` (int tgt, CoinBigIndex ks, CoinBigIndex ke, const int *minndx)
Find position of a minor index in a major vector.

- CoinBigIndex [CoinPrePostsolveMatrix::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 [CoinPrePostsolveMatrix::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 [CoinPostsolveMatrix::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 [CoinPostsolveMatrix::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 [CoinPostsolveMatrix::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 [CoinPostsolveMatrix::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 [CoinPrePostsolveMatrix::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 [CoinPrePostsolveMatrix::presolve_delete_many_from_major](#) (int majndx, char *marked, const CoinBigIndex *majstrts, int *majlens, int *minndx, double *els)
Delete marked entries.
- void [CoinPrePostsolveMatrix::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 [CoinPrePostsolveMatrix::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 `CoinPostsolveMatrix::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 `CoinPostsolveMatrix::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.

6.1.1 Detailed Description

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

6.1.2 Function Documentation

- 6.1.2.1** `bool presolve_expand_major (CoinBigIndex * majstrts, double * majels, int * minndx, int * majlens, presolvehlink * majlinks, int nmaj, int k)` [**related**, **inherited**]

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`

- 6.1.2.2** `CoinBigIndex presolve_find_minor (int tgt, CoinBigIndex ks, CoinBigIndex ke, const int * minndx)` [**related**, **inherited**]

Find position of a minor index in a major vector.

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 wrappers `presolve_find_row` and `presolve_find_col`.

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

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

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`.

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

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`.

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

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`.

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

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`.

6.1.2.7 `CoinBigIndex presolve_find_col1 (int col, CoinBigIndex krs, CoinBigIndex kre, const int * hcol) [related, inherited]`

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`.

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

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`.

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

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`.

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

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`.

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

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`.

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

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 (`minndx`) 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`.

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

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`.

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

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`.

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

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`.

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

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 (`minndx`) 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.

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

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.

6.2 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.

6.2.1 Detailed Description

Utilities used by multiple presolve transform objects.

6.2.2 Function Documentation

6.2.2.1 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.

6.3 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 `CoinPresolveMatrix::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 `CoinPresolveMatrix::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 `CoinPresolveMatrix::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 `CoinPresolveMatrix::presolve_consistent` (const `CoinPresolveMatrix *preObj`, bool chkvals=true)
Checks for equivalence of the column- and row-major matrices.
- void `CoinPostsolveMatrix::presolve_check_threads` (const `CoinPostsolveMatrix *obj`)
Checks that column threads agree with column lengths.
- void `CoinPostsolveMatrix::presolve_check_free_list` (const `CoinPostsolveMatrix *obj`, bool chkElemCnt=false)
Checks the free list.
- void `CoinPostsolveMatrix::presolve_check_reduced_costs` (const `CoinPostsolveMatrix *obj`)
Check stored reduced costs for accuracy and consistency with variable status.
- void `CoinPostsolveMatrix::presolve_check_duals` (const `CoinPostsolveMatrix *postObj`)
Check the dual variables for consistency with row activity.
- void `CoinPresolveMatrix::presolve_check_sol` (const `CoinPresolveMatrix *preObj`, int chkColSol=2, int chkRowAct=1, int chkStatus=1)
Check primal solution and architectural variable status.
- void `CoinPostsolveMatrix::presolve_check_sol` (const `CoinPostsolveMatrix *postObj`, int chkColSol=2, int chkRowAct=2, int chkStatus=1)
Check primal solution and architectural variable status.

- void `CoinPresolveMatrix::presolve_check_nbasic` (const `CoinPresolveMatrix *preObj`)
Check for the proper number of basic variables.
- void `CoinPostsolveMatrix::presolve_check_nbasic` (const `CoinPostsolveMatrix *postObj`)
Check for the proper number of basic variables.

6.3.1 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.

6.3.2 Function Documentation

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

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.

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

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.

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

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.

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

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.

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

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.

6.3.2.6 `void presolve_check_reduced_costs (const CoinPostsolveMatrix * obj) [related, inherited]`

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_](#).

6.3.2.7 `void presolve_check_duals (const CoinPostsolveMatrix * postObj)`
`[related, inherited]`

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).

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

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.

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

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 Namespace Documentation

7.1 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.

7.1.1 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.

7.1.2 Function Documentation

7.1.2.1 void CoinParamUtils::setInputSrc (FILE * src)

Take command input from the file specified by src.

Use stdin for src to specify interactive prompting for commands.

7.1.2.2 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.

7.1.2.3 `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.

7.1.2.4 `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.

7.1.2.5 `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).

7.1.2.6 `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.

7.1.2.7 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)

7.1.2.8 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.

7.1.2.9 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.

7.1.2.10 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.

7.1.2.11 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.

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

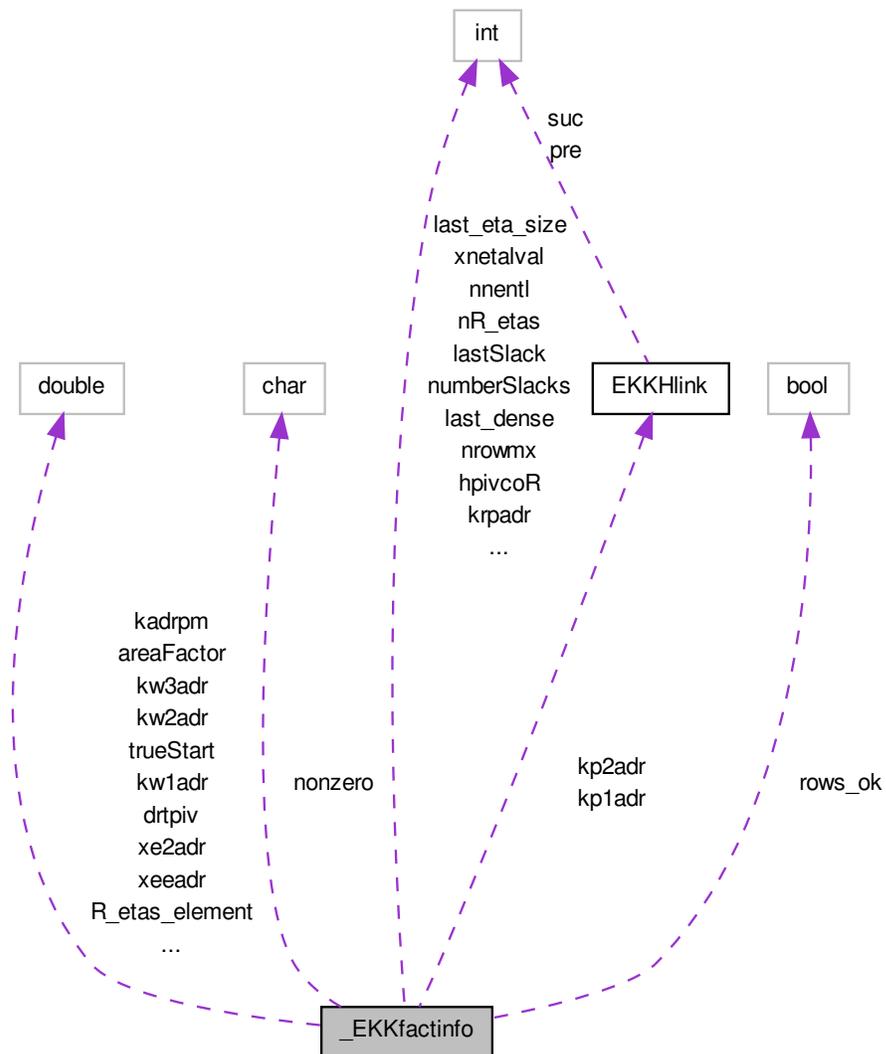
Utility routine to print help messages for one or more parameters.

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.

8 Class Documentation

8.1 `_EKKfactinfo` Struct Reference

Collaboration diagram for `_EKKfactinfo`:



8.1.1 Detailed Description

Definition at line 29 of file CoinOsIFactorization.hpp.

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

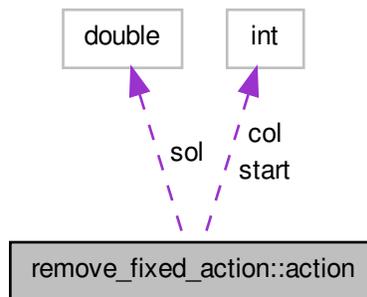
- CoinOsIFactorization.hpp

8.2 remove_fixed_action::action Struct Reference

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

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

Collaboration diagram for remove_fixed_action::action:



Public Attributes

- int `col`
column index of variable
- int `start`
start of coefficients in `coels_` and `colrows_`
- double `sol`
value of variable

8.2.1 Detailed Description

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

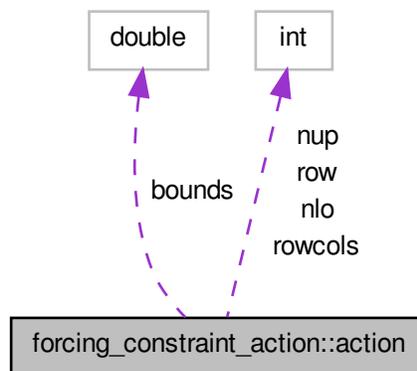
Definition at line 30 of file CoinPresolveFixed.hpp.

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

- [CoinPresolveFixed.hpp](#)

8.3 forcing_constraint_action::action Struct Reference

Collaboration diagram for forcing_constraint_action::action:



8.3.1 Detailed Description

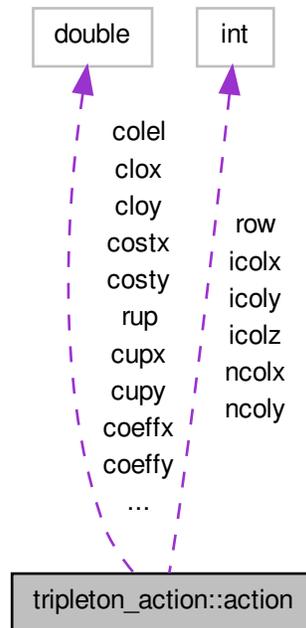
Definition at line 32 of file CoinPresolveForcing.hpp.

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

- [CoinPresolveForcing.hpp](#)

8.4 tripleton_action::action Struct Reference

Collaboration diagram for tripleton_action::action:



8.4.1 Detailed Description

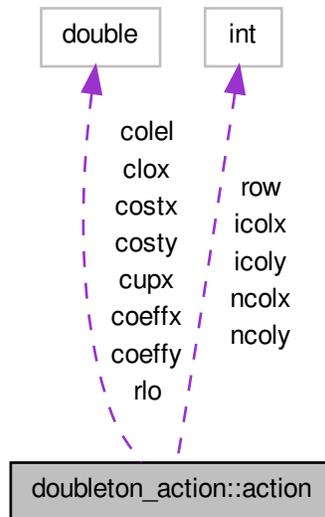
Definition at line 17 of file `CoinPresolveTripleton.hpp`.

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

- `CoinPresolveTripleton.hpp`

8.5 doubleton_action::action Struct Reference

Collaboration diagram for doubleton_action::action:



8.5.1 Detailed Description

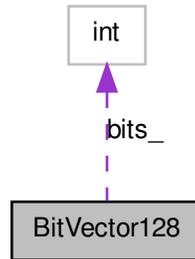
Definition at line 28 of file `CoinPresolveDoubleton.hpp`.

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

- `CoinPresolveDoubleton.hpp`

8.6 BitVector128 Class Reference

Collaboration diagram for BitVector128:



8.6.1 Detailed Description

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

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

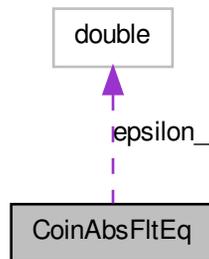
- `CoinSearchTree.hpp`

8.7 CoinAbsFltEq Class Reference

Equality to an absolute tolerance.

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

Collaboration diagram for CoinAbsFltEq:



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.

8.7.1 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.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 CoinAbsFltEq::CoinAbsFltEq() [inline]

Default constructor.

Default tolerance is 1.0e-10.

Definition at line 66 of file CoinFloatEqual.hpp.

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

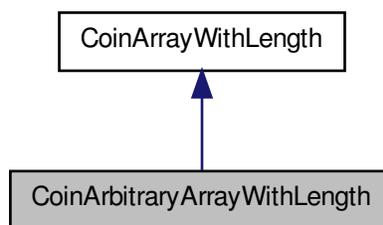
- [CoinFloatEqual.hpp](#)

8.8 CoinArbitraryArrayWithLength Class Reference

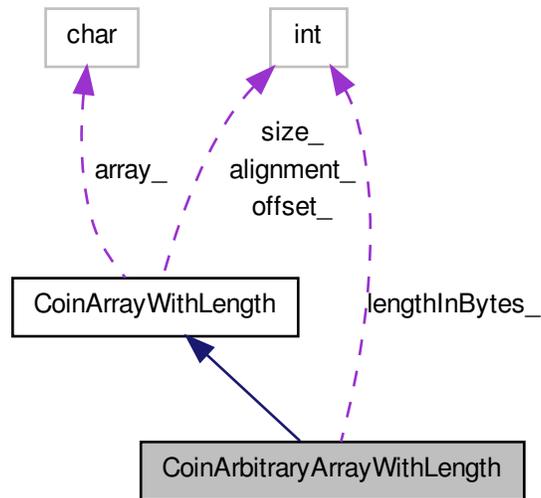
arbitrary version

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

Inheritance diagram for CoinArbitraryArrayWithLength:



Collaboration 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.

8.8.1 Detailed Description

arbitrary version

Definition at line 995 of file CoinIndexedVector.hpp.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 CoinArbitraryArrayWithLength::CoinArbitraryArrayWithLength (const CoinArbitraryArrayWithLength & *rhs*) `[inline]`

Copy constructor.

Definition at line 1038 of file CoinIndexedVector.hpp.

8.8.3 Member Function Documentation

8.8.3.1 CoinArbitraryArrayWithLength& CoinArbitraryArrayWithLength::operator= (const CoinArbitraryArrayWithLength & *rhs*) `[inline]`

Assignment operator.

Definition at line 1044 of file CoinIndexedVector.hpp.

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

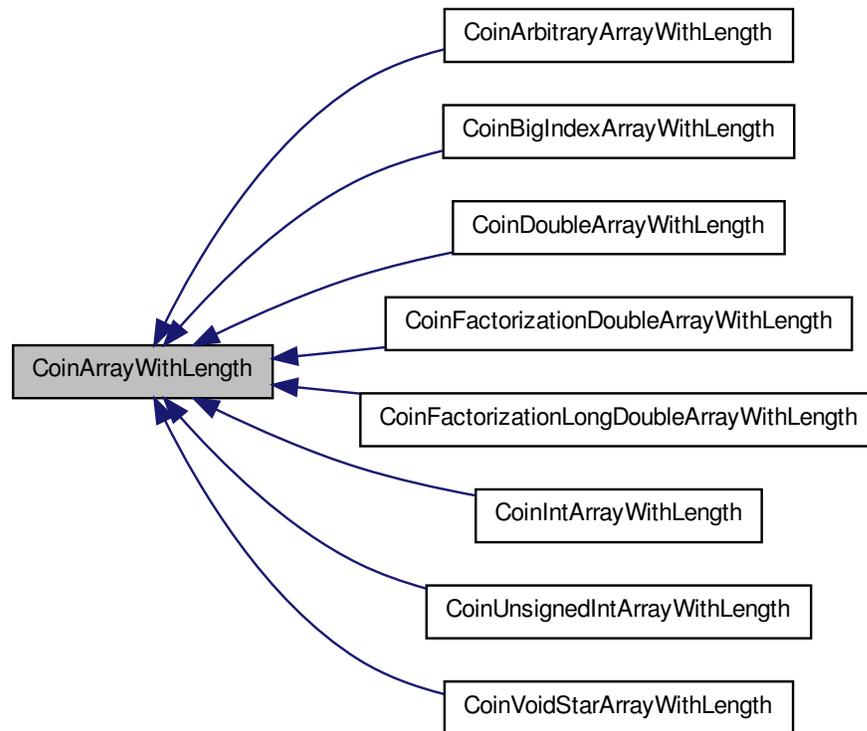
- CoinIndexedVector.hpp

8.9 CoinArrayWithLength Class Reference

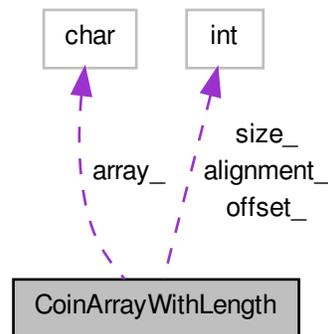
Pointer with length in bytes.

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

Inheritance diagram for CoinArrayWithLength:



Collaboration 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).

8.9.1 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.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 CoinArrayWithLength::CoinArrayWithLength (const CoinArrayWithLength & *rhs*)

Copy constructor.

8.9.3 Member Function Documentation

8.9.3.1 CoinArrayWithLength& CoinArrayWithLength::operator= (const CoinArrayWithLength & *rhs*)

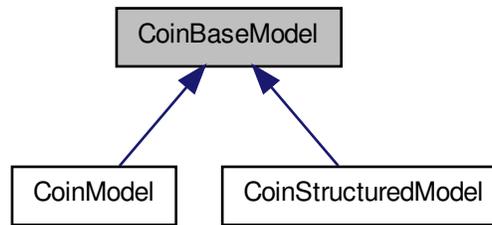
Assignment operator.

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

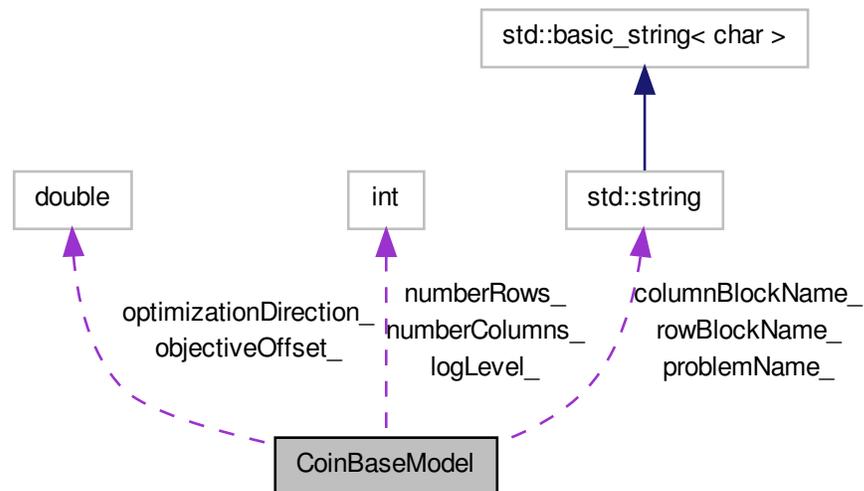
- CoinIndexedVector.hpp

8.10 CoinBaseModel Class Reference

Inheritance diagram for CoinBaseModel:



Collaboration 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.

8.10.1 Detailed Description

Definition at line 12 of file CoinModel.hpp.

8.10.2 Member Data Documentation

8.10.2.1 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 108 of file CoinModel.hpp.

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

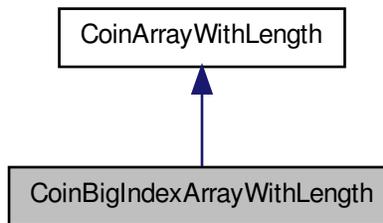
- CoinModel.hpp

8.11 CoinBigIndexArrayWithLength Class Reference

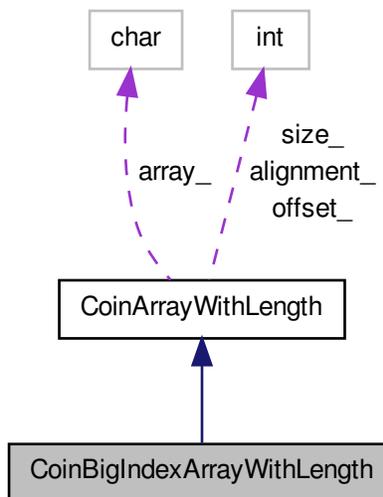
CoinBigIndex * version.

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

Inheritance diagram for CoinBigIndexArrayWithLength:



Collaboration 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.

8.11.1 Detailed Description

CoinBigIndex * version.

Definition at line 833 of file CoinIndexedVector.hpp.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 CoinBigIndexArrayWithLength::CoinBigIndexArrayWithLength (const CoinBigIndexArrayWithLength & *rhs*) [inline]

Copy constructor.

Definition at line 875 of file CoinIndexedVector.hpp.

8.11.3 Member Function Documentation

8.11.3.1 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:

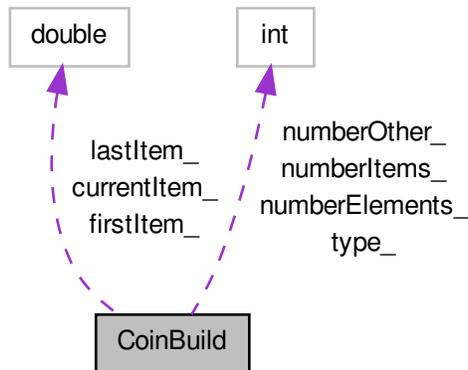
- CoinIndexedVector.hpp

8.12 CoinBuild Class Reference

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

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

Collaboration diagram for CoinBuild:



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 &)`

`=`

8.12.1 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`.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 `CoinBuild::CoinBuild ()`

Default constructor.

8.12.2.2 `CoinBuild::CoinBuild (int type)`

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

8.12.2.3 `CoinBuild::CoinBuild (const CoinBuild &)`

The copy constructor.

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

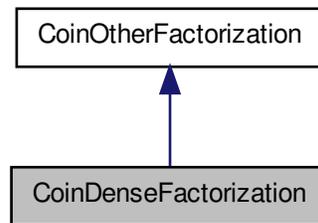
- CoinBuild.hpp

8.13 CoinDenseFactorization Class Reference

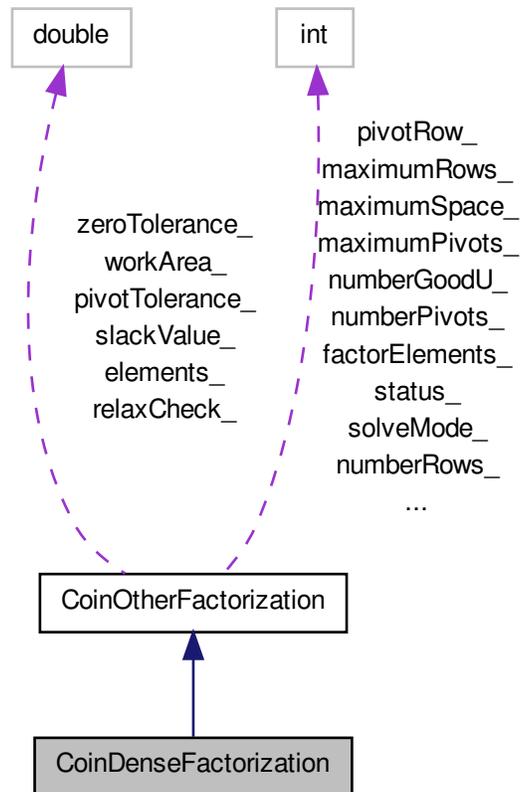
This deals with Factorization and Updates This is a simple dense version so other people can write a better one.

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

Inheritance diagram for CoinDenseFactorization:



Collaboration diagram for CoinDenseFactorization:



Public Member Functions

- void [gutsOfDestructor](#) ()
The real work of desstructor.
- 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 numberOfRows, int numberOfColumns, [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 numberOfColumns)
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.

8.13.1 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.

8.13.2 Member Function Documentation

8.13.2.1 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](#).

8.13.2.2 `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.

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

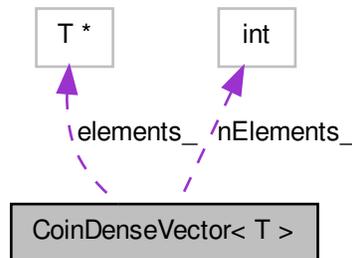
- CoinDenseFactorization.hpp

8.14 CoinDenseVector< T > Class Template Reference

Dense Vector.

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

Collaboration diagram for CoinDenseVector< T >:



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.

8.14.1 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.

8.14.2 Member Function Documentation

8.14.2.1 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.

8.14.2.2 `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

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

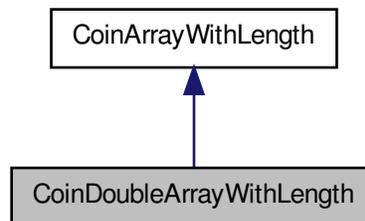
- CoinDenseVector.hpp

8.15 CoinDoubleArrayWithLength Class Reference

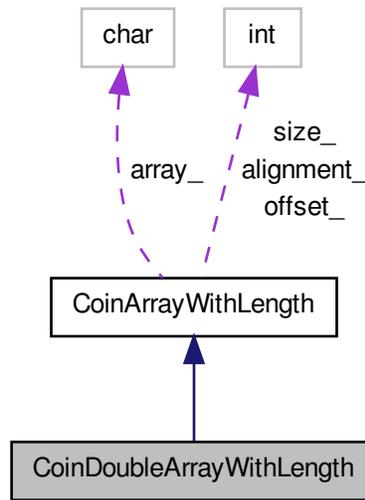
double * version

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

Inheritance diagram for CoinDoubleArrayWithLength:



Collaboration 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.

8.15.1 Detailed Description

`double * version`

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

8.15.2 Constructor & Destructor Documentation

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

Copy constructor.

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

8.15.3 Member Function Documentation

8.15.3.1 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:

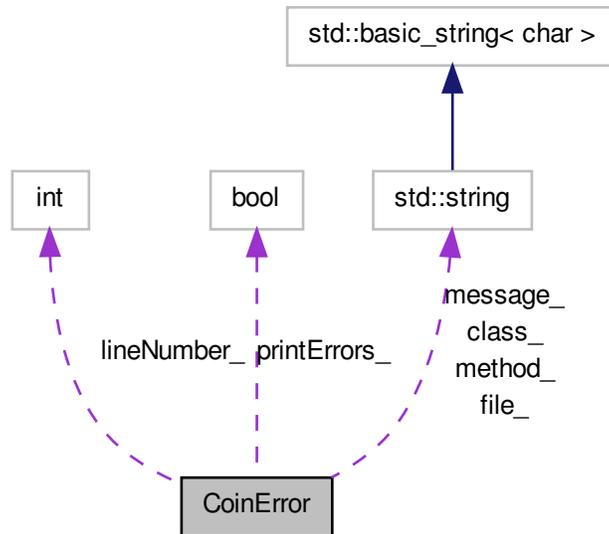
- CoinIndexedVector.hpp

8.16 CoinError Class Reference

Error Class thrown by an exception.

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

Collaboration diagram for CoinError:



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.

8.16.1 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

8.17 CoinExternalVectorFirstGreater_2< S, T, V > Class Template Reference77

- line number

For asserts class=> optional hint

Definition at line 42 of file CoinError.hpp.

8.16.2 Friends And Related Function Documentation

8.16.2.1 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.

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

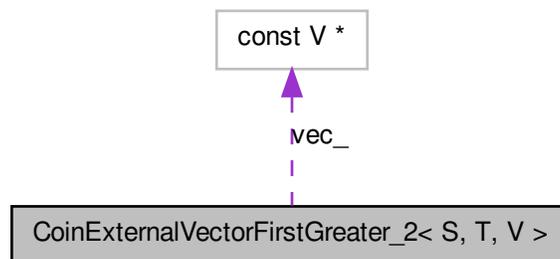
- CoinError.hpp

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

Function operator.

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

Collaboration diagram for CoinExternalVectorFirstGreater_2< S, T, V >:



8.17.1 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]` (i.e., decreasing wrt. `vec`). 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`.

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

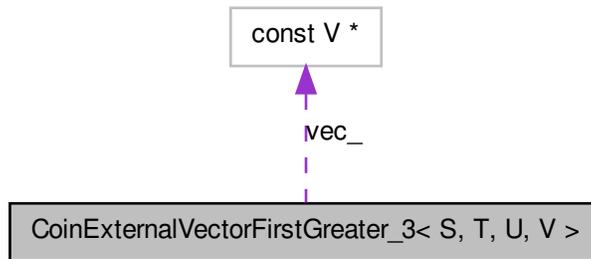
- `CoinSort.hpp`

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

Function operator.

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

Collaboration diagram for `CoinExternalVectorFirstGreater_3< S, T, U, V >`:



8.18.1 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 > vec[t2.first]` (i.e., decreasing wrt. `vec`). 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`.

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

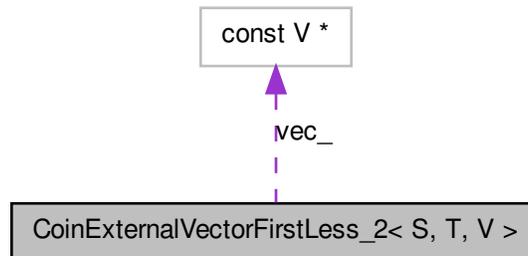
- `CoinSort.hpp`

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

Function operator.

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

Collaboration diagram for `CoinExternalVectorFirstLess_2< S, T, V >`:



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

8.19.1 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 < vec[t2.first]` (i.e., increasing wrt. `vec`). 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`.

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

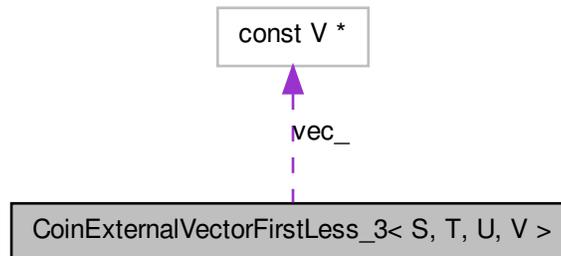
- `CoinSort.hpp`

8.20 **CoinExternalVectorFirstLess_3< S, T, U, V > Class Template Reference**

Function operator.

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

Collaboration diagram for `CoinExternalVectorFirstLess_3< S, T, U, V >`:



8.20.1 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]` (i.e., increasing wrt. `vec`). 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`.

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

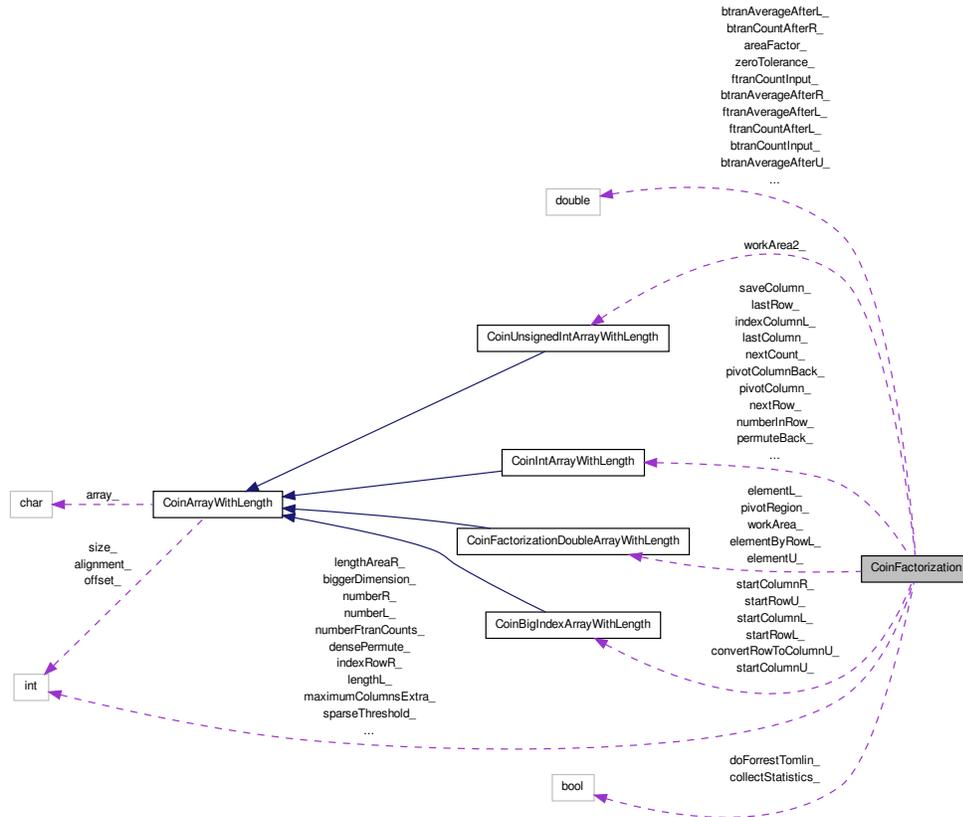
- `CoinSort.hpp`

8.21 CoinFactorization Class Reference

This deals with Factorization and Updates.

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

Collaboration diagram for CoinFactorization:



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 rowIsBasic[], int columnIsBasic[], double areaFactor=0.0)
When part of LP - given by basic variables.
- int `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.
- int `factorizePart1` (int numberOfRows, 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 `numberRowsExtra` () 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 regionSparse2 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
– taken out as always 2*
- 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.

used by factorization

- void [getAreas](#) (int numberOfRows, 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 `getColumnSpaceIterateR` (int iColumn, double value, int iRow)
getColumnSpaceIterateR.
- CoinBigIndex `getColumnSpaceIterate` (int iColumn, double value, int iRow)
getColumnSpaceIterate.
- 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 `getRowSpaceIterate` (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)

8.21.1 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`.

8.21.2 Member Function Documentation

8.21.2.1 `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`

8.21.2.2 `int CoinFactorization::factorize (const CoinPackedMatrix & matrix, int rowIsBasic[], int columnIsBasic[], 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

8.21.2.3 `int CoinFactorization::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.

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

8.21.2.4 `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

8.21.2.5 `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

8.21.2.6 `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 regionSparse starts as zero and is zero at end.

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

8.21.2.7 `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

8.21.2.8 `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

8.21.2.9 `int CoinFactorization::addColumn (CoinBigIndex numberElements, int indicesRow[], double elements[])`

Adds one Column to basis, can increase size of basis.

Returns rank

8.21.2.10 `int CoinFactorization::addRow (CoinBigIndex numberElements, int indicesColumn[], double elements[])`

Adds one Row to basis, can increase size of basis.

Returns rank

8.21.2.11 void CoinFactorization::preProcess (int *state*, int *possibleDuplicates* = -1)

PreProcesses raw triplet data.

state is 0 - triplets, 1 - some counts etc , 2 - ..

8.21.2.12 bool CoinFactorization::getColumnSpaceIterateR (int *iColumn*, double *value*, int *iRow*) [protected]

getColumnSpaceIterateR.

Gets space for one extra R element in Column may have to do compression (returns true) also moves existing vector

8.21.2.13 CoinBigIndex CoinFactorization::getColumnSpaceIterate (int *iColumn*, double *value*, int *iRow*) [protected]

getColumnSpaceIterate.

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)

8.21.2.14 void CoinFactorization::updateColumnRFT (CoinIndexedVector * *region*, int * *indexIn*) [protected]

Updates part of column (FTRANR) with FT update.

Also stores update after L and R

8.21.2.15 void CoinFactorization::updateColumnTransposeU (CoinIndexedVector * *region*, int *smallestIndex*) const [protected]

Updates part of column transpose (BTRANU), assumes index is sorted i.e. region is correct

8.21.2.16 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

8.21.2.17 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

8.21.2.18 void CoinFactorization::updateColumnTransposeUSparse (
CoinIndexedVector * *region*) const [protected]

Updates part of column transpose (BTRANU) when sparse, assumes index is sorted
i.e.

region is correct

8.21.2.19 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

8.21.2.20 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)

8.21.3 Member Data Documentation

8.21.3.1 `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.

8.21.3.2 `CoinBigIndex CoinFactorization::lengthU_` `[protected]`

Base of U is always 0.

Length of U

Definition at line 1314 of file CoinFactorization.hpp.

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

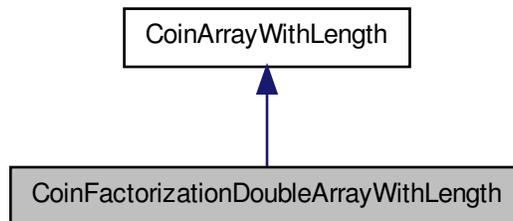
- CoinFactorization.hpp

8.22 CoinFactorizationDoubleArrayWithLength Class Reference

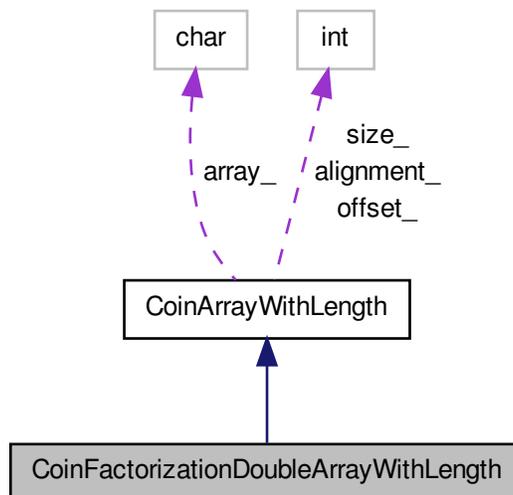
CoinFactorizationDouble * version.

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

Inheritance diagram for CoinFactorizationDoubleArrayWithLength:



Collaboration 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.

8.22.1 Detailed Description

CoinFactorizationDouble * version.

Definition at line 671 of file CoinIndexedVector.hpp.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 CoinFactorizationDoubleArrayWithLength::CoinFactorizationDoubleArrayWithLength (const CoinFactorizationDoubleArrayWithLength & rhs) [inline]

Copy constructor.

Definition at line 713 of file CoinIndexedVector.hpp.

8.22.3 Member Function Documentation

8.22.3.1 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:

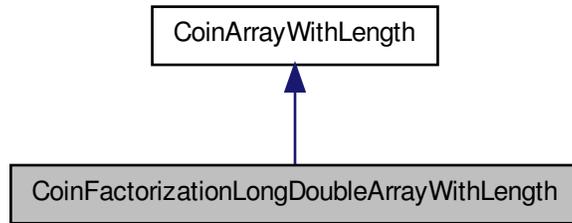
- CoinIndexedVector.hpp

8.23 CoinFactorizationLongDoubleArrayWithLength Class Reference

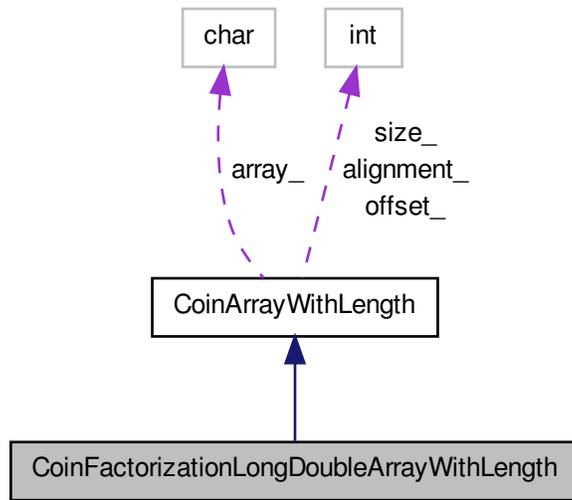
CoinFactorizationLongDouble * version.

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

Inheritance diagram for CoinFactorizationLongDoubleArrayWithLength:



Collaboration 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.

8.23.1 Detailed Description

CoinFactorizationLongDouble * version.

Definition at line 725 of file CoinIndexedVector.hpp.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 CoinFactorizationLongDoubleArrayWithLength::CoinFactorizationLongDoubleArrayWithLength (const CoinFactorizationLongDoubleArrayWithLength & rhs) [inline]

Copy constructor.

Definition at line 767 of file CoinIndexedVector.hpp.

8.23.3 Member Function Documentation

8.23.3.1 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:

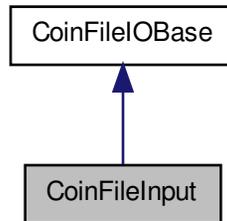
- CoinIndexedVector.hpp

8.24 CoinFileInput Class Reference

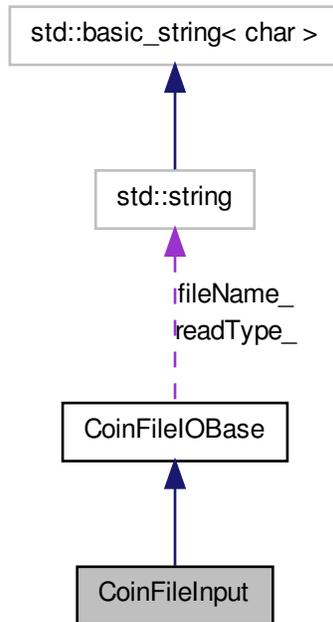
Abstract base class for file input classes.

```
#include <CoinFileIO.hpp>
```

Inheritance diagram for CoinFileInput:



Collaboration 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.

8.24.1 Detailed Description

Abstract base class for file input classes.

Definition at line 37 of file CoinFileIO.hpp.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 CoinFileInput::CoinFileInput (const std::string & fileName)

Constructor (don't use this, use the create method instead).

Parameters

fileName The name of the file used by this object.

8.24.3 Member Function Documentation

8.24.3.1 `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

fileName The file that should be read.

8.24.3.2 `virtual int CoinFileInput::read (void * buffer, int size) [pure virtual]`

Read a block of data from the file, similar to fread.

Parameters

buffer Address of a buffer to store the data into.

size Number of bytes to read (buffer should be large enough).

Returns

Number of bytes read.

8.24.3.3 `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 `''`. If reading ends due to an encountered newline, the `'`

`'` is put into the buffer, before the `''` is appended.

Parameters

buffer The buffer to put the string into.

size The size of the buffer in characters.

Returns

buffer on success, or 0 if no characters have been read.

8.24.4 Friends And Related Function Documentation

8.24.4.1 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)

8.24.4.2 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:

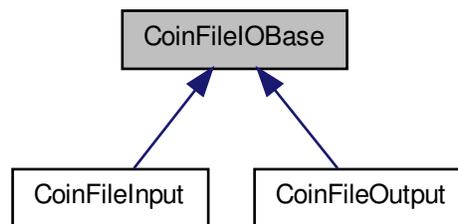
- CoinFileIO.hpp

8.25 CoinFileIOBase Class Reference

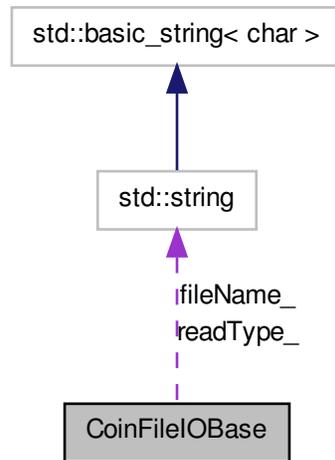
Base class for FileIO classes.

```
#include <CoinFileIO.hpp>
```

Inheritance diagram for CoinFileIOBase:



Collaboration 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.

8.25.1 Detailed Description

Base class for FileIO classes.

Definition at line 11 of file CoinFileIO.hpp.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 CoinFileIOBase::CoinFileIOBase (const std::string & *fileName*)

Constructor.

Parameters

fileName The name of the file used by this object.

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

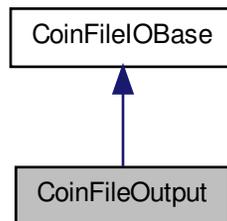
- CoinFileIO.hpp

8.26 CoinFileOutput Class Reference

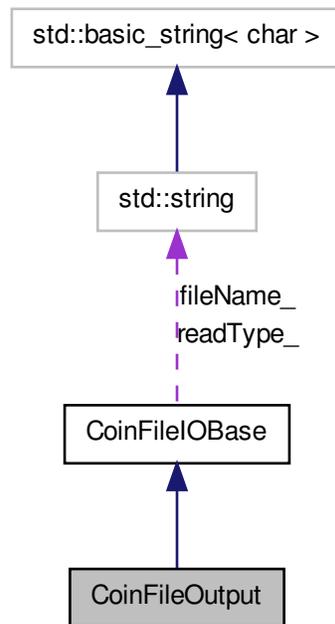
Abstract base class for file output classes.

```
#include <CoinFileIO.hpp>
```

Inheritance diagram for CoinFileOutput:



Collaboration 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.

8.26.1 Detailed Description

Abstract base class for file output classes.

Definition at line 80 of file CoinFileIO.hpp.

8.26.2 Member Enumeration Documentation

8.26.2.1 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.

8.26.3 Constructor & Destructor Documentation

8.26.3.1 CoinFileOutput::CoinFileOutput (const std::string & *fileName*)

Constructor (don't use this, use the create method instead).

Parameters

fileName The name of the file used by this object.

8.26.4 Member Function Documentation

8.26.4.1 static bool CoinFileOutput::compressionSupported (Compression *compression*) [static]

Returns whether the specified compression method is supported (i.e. was compiled into COIN).

8.26.4.2 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

fileName The file that should be read.

compression Compression method used.

8.26.4.3 virtual int CoinFileOutput::write (const void * *buffer*, int *size*) [pure virtual]

Write a block of data to the file, similar to fwrite.

Parameters

buffer Address of a buffer containing the data to be written.

size Number of bytes to write.

Returns

Number of bytes written.

8.26.4.4 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 ” is not written to the file. The default implementation determines the length of the string and calls write on it.

Parameters

s The zero terminated string to be written.

Returns

true on success, false on error.

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

- CoinFileIO.hpp

8.27 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.

8.27.1 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`.

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

- `CoinSort.hpp`

8.28 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.

8.28.1 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`.

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

- `CoinSort.hpp`

8.29 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.

8.29.1 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`.

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

- `CoinSort.hpp`

8.30 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.

8.30.1 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`.

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

- `CoinSort.hpp`

8.31 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.

8.31.1 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.

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

- CoinSort.hpp

8.32 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.

8.32.1 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.

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

- CoinSort.hpp

8.33 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.

8.33.1 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.

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

- CoinSort.hpp

8.34 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.

8.34.1 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).

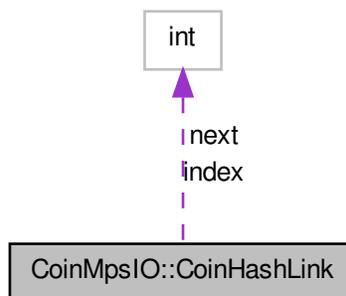
Definition at line 478 of file `CoinSort.hpp`.

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

- `CoinSort.hpp`

8.35 CoinMpsIO::CoinHashLink Struct Reference

Collaboration diagram for `CoinMpsIO::CoinHashLink`:



8.35.1 Detailed Description

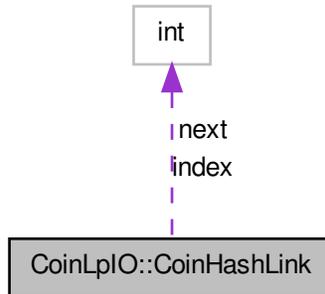
Definition at line 897 of file `CoinMpsIO.hpp`.

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

- `CoinMpsIO.hpp`

8.36 CoinLpIO::CoinHashLink Struct Reference

Collaboration diagram for CoinLpIO::CoinHashLink:



8.36.1 Detailed Description

Definition at line 589 of file `CoinLpIO.hpp`.

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

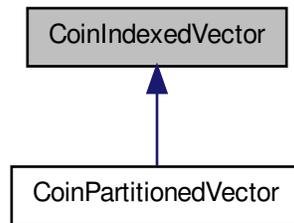
- `CoinLpIO.hpp`

8.37 CoinIndexedVector Class Reference

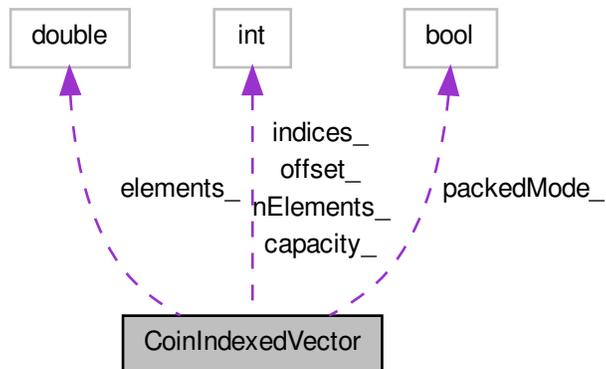
Indexed Vector.

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

Inheritance diagram for CoinIndexedVector:



Collaboration 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
 - $< tolerance \Rightarrow 0.0$
- 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 \geq 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 ($0.0/0.0 \Rightarrow 0.0$) (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.

8.37.1 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 x1
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.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 CoinIndexedVector::CoinIndexedVector (int *size*, const double * *elements*)

Alternate Constructors - construct full storage with indices 0 through size-1.

8.37.2.2 CoinIndexedVector::CoinIndexedVector (const CoinIndexedVector &)

Copy constructor.

8.37.2.3 CoinIndexedVector::CoinIndexedVector (const CoinPackedVectorBase & *rhs*)

Copy constructor *from a PackedVectorBase*.

8.37.3 Member Function Documentation

8.37.3.1 int* CoinIndexedVector::getIndices () [inline]

Get element values.

Get indices of elements

Definition at line 117 of file CoinIndexedVector.hpp.

8.37.3.2 `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.

8.37.3.3 `CoinIndexedVector& CoinIndexedVector::operator= (const CoinIndexedVector &)`

Assignment operator.

8.37.3.4 `CoinIndexedVector& CoinIndexedVector::operator= (const CoinPackedVectorBase & rhs)`

Assignment operator from a [CoinPackedVectorBase](#).

NOTE: This assumes no duplicates

8.37.3.5 `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

8.37.3.6 `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.

8.37.3.7 void CoinIndexedVector::returnVector ()

Return ownership of the arguments to this vector.

State after is empty .

8.37.3.8 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

8.37.3.9 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

**8.37.3.10 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.

8.37.3.11 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.

8.37.3.12 void CoinIndexedVector::zero (int *index*) [inline]

Makes nonzero tiny.

This version does no checking

Definition at line 243 of file CoinIndexedVector.hpp.

8.37.3.13 int CoinIndexedVector::scan (double *tolerance*)

Scan dense region and set up indices (returns number found).

Only ones \geq tolerance

8.37.3.14 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

8.37.3.15 bool CoinIndexedVector::operator== (const CoinPackedVectorBase & *rhs*) const

Equal.

Returns true if vectors have same length and corresponding element of each vector is equal.

8.37.3.16 bool CoinIndexedVector::operator== (const CoinIndexedVector & *rhs*) const

Equal.

Returns true if vectors have same length and corresponding element of each vector is equal.

8.37.3.17 void CoinIndexedVector::sort () [inline]

Sort the indexed storage vector (increasing indices).

Reimplemented in [CoinPartitionedVector](#).

Definition at line 354 of file CoinIndexedVector.hpp.

8.37.3.18 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.

Reimplemented in [CoinPartitionedVector](#).

8.37.4 Friends And Related Function Documentation

8.37.4.1 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.

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

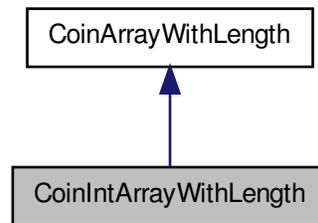
- CoinIndexedVector.hpp

8.38 CoinIntArrayWithLength Class Reference

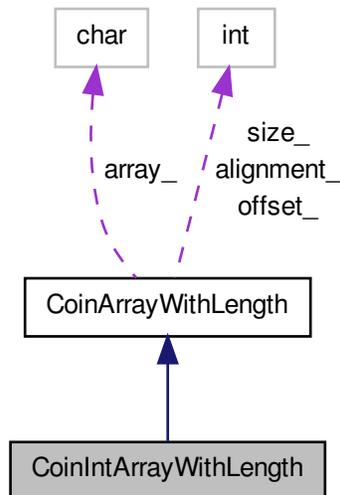
```
int * version
```

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

Inheritance diagram for CoinIntArrayWithLength:



Collaboration 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.

8.38.1 Detailed Description

int * version

Definition at line 779 of file CoinIndexedVector.hpp.

8.38.2 Constructor & Destructor Documentation

8.38.2.1 CoinIntArrayWithLength::CoinIntArrayWithLength (const CoinIntArrayWithLength & rhs) [inline]

Copy constructor.

Definition at line 821 of file CoinIndexedVector.hpp.

8.38.3 Member Function Documentation

8.38.3.1 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:

- CoinIndexedVector.hpp

8.39 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.

8.39.1 Detailed Description

Class to read and write Lp files. Lp file format:

/ this is a comment

\ this too

Min

obj: $x_0 + x_1 + 3 x_2 - 4.5 x_{yr} + 1$

s.t.

cons1: $x_0 - x_2 - 2.3 x_4 \leq 4.2$ / 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

x_0

Generals

x_1 x_{yr}

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.

8.39.2 Member Function Documentation

8.39.2.1 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.

8.39.2.2 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

8.39.2.3 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`

8.39.2.4 `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.

8.39.2.5 `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.

8.39.2.6 `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.

8.39.2.7 `char const* const* CoinLpIO::getRowNames () const`

Get pointer to array[getNumRows()+1] of row names, including objective function name as last entry.

8.39.2.8 `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.

8.39.2.9 `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.

8.39.2.10 `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.

8.39.2.11 `int CoinLpIO::columnIndex (const char * name) const`

Return the index for the specified column name.

Return -1 if the name is not found.

8.39.2.12 `void CoinLpIO::setInfinity (const double)`

Set infinity.

Any number larger is considered infinity. Default: DBL_MAX

8.39.2.13 void CoinLpIO::setEpsilon (const double)

Set epsilon.

Default: 1e-5.

8.39.2.14 void CoinLpIO::setNumberAcross (const int)

Set numberAcross.

Default: 10.

8.39.2.15 void CoinLpIO::setDecimals (const int)

Set decimals.

Default: 5

8.39.2.16 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.

8.39.2.17 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.

8.39.2.18 `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.

8.39.2.19 `void CoinLpIO::setDefaultRowNames ()`

Set objective function name to the default "obj" and row names to the default "cons0", "cons1", ...

8.39.2.20 `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.

8.39.2.21 `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\(\)](#)).

8.39.2.22 `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\(\)](#)).

8.39.2.23 `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.

8.39.2.24 `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.

8.39.2.25 `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.

8.39.2.26 `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.

8.39.2.27 `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 immediately flipped to get a minimization problem.

8.39.2.28 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 immediately flipped to get a minimization problem.

8.39.2.29 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.

8.39.2.30 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.

8.39.2.31 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.

8.39.2.32 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.

8.39.2.33 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.

8.39.2.34 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.

8.39.2.35 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.

8.39.2.36 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.

8.39.2.37 int CoinLpIO::first_is_number (const char * *buff*) const
[protected]

Return 1 if the first character of *buff* is a number.

Return 0 otherwise.

8.39.2.38 `int CoinLpIO::is_comment (const char * buff) const`
`[protected]`

Return 1 if the first character of *buff* is `'/'` or `'\'`.

Return 0 otherwise.

8.39.2.39 `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.

8.39.2.40 `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.

8.39.2.41 `int CoinLpIO::is_sense (const char * buff) const` `[protected]`

Return an integer indicating the inequality sense read.

Return 0 if *buff* is `'<='`. Return 1 if *buff* is `'='`. Return 2 if *buff* is `'>='`. Return -1 otherwise.

8.39.2.42 `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.

8.39.2.43 `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.

8.39.2.44 `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.

8.39.2.45 `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.

8.39.2.46 `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.

8.39.3 Member Data Documentation

8.39.3.1 `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`.

8.39.3.2 `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`.

8.39.3.3 `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`.

8.39.3.4 `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`.

8.39.3.5 `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.

8.39.3.6 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.

8.39.3.7 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.

8.39.3.8 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:

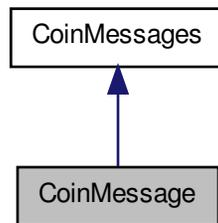
- CoinLpIO.hpp

8.40 CoinMessage Class Reference

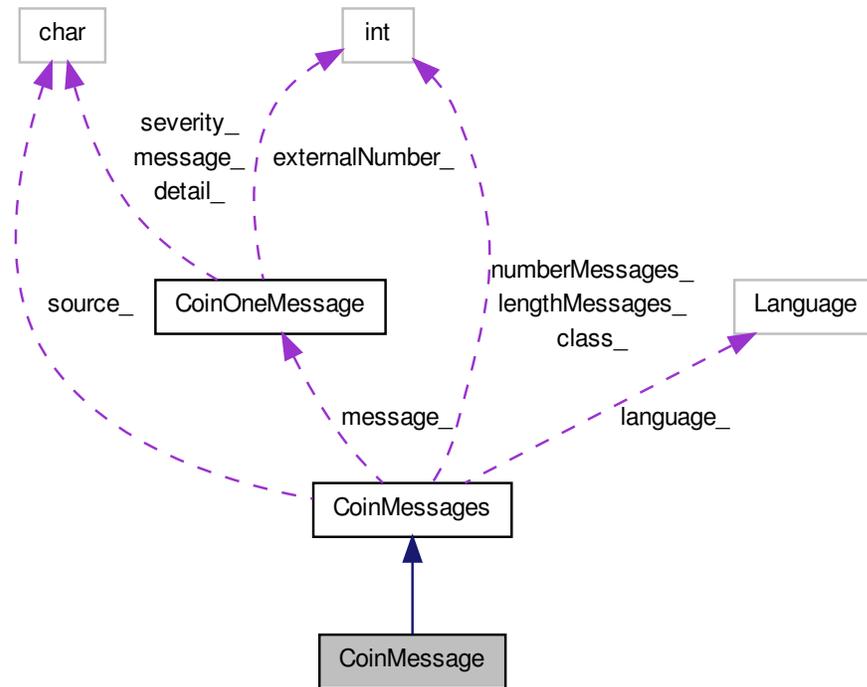
The standard set of Coin messages.

```
#include <CoinMessage.hpp>
```

Inheritance diagram for CoinMessage:



Collaboration diagram for CoinMessage:



Public Member Functions

Constructors etc

- [CoinMessage](#) ([Language](#) language=us_en)
Constructor.

8.40.1 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.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 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:

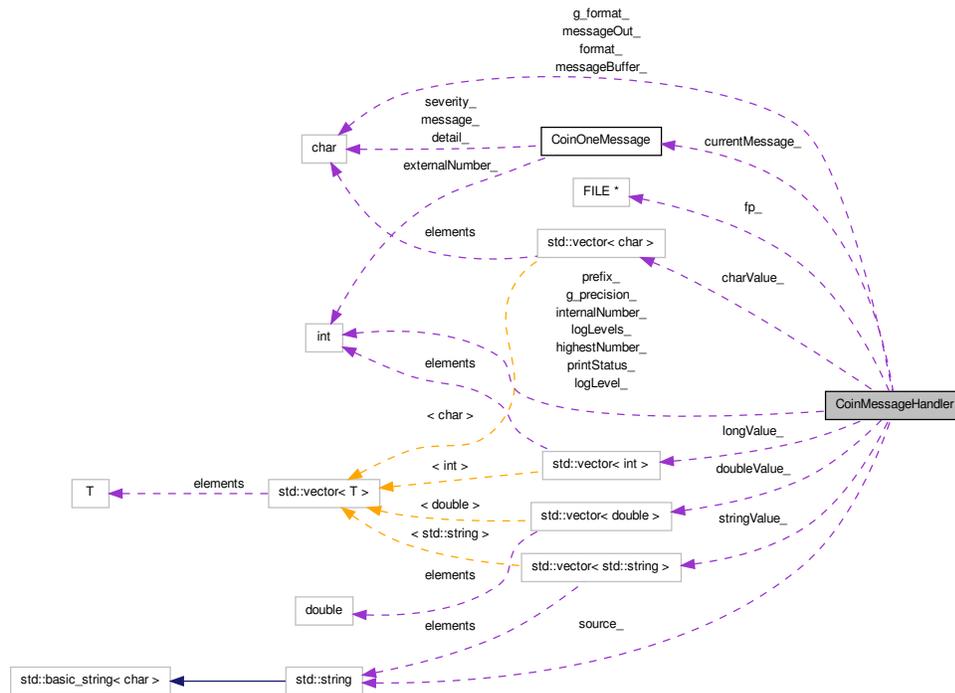
- [CoinMessage.hpp](#)

8.41 CoinMessageHandler Class Reference

Base class for message handling.

```
#include <CoinMessageHandler.hpp>
```

Collaboration diagram for CoinMessageHandler:



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.

8.41.1 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`.

8.41.2 Member Function Documentation

8.41.2.1 `CoinMessageHandler& CoinMessageHandler::operator= (const CoinMessageHandler &)`

Assignment operator.

8.41.2.2 `int CoinMessageHandler::logLevel () const [inline]`

Get current log (detail) level.

Definition at line 363 of file `CoinMessageHandler.hpp`.

8.41.2.3 `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.)

8.41.2.4 int CoinMessageHandler::logLevel (int *which*) const [inline]

Get alternative log level.

Definition at line 384 of file CoinMessageHandler.hpp.

8.41.2.5 void CoinMessageHandler::setLogLevel (int *which*, int *value*)

Set alternative log level value.

Can be used to store alternative log level information within the handler.

8.41.2.6 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.

8.41.2.7 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.

8.41.2.8 int CoinMessageHandler::intValue (int *position*) const [inline]

Values of integer fields already processed.

As the parameter for an integer field is processed, the value is saved and can be retrieved using this function.

Definition at line 419 of file CoinMessageHandler.hpp.

8.41.2.9 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.

8.41.2.10 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.

8.41.2.11 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.

8.41.2.12 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.

8.41.2.13 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.

8.41.2.14 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.

8.41.2.15 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.

8.41.2.16 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).

**8.41.2.17 CoinMessageHandler& CoinMessageHandler::operator<< (int
intvalue)**

Process an integer parameter value.

The default format code is 'd'.

**8.41.2.18 CoinMessageHandler& CoinMessageHandler::operator<< (double
doublevalue)**

Process a double parameter value.

The default format code is 'd'.

**8.41.2.19 CoinMessageHandler& CoinMessageHandler::operator<< (const
std::string & stringvalue)**

Process a STL string parameter value.

The default format code is 'g'.

**8.41.2.20 CoinMessageHandler& CoinMessageHandler::operator<< (char
charvalue)**

Process a char parameter value.

The default format code is 's'.

**8.41.2.21 CoinMessageHandler& CoinMessageHandler::operator<< (const
char * stringvalue)**

Process a C-style string parameter value.

The default format code is 'c'.

8.41.2.22 CoinMessageHandler& CoinMessageHandler::operator<< (CoinMessageMarker)

Process a marker.

The default format code is 's'.

8.41.2.23 int CoinMessageHandler::finish ()

Finish (and print) the message.

Equivalent to using the CoinMessageEol marker.

8.41.2.24 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.

8.41.3 Friends And Related Function Documentation

8.41.3.1 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.

8.41.4 Member Data Documentation

8.41.4.1 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.

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

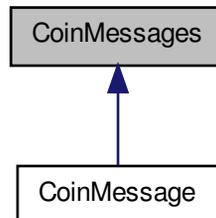
- [CoinMessageHandler.hpp](#)

8.42 CoinMessages Class Reference

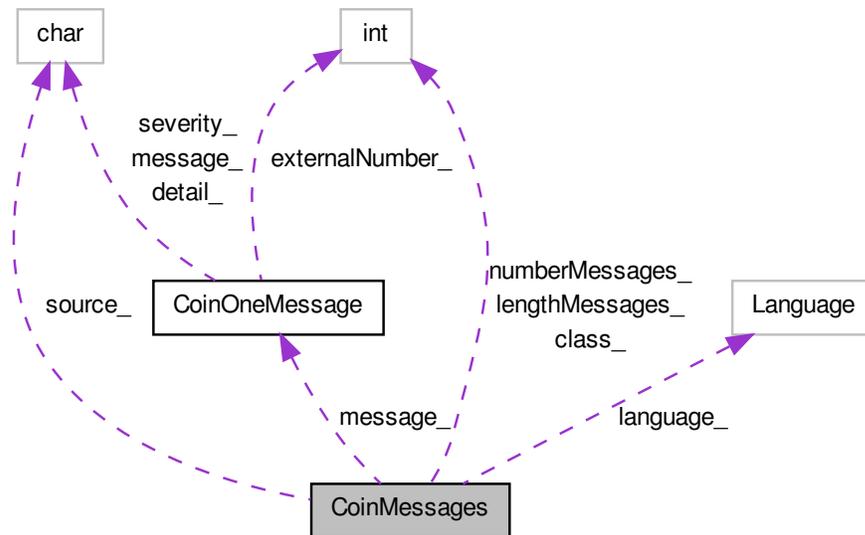
Class to hold and manipulate an array of massaged messages.

```
#include <CoinMessageHandler.hpp>
```

Inheritance diagram for CoinMessages:



Collaboration diagram for CoinMessages:



Public Types

- enum [Language](#)
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.

8.42.1 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`.

8.42.2 Member Enumeration Documentation

8.42.2.1 `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.

Definition at line 136 of file `CoinMessageHandler.hpp`.

8.42.3 Constructor & Destructor Documentation

8.42.3.1 `CoinMessages::CoinMessages (int numberMessages = 0)`

Constructor with number of messages.

8.42.4 Member Function Documentation

8.42.4.1 CoinMessages& CoinMessages::operator= (const CoinMessages &)

assignment operator.

8.42.4.2 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.

8.42.4.3 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.

8.42.4.4 Language CoinMessages::language () const [inline]

Language.

Need to think about iso codes

Definition at line 169 of file CoinMessageHandler.hpp.

8.42.4.5 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* >= 10000 or *messageNumbers* is NULL, the detail level is changed on all messages.

8.42.5 Member Data Documentation

8.42.5.1 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.

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

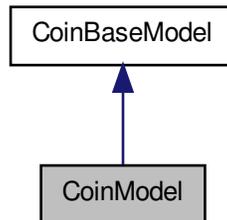
- [CoinMessageHandler.hpp](#)

8.43 CoinModel Class Reference

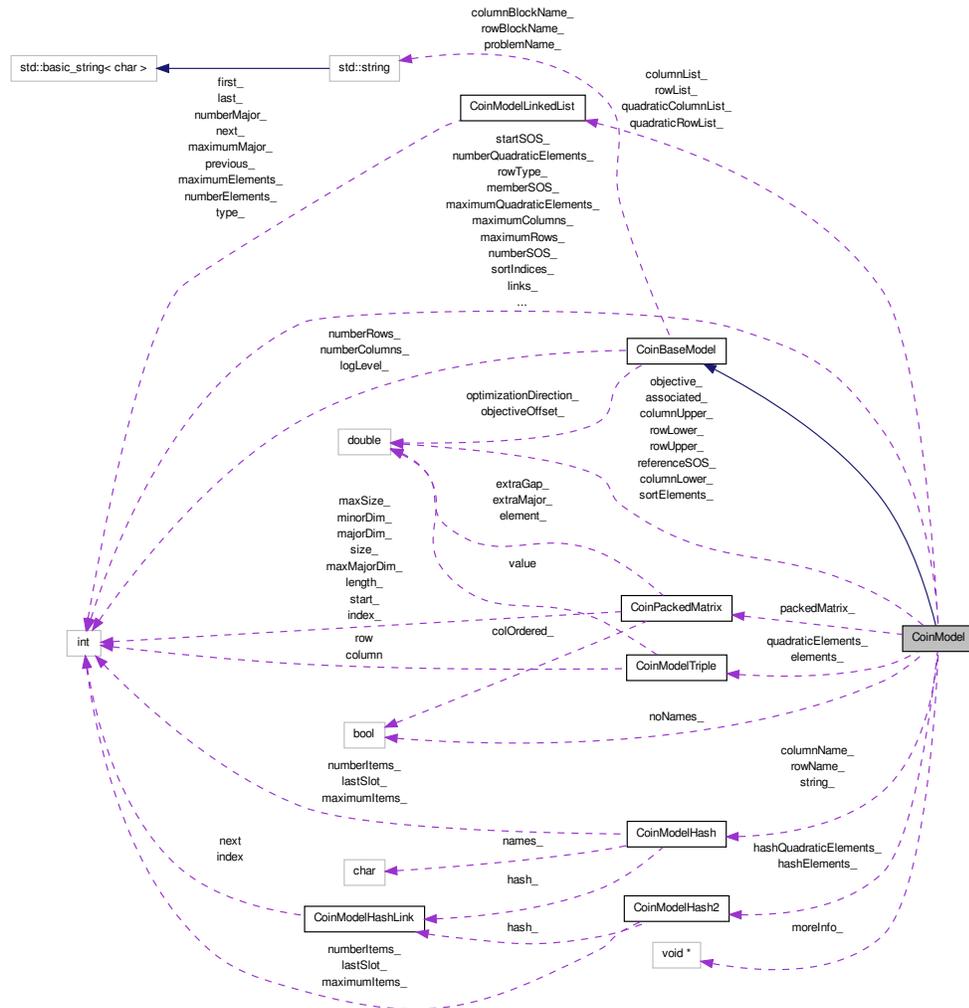
This is a simple minded model which is stored in a format which makes it easier to construct and modify but not efficient for algorithms.

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

Inheritance diagram for CoinModel:



Collaboration 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 **setColumnIsInteger** (int whichColumn, bool columnIsInteger)
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 columnIsInteger)
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 **setColIsInteger** (int whichColumn, bool columnIsInteger)
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 **setColumnIsInteger** (int whichColumn, const char *columnIsInteger)
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 *columnIsInteger)
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 `getColIsInteger` (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 *rowsen, 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 *rowsen, 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.

8.43.1 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`.

8.43.2 Constructor & Destructor Documentation

8.43.2.1 CoinModel::CoinModel ()

Default constructor.

8.43.2.2 CoinModel::CoinModel (int *firstRows*, int *firstColumns*, int *firstElements*, bool *noNames* = *false*)

Constructor with sizes.

8.43.2.3 CoinModel::CoinModel (const CoinModel &)

The copy constructor.

8.43.3 Member Function Documentation

8.43.3.1 `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

8.43.3.2 `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

8.43.3.3 `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.

8.43.3.4 `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.

8.43.3.5 `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.

8.43.3.6 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.

8.43.3.7 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.

8.43.3.8 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.

8.43.3.9 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.

8.43.3.10 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

compression can be set to three values to indicate what kind of file should be written

- 0: plain text (default)
- 1: gzip compressed (.gz is appended to `filename`)
- 2: bzip2 compressed (.bz2 is appended to `filename`) (TODO)

If the library was not compiled with the requested compression then `writeMps` falls back to writing a plain text file.

formatType specifies the precision to used for values in the MPS file

- 0: normal precision (default)
- 1: extra accuracy
- 2: IEEE hex

numberAcross 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

8.43.3.11 `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

8.43.3.12 `int CoinModel::convertMatrix ()`

Convert elements to [CoinPackedMatrix](#) (and switch off element updates).

Returns number of errors

8.43.3.13 `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

8.43.3.14 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

8.43.3.15 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

8.43.3.16 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`

8.43.3.17 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`

**8.43.3.18 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`

**8.43.3.19 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`

**8.43.3.20 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

**8.43.3.21 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.

**8.43.3.22 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.

**8.43.3.23 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`

**8.43.3.24 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

8.43.3.25 `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 $-infty \leq ax \leq infty$. This is probably not what you want.

8.43.3.26 `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 $ax \geq 0$.

8.43.3.27 `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 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.

8.43.3.28 `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 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.

8.43.3.29 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

8.43.3.30 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.

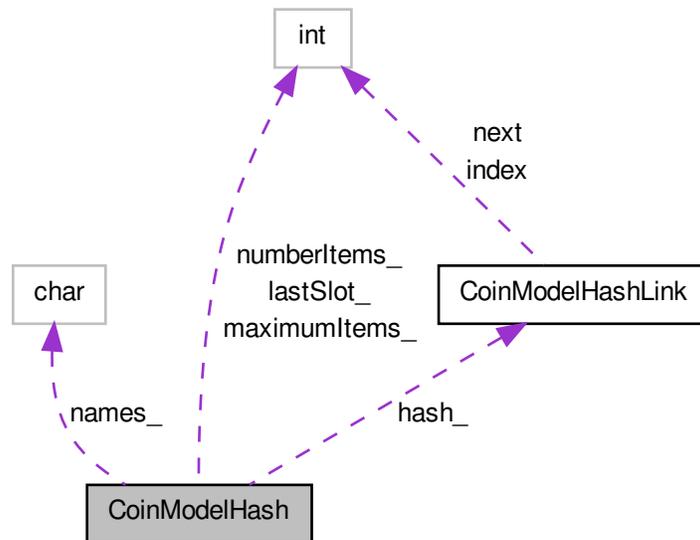
numberOutput will have at least space to return values which reconstruct input. Rows returned will be original rows but no entries will be returned for any rows all of whose entries are in knapsack. So up to user to allow for this. If reConstruct ≥ 0 then returns number of entries which make up item "reConstruct" in expanded knapsack. Values in buildRow and buildElement;

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

- CoinModel.hpp

8.44 CoinModelHash Class Reference

Collaboration diagram for CoinModelHash:



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.

8.44.1 Detailed Description

Definition at line 180 of file CoinModelUseful.hpp.

8.44.2 Constructor & Destructor Documentation

8.44.2.1 CoinModelHash::CoinModelHash ()

Default constructor.

8.44.2.2 CoinModelHash::CoinModelHash (const CoinModelHash &)

The copy constructor.

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

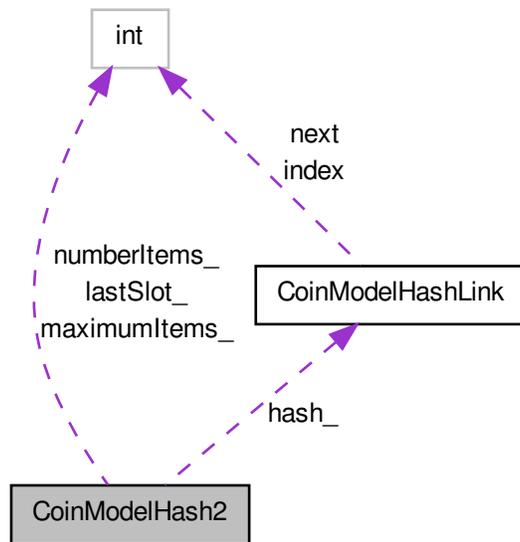
- CoinModelUseful.hpp

8.45 CoinModelHash2 Class Reference

For int,int hashing.

```
#include <CoinModelUseful.hpp>
```

Collaboration diagram for CoinModelHash2:



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.

8.45.1 Detailed Description

For int,int hashing.

Definition at line 253 of file CoinModelUseful.hpp.

8.45.2 Constructor & Destructor Documentation**8.45.2.1 CoinModelHash2::CoinModelHash2 ()**

Default constructor.

8.45.2.2 CoinModelHash2::CoinModelHash2 (const CoinModelHash2 &)

The copy constructor.

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

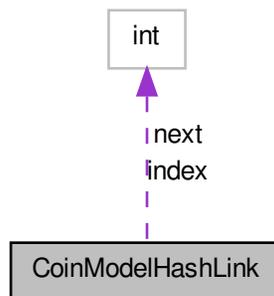
- CoinModelUseful.hpp

8.46 CoinModelHashLink Struct Reference

for names and hashing

```
#include <CoinModelUseful.hpp>
```

Collaboration diagram for CoinModelHashLink:



8.46.1 Detailed Description

for names and hashing

Definition at line 128 of file CoinModelUseful.hpp.

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

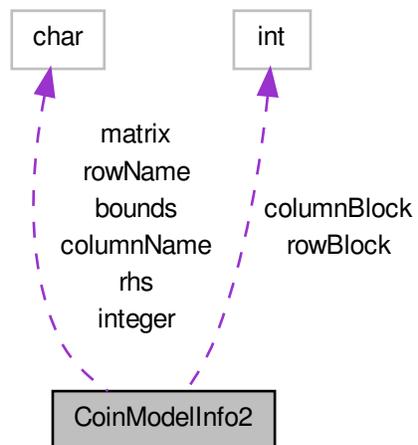
- CoinModelUseful.hpp

8.47 CoinModelInfo2 Struct Reference

This is a model which is made up of Coin(Structured)Model blocks.

```
#include <CoinStructuredModel.hpp>
```

Collaboration diagram for CoinModelInfo2:



8.47.1 Detailed Description

This is a model which is made up of Coin(Structured)Model blocks.

Definition at line 15 of file `CoinStructuredModel.hpp`.

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

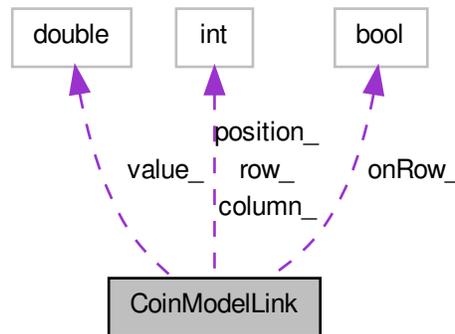
- `CoinStructuredModel.hpp`

8.48 CoinModelLink Class Reference

This is for various structures/classes needed by [CoinModel](#).

```
#include <CoinModelUseful.hpp>
```

Collaboration diagram for CoinModelLink:



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.

8.48.1 Detailed Description

This is for various structures/classes needed by [CoinModel](#). [CoinModelLink](#) [CoinModelLinkedList](#) [CoinModelHash](#) for going through row or column

Definition at line 30 of file [CoinModelUseful.hpp](#).

8.48.2 Constructor & Destructor Documentation

8.48.2.1 CoinModelLink::CoinModelLink ()

Default constructor.

8.48.2.2 CoinModelLink::CoinModelLink (const CoinModelLink &)

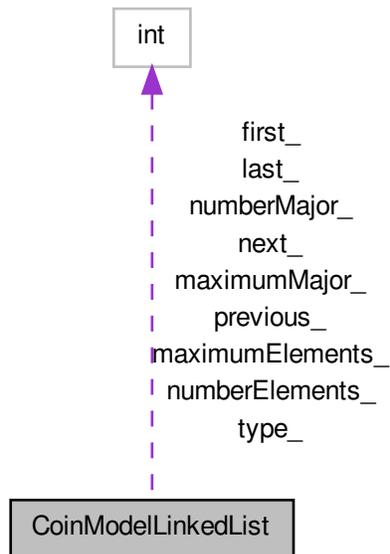
The copy constructor.

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

- CoinModelUseful.hpp

8.49 CoinModelLinkedList Class Reference

Collaboration diagram for CoinModelLinkedList:



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.

8.49.1 Detailed Description

Definition at line 312 of file CoinModelUseful.hpp.

8.49.2 Constructor & Destructor Documentation

8.49.2.1 CoinModelLinkedList::CoinModelLinkedList ()

Default constructor.

8.49.2.2 CoinModelLinkedList::CoinModelLinkedList (const CoinModelLinkedList &)

The copy constructor.

8.49.3 Member Function Documentation

8.49.3.1 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

8.49.3.2 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

8.49.3.3 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

8.49.3.4 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

8.49.3.5 void CoinModelLinkedList::deleteSame (int *which*, CoinModelTriple * *triples*, CoinModelHash2 & *hash*, bool *zapTriples*)

Deletes from list - same case i.e.

delete row from row list

8.49.3.6 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

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

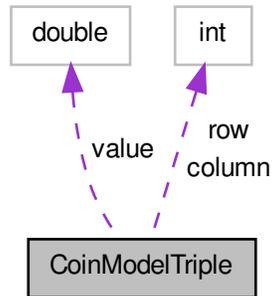
- CoinModelUseful.hpp

8.50 CoinModelTriple Struct Reference

for linked lists

```
#include <CoinModelUseful.hpp>
```

Collaboration diagram for CoinModelTriple:



8.50.1 Detailed Description

for linked lists

Definition at line 107 of file `CoinModelUseful.hpp`.

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

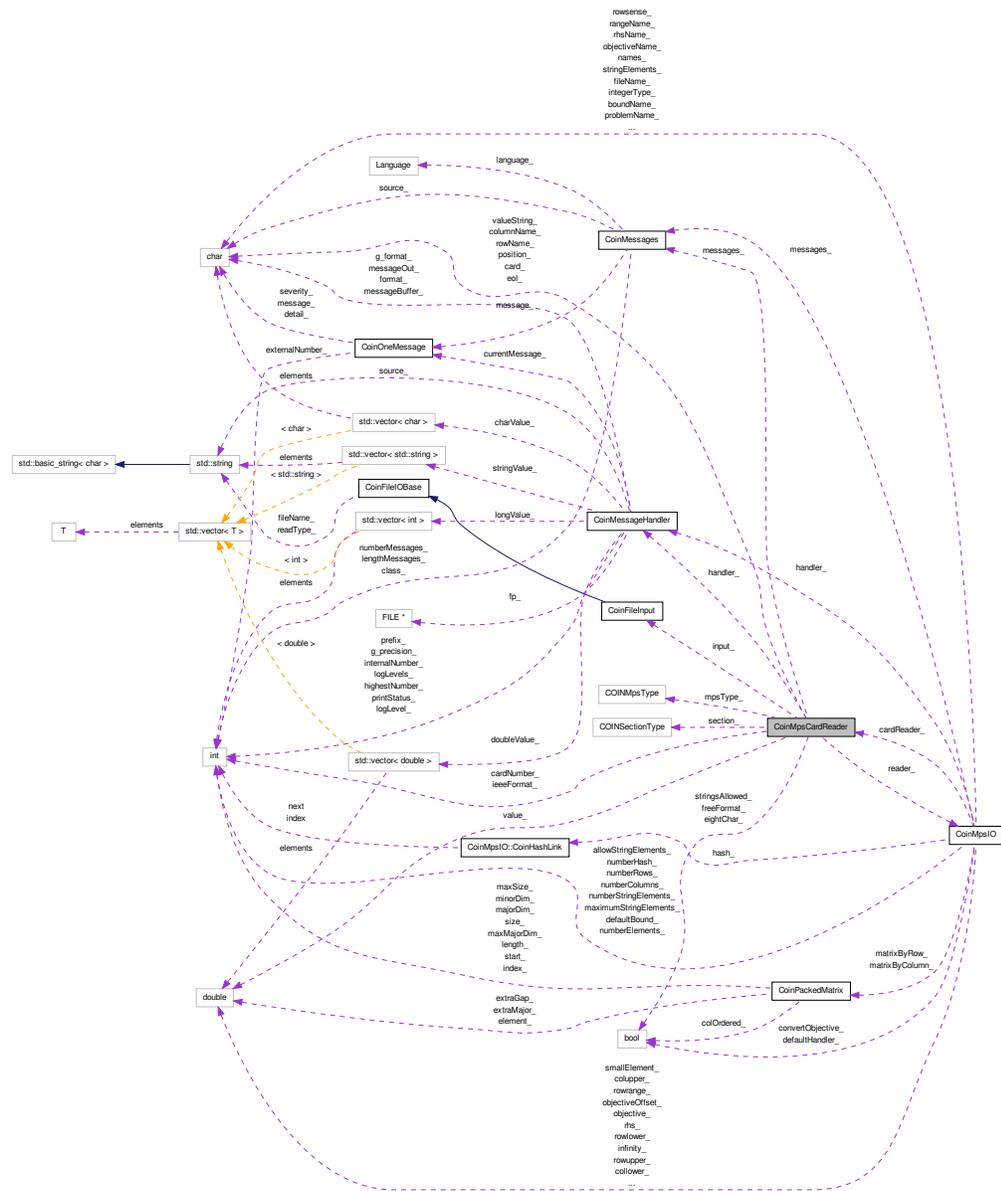
- `CoinModelUseful.hpp`

8.51 CoinMpsCardReader Class Reference

Very simple code for reading MPS data.

```
#include <CoinMpsIO.hpp>
```

Collaboration diagram for CoinMpsCardReader:



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)

8.51.1 Detailed Description

Very simple code for reading MPS data.

Definition at line 59 of file CoinMpsIO.hpp.

8.51.2 Member Function Documentation

8.51.2.1 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 - ;

8.51.2.2 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.

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

- CoinMpsIO.hpp

8.52 CoinMpsIO Class Reference

MPS IO Interface.

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.)

- 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 unneeded 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, [Coin-FileInput](#) *&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.

8.52.1 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](#).

8.52.2 Member Function Documentation

8.52.2.1 [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

8.52.2.2 `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`

8.52.2.3 `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.

8.52.2.4 `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.

8.52.2.5 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.

8.52.2.6 const char* CoinMpsIO::rowName (int *index*) const

Returns the row name for the specified index.

Returns 0 if the index is out of range.

8.52.2.7 const char* CoinMpsIO::columnName (int *index*) const

Returns the column name for the specified index.

Returns 0 if the index is out of range.

8.52.2.8 int CoinMpsIO::rowIndex (const char * *name*) const

Returns the index for the specified row name.

Returns -1 if the name is not found. Returns `numberRows` for the objective row and `> numberRows` for dropped free rows.

8.52.2.9 int CoinMpsIO::columnIndex (const char * *name*) const

Returns the index for the specified column name.

Returns -1 if the name is not found.

8.52.2.10 double CoinMpsIO::objectiveOffset () const

Returns the (constant) objective offset.

This is the RHS entry for the objective row

8.52.2.11 `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.

8.52.2.12 `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.

8.52.2.13 `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

8.52.2.14 `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.

8.52.2.15 `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

8.52.2.16 `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 messages objective column

8.52.2.17 `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

8.52.2.18 `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

8.52.2.19 `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

compression can be set to three values to indicate what kind of file should be

written

- 0: plain text (default)
- 1: gzip compressed (.gz is appended to `filename`)
- 2: bzip2 compressed (.bz2 is appended to `filename`) (TODO)

If the library was not compiled with the requested compression then `writeMps` falls back to writing a plain text file.

formatType specifies the precision to used for values in the MPS file

- 0: normal precision (default)
- 1: extra accuracy
- 2: IEEE hex

numberAcross specifies whether 1 or 2 (default) values should be specified on every data line in the MPS file.

quadratic specifies quadratic objective to be output

8.52.2.20 `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 $c + 1/2 x^T Q x$ 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 $2*Q$) 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`

8.52.2.21 `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)

8.52.2.22 `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.

8.52.2.23 `void CoinMpsIO::releaseRedundantInformation ()`

Release all information which can be re-calculated.

E.g., row sense, copies of rows, hash tables for names.

8.52.2.24 `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.

8.52.3 Friends And Related Function Documentation

8.52.3.1 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!)..

8.52.4 Member Data Documentation

8.52.4.1 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](#).

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

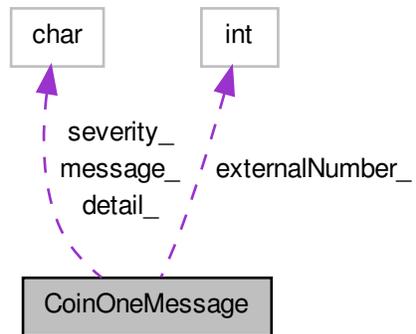
- [CoinMpsIO.hpp](#)

8.53 CoinOneMessage Class Reference

Class for one massaged message.

```
#include <CoinMessageHandler.hpp>
```

Collaboration diagram for CoinOneMessage:



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).

8.53.1 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.

8.53.2 Constructor & Destructor Documentation

8.53.2.1 CoinOneMessage::CoinOneMessage ()

Default constructor.

8.53.3 Member Function Documentation

8.53.3.1 CoinOneMessage& CoinOneMessage::operator= (const CoinOneMessage &)

assignment operator.

8.53.3.2 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.

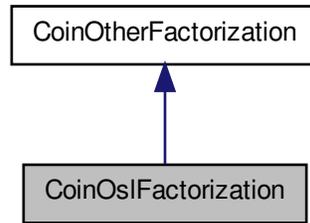
Definition at line 92 of file CoinMessageHandler.hpp.

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

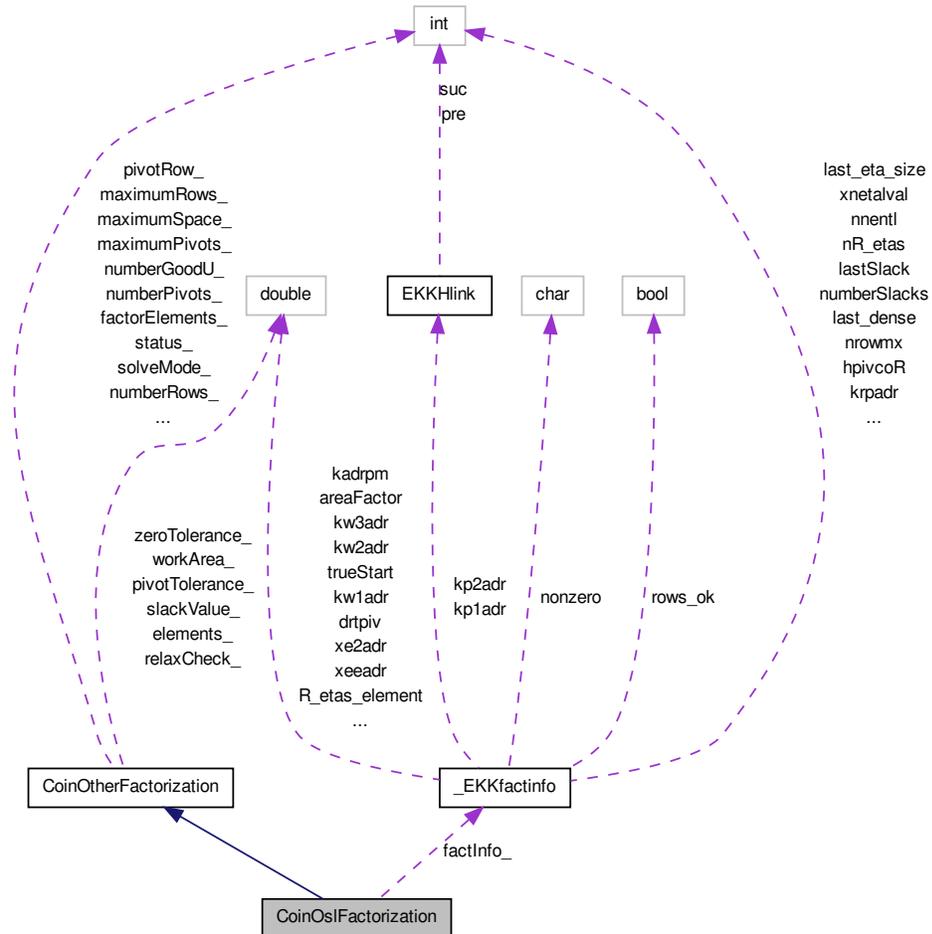
- [CoinMessageHandler.hpp](#)

8.54 CoinOslFactorization Class Reference

Inheritance diagram for CoinOslFactorization:



Collaboration 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 rowIsBasic[], int columnIsBasic[], 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 `setUsefulInformation (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.

8.54.1 Detailed Description

Definition at line 106 of file CoinOslFactorization.hpp.

8.54.2 Member Function Documentation

8.54.2.1 int CoinOslFactorization::factorize (const CoinPackedMatrix & matrix, int rowIsBasic[], int columnIsBasic[], 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

8.54.2.2 `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](#).

8.54.2.3 `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 regionSparse starts as zero and is zero at end.

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

Implements [CoinOtherFactorization](#).

8.54.2.4 `virtual int* CoinOslFactorization::indices () const [virtual]`

Get rid of all memory.

Returns array to put basis indices in

Implements [CoinOtherFactorization](#).

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

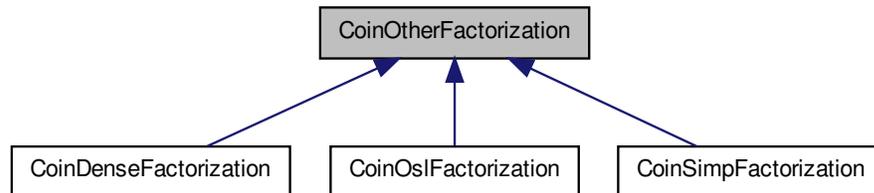
- CoinOslFactorization.hpp

8.55 CoinOtherFactorization Class Reference

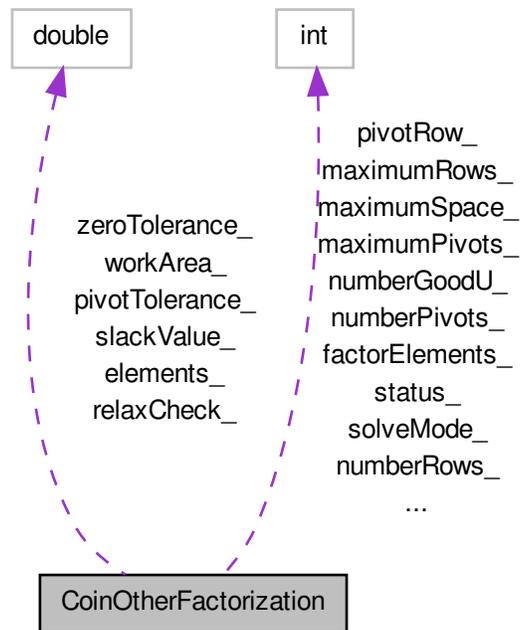
Abstract base class which also has some scalars so can be used from Dense or Simp.

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

Inheritance diagram for CoinOtherFactorization:



Collaboration 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 `setUsefulInformation` (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 numberOfRows, 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.

8.55.1 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.

8.55.2 Member Function Documentation

8.55.2.1 `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.

8.55.2.2 `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.

8.55.2.3 `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](#), [CoinOsIFactorization](#), and [CoinSimpFactorization](#).

8.55.2.4 `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](#).

8.55.3 Member Data Documentation

8.55.3.1 `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:

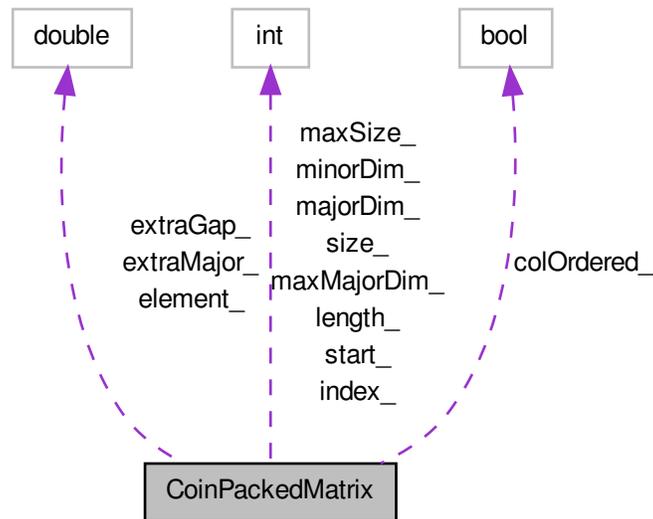
- [CoinDenseFactorization.hpp](#)

8.56 CoinPackedMatrix Class Reference

Sparse Matrix Base Class.

```
#include <CoinPackedMatrix.hpp>
```

Collaboration diagram for CoinPackedMatrix:



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 numberOfRows=-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 numberColumns=-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 $fabs() \leq removeValue$.
- 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 numberRows, 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 [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.
- 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.

8.56.1 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`.

8.56.2 Constructor & Destructor Documentation

8.56.2.1 `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.

8.56.2.2 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.

8.56.2.3 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

8.56.3 Member Function Documentation

8.56.3.1 double CoinPackedMatrix::getExtraGap () const [inline]

Return the current setting of the extra gap.

Definition at line 89 of file CoinPackedMatrix.hpp.

8.56.3.2 double CoinPackedMatrix::getExtraMajor () const [inline]

Return the current setting of the extra major.

Definition at line 91 of file CoinPackedMatrix.hpp.

8.56.3.3 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)

8.56.3.4 bool CoinPackedMatrix::isColOrdered () const [inline]

Whether the packed matrix is column major ordered or not.

Definition at line 101 of file CoinPackedMatrix.hpp.

8.56.3.5 bool CoinPackedMatrix::hasGaps () const [inline]

Whether the packed matrix has gaps or not.

Definition at line 104 of file CoinPackedMatrix.hpp.

8.56.3.6 CoinBigIndex CoinPackedMatrix::getNumElements () const [inline]

Number of entries in the packed matrix.

Definition at line 107 of file CoinPackedMatrix.hpp.

8.56.3.7 int CoinPackedMatrix::getNumCols () const [inline]

Number of columns.

Definition at line 110 of file CoinPackedMatrix.hpp.

8.56.3.8 int CoinPackedMatrix::getNumRows () const [inline]

Number of rows.

Definition at line 114 of file CoinPackedMatrix.hpp.

8.56.3.9 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.

8.56.3.10 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.

8.56.3.11 int CoinPackedMatrix::getSizeVectorStarts () const [inline]

The size of the `vectorStarts` array.

See `start_`.

Definition at line 140 of file CoinPackedMatrix.hpp.

8.56.3.12 int CoinPackedMatrix::getSizeVectorLengths () const [inline]

The size of the `vectorLengths` array.

See [length_](#).

Definition at line 147 of file `CoinPackedMatrix.hpp`.

8.56.3.13 `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`.

8.56.3.14 `const int* CoinPackedMatrix::getVectorLengths () const` `[inline]`

The lengths of the major-dimension vectors.

See [length_](#).

Definition at line 160 of file `CoinPackedMatrix.hpp`.

8.56.3.15 `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`.

8.56.3.16 `int CoinPackedMatrix::getVectorSize (const int i) const` `[inline]`

The length of *i*'th vector.

Definition at line 181 of file `CoinPackedMatrix.hpp`.

8.56.3.17 `const CoinShallowPackedVector CoinPackedMatrix::getVector (int i) const [inline]`

Return the i'th vector in matrix.

Definition at line 190 of file CoinPackedMatrix.hpp.

8.56.3.18 `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 []`.

8.56.3.19 `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.

8.56.3.20 `void CoinPackedMatrix::setExtraGap (const double newGap)`

Set the extra gap to be allocated to the specified value.

8.56.3.21 `void CoinPackedMatrix::setExtraMajor (const double newMajor)`

Set the extra major to be allocated to the specified value.

8.56.3.22 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.

8.56.3.23 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.

8.56.3.24 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.

8.56.3.25 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`.

8.56.3.26 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.

8.56.3.27 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.

8.56.3.28 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.

8.56.3.29 int CoinPackedMatrix::appendRows (const int *numrows*, const CoinBigIndex * *rowStarts*, const int * *column*, const double * *element*, int *numberColumns* = -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 *numberColumns* < 0.

8.56.3.30 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.

8.56.3.31 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.

8.56.3.32 void CoinPackedMatrix::deleteCols (const int *numDel*, const int * *indDel*)

Delete the columns whose indices are listed in *indDel*.

8.56.3.33 void CoinPackedMatrix::deleteRows (const int *numDel*, const int * *indDel*)

Delete the rows whose indices are listed in *indDel*.

8.56.3.34 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

8.56.3.35 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

8.56.3.36 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

8.56.3.37 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

8.56.3.38 int CoinPackedMatrix::eliminateDuplicates (double threshold)

Eliminate all duplicate AND small elements in matrix The column starts are not affected.

Returns number of elements eliminated.

8.56.3.39 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

8.56.3.40 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.

8.56.3.41 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.

8.56.3.42 void CoinPackedMatrix::copyOf (const CoinPackedMatrix & *rhs*)

Copy method.

This method makes an exact replica of the argument, including the extra space parameters.

8.56.3.43 void CoinPackedMatrix::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.

If *len* is a NULL pointer then the matrix is assumed to have no gaps in it and *len* will be created accordingly.

8.56.3.44 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

8.56.3.45 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.)

8.56.3.46 void CoinPackedMatrix::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.

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[]`!

8.56.3.47 CoinPackedMatrix& CoinPackedMatrix::operator= (const CoinPackedMatrix & rhs)

Assignment operator.

This copies out the data, but uses the current matrix's extra space parameters.

8.56.3.48 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. [reverseOrderedCopyOf](#), which does the same thing but produces a new matrix.)

8.56.3.49 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\(\)](#).

8.56.3.50 void CoinPackedMatrix::swap (CoinPackedMatrix & *matrix*)

Swap the content of two packed matrices.

8.56.3.51 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()`

8.56.3.52 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.

8.56.3.53 `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 ()`

8.56.3.54 `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.

8.56.3.55 `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 []`.

8.56.3.56 `void CoinPackedMatrix::countOrthoLength (int * counts) const`

Count the number of entries in every minor-dimension vector and fill in an array containing these lengths.

8.56.3.57 `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`.

8.56.3.58 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.

8.56.3.59 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.

8.56.3.60 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.

8.56.3.61 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.

8.56.3.62 void CoinPackedMatrix::dumpMatrix (const char * *fname* = *NULL*) const

Dump the matrix on stdout.

When in dire straits this method can help.

8.56.3.63 void CoinPackedMatrix::appendMajorVector (const CoinPackedVectorBase & *vec*)

Append a major-dimension vector to the end of the matrix.

8.56.3.64 void CoinPackedMatrix::appendMajorVector (const int *vecsize*, const int * *vecind*, const double * *vecelem*)

Append a major-dimension vector to the end of the matrix.

8.56.3.65 void CoinPackedMatrix::appendMinorVector (const CoinPackedVectorBase & *vec*)

Append a minor-dimension vector to the end of the matrix.

8.56.3.66 void CoinPackedMatrix::appendMinorVector (const int *vecsize*, const int * *vecind*, const double * *vecelem*)

Append a minor-dimension vector to the end of the matrix.

8.56.3.67 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.

8.56.3.68 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.

8.56.3.69 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.

8.56.3.70 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.

8.56.3.71 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.

8.56.3.72 void CoinPackedMatrix::deleteMajorVectors (const int *numDel*,
const int * *indDel*)

Delete the major-dimension vectors whose indices are listed in *indDel*.

8.56.3.73 void CoinPackedMatrix::deleteMinorVectors (const int *numDel*,
const int * *indDel*)

Delete the minor-dimension vectors whose indices are listed in *indDel*.

8.56.3.74 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()`

8.56.3.75 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.

8.56.3.76 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()`

8.56.3.77 `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.

8.56.3.78 `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`.

8.56.3.79 `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.

8.56.3.80 `bool CoinPackedMatrix::isEquivalent (const CoinPackedMatrix & rhs) const`

Test for equivalence.

The test for element equality is the default [CoinRelFltEq](#) operator.

8.56.3.81 `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.

8.56.3.82 `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.

8.56.3.83 `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.

8.56.3.84 `int* CoinPackedMatrix::getMutableVectorLengths () const [inline]`

The lengths of the major-dimension vectors.

Definition at line 721 of file CoinPackedMatrix.hpp.

8.56.3.85 `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.

8.56.3.86 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.

8.56.3.87 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.

8.56.3.88 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.

8.56.3.89 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.

8.56.3.90 `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

$$0 \leq k < \text{numberOther}$$

If `numberOther < 0` no checking is performed.

8.56.3.91 `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.

8.56.4 Friends And Related Function Documentation

8.56.4.1 `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.

8.56.5 Member Data Documentation

8.56.5.1 `bool CoinPackedMatrix::colOrdered_` `[protected]`

A flag indicating whether the matrix is column or row major ordered.

Definition at line 900 of file `CoinPackedMatrix.hpp`.

8.56.5.2 `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`.

8.56.5.3 `double CoinPackedMatrix::extraMajor_` `[protected]`

This 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`.

8.56.5.4 `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`.

8.56.5.5 `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.

8.56.5.6 CoinBigIndex* CoinPackedMatrix::start_ [protected]

Starting positions of major-dimension vectors.

Definition at line 918 of file CoinPackedMatrix.hpp.

8.56.5.7 int* CoinPackedMatrix::length_ [protected]

Lengths of major-dimension vectors.

Definition at line 920 of file CoinPackedMatrix.hpp.

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

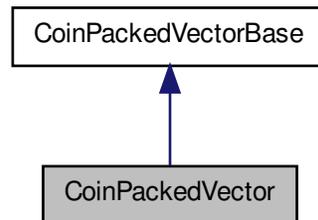
- CoinPackedMatrix.hpp

8.57 CoinPackedVector Class Reference

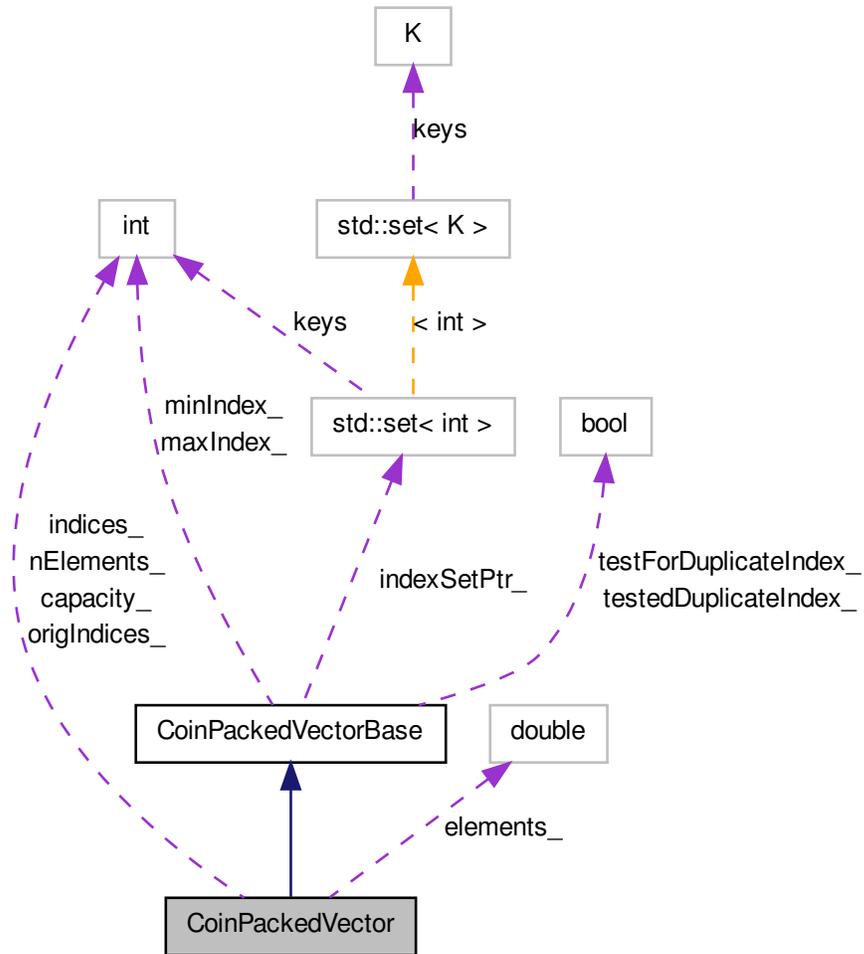
Sparse Vector.

```
#include <CoinPackedVector.hpp>
```

Inheritance diagram for CoinPackedVector:



Collaboration 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.

8.57.1 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.

8.57.2 Constructor & Destructor Documentation

8.57.2.1 `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.

8.57.2.2 `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.

8.57.2.3 `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.

8.57.2.4 `CoinPackedVector::CoinPackedVector (const CoinPackedVector &)`

Copy constructor.

8.57.2.5 `CoinPackedVector::CoinPackedVector (const CoinPackedVectorBase & rhs)`

Copy constructor *from a PackedVectorBase*.

8.57.3 Member Function Documentation

8.57.3.1 `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`.

8.57.3.2 `CoinPackedVector& CoinPackedVector::operator= (const CoinPackedVector &)`

Assignment operator.

NOTE: This operator keeps the current `testForDuplicateIndex` setting, and after copying the data it acts accordingly.

8.57.3.3 `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.

Reimplemented from [CoinPackedVectorBase](#).

8.57.3.4 `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.

8.57.3.5 `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.

8.57.3.6 `void CoinPackedVector::truncate (int newSize)`

Resize the packed vector to be the first *newSize* elements.

Problem with truncate: what happens with *origIndices_* ???

8.57.3.7 `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`.

8.57.3.8 `void CoinPackedVector::sortOriginalOrder ()`

Sort in original order.

If the vector has been sorted, then this method restores to its original sort order.

8.57.3.9 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.

8.57.4 Friends And Related Function Documentation

8.57.4.1 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:

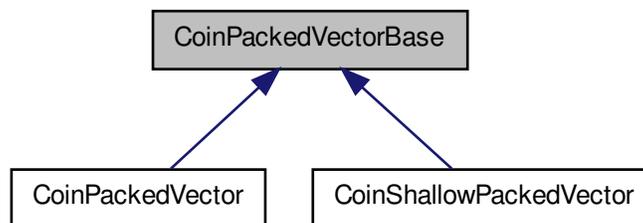
- CoinPackedVector.hpp

8.58 CoinPackedVectorBase Class Reference

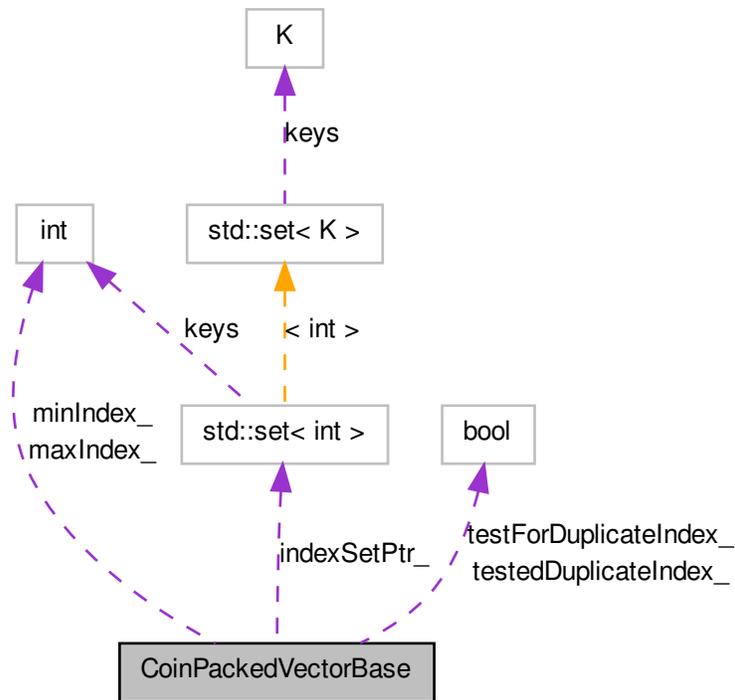
Abstract base class for various sparse vectors.

```
#include <CoinPackedVectorBase.hpp>
```

Inheritance diagram for CoinPackedVectorBase:



Collaboration 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.

8.58.1 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`.

8.58.2 Constructor & Destructor Documentation

8.58.2.1 `CoinPackedVectorBase::CoinPackedVectorBase ()` [protected]

Default constructor.

8.58.3 Member Function Documentation

8.58.3.1 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.

8.58.3.2 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.

8.58.3.3 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`.

8.58.3.4 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.

8.58.3.5 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()`.

8.58.3.6 `bool CoinPackedVectorBase::isExistingIndex (int i) const`

Return true if the i 'th element of the full storage vector exists in the packed storage vector.

8.58.3.7 `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

8.58.3.8 `bool CoinPackedVectorBase::operator==(const CoinPackedVectorBase & rhs) const`

Equal.

Returns true if vectors have same length and corresponding element of each vector is equal.

8.58.3.9 `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`.

8.58.3.10 `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.

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

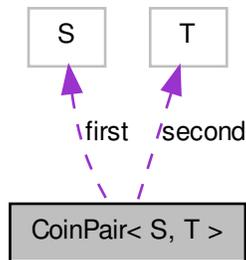
- CoinPackedVectorBase.hpp

8.59 CoinPair< S, T > Struct Template Reference

An ordered pair.

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

Collaboration diagram for CoinPair< S, T >:



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.

8.59.1 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`.

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

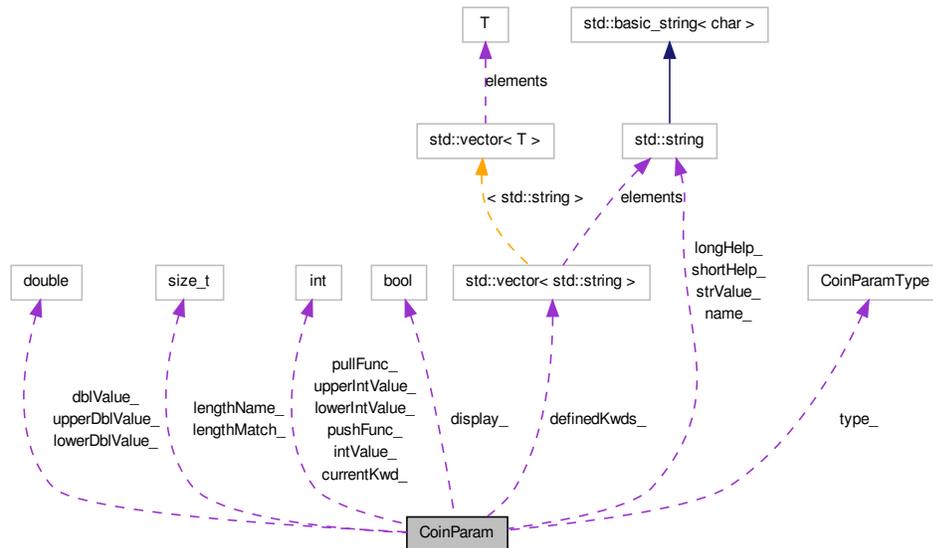
- `CoinSort.hpp`

8.60 CoinParam Class Reference

A base class for 'keyword value' command line parameters.

```
#include <CoinParam.hpp>
```

Collaboration diagram for CoinParam:



Public Types

Subtypes

- enum [CoinParamType](#)
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 `setDbfVal` (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 a integer parameter.
- int `intVal` () const
Get the value of a 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.

8.60.1 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!ableGap 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.

8.60.2 Member Typedef Documentation

8.60.2.1 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 non-fatal 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.

8.60.3 Member Enumeration Documentation

8.60.3.1 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.

Definition at line 95 of file CoinParam.hpp.

8.60.4 Constructor & Destructor Documentation

8.60.4.1 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.

8.60.4.2 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.

8.60.4.3 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.

8.60.4.4 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.

8.60.5 Member Function Documentation

8.60.5.1 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.

8.60.5.2 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.

8.60.5.3 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.

8.60.5.4 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.

8.60.5.5 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.

8.60.5.6 `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.

8.60.5.7 `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)`.

8.60.5.8 `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`.

8.60.6 Friends And Related Function Documentation

8.60.6.1 `void setInputSrc (FILE * src) [related]`

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

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

8.60.6.2 `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.

8.60.6.3 `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.

8.60.6.4 `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.

8.60.6.5 `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).

8.60.6.6 `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.

8.60.6.7 `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)

8.60.6.8 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.

8.60.6.9 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.

8.60.6.10 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.

8.60.6.11 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.

8.60.6.12 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.

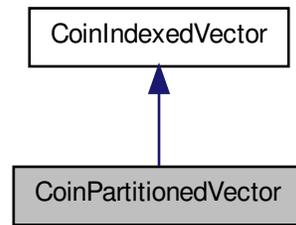
Intended as a utility to implement explicit 'help' commands. Help will be printed for all parameters in `paramVec` from `firstParam` to `lastParam`, inclusive. If `shortHelp` is true, short help messages will be printed. If `longHelp` is true, long help messages are printed. `shortHelp` overrules `longHelp`. 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:

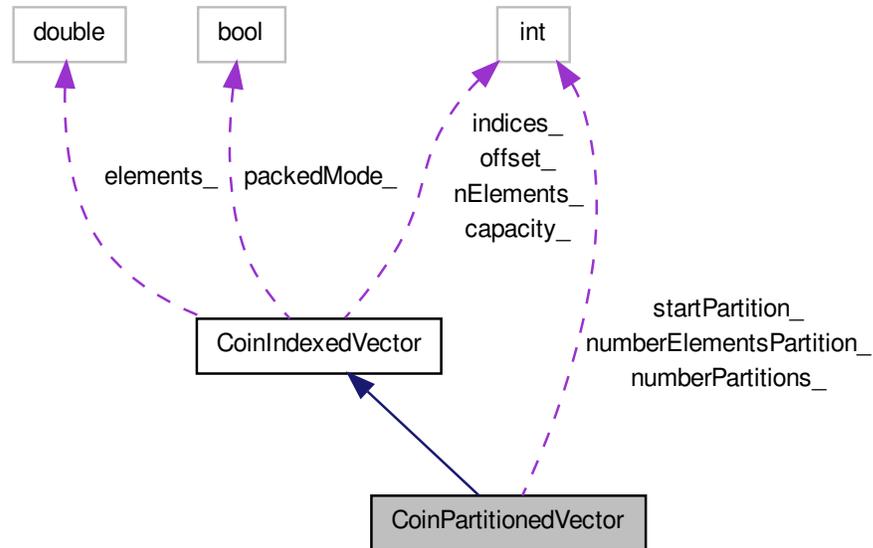
- [CoinParam.hpp](#)

8.61 CoinPartitionedVector Class Reference

Inheritance diagram for CoinPartitionedVector:



Collaboration 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).

8.61.1 Detailed Description

Definition at line 1055 of file CoinIndexedVector.hpp.

8.61.2 Constructor & Destructor Documentation

8.61.2.1 CoinPartitionedVector::CoinPartitionedVector (int *size*, const double * *elements*)

Alternate Constructors - construct full storage with indices 0 through size-1.

8.61.2.2 CoinPartitionedVector::CoinPartitionedVector (const CoinPartitionedVector &)

Copy constructor.

8.61.3 Member Function Documentation

8.61.3.1 void CoinPartitionedVector::print () const

Scan dense region from start to < end and set up indices returns number found.

Print out

Reimplemented from [CoinIndexedVector](#).

8.61.3.2 void CoinPartitionedVector::sort ()

Sort the indexed storage vector (increasing indices).

Reimplemented from [CoinIndexedVector](#).

8.61.3.3 CoinPartitionedVector& CoinPartitionedVector::operator= (const CoinPartitionedVector &)

Assignment operator.

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

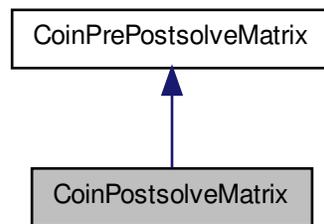
- CoinIndexedVector.hpp

8.62 CoinPostsolveMatrix Class Reference

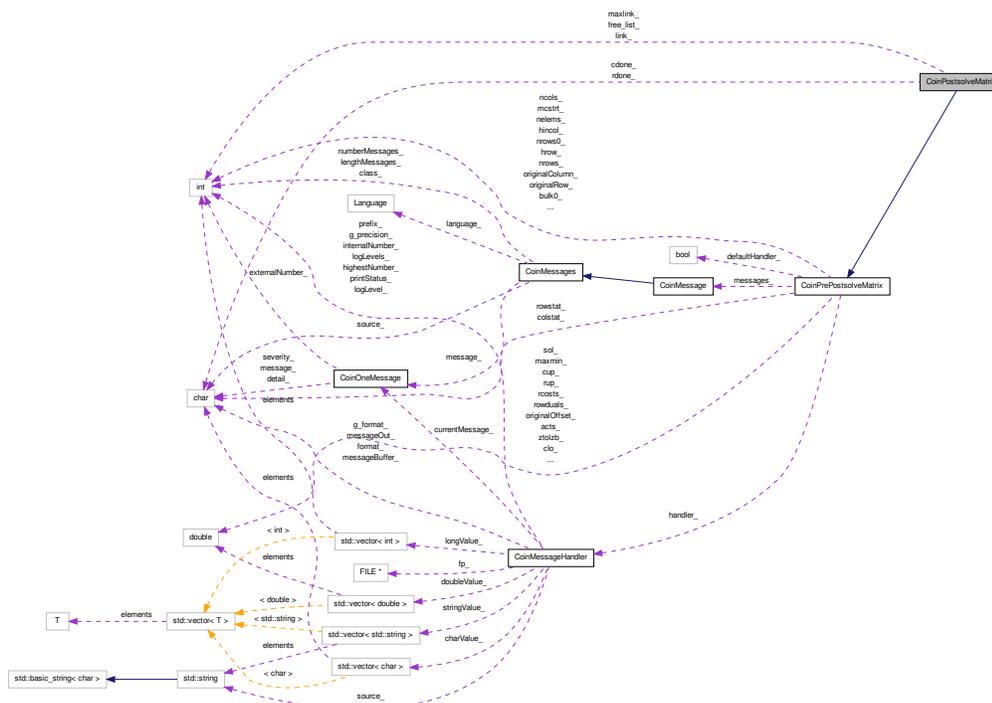
Augments [CoinPrePostsolveMatrix](#) with information about the problem that is only needed during postsolve.

```
#include <CoinPresolveMatrix.hpp>
```

Inheritance diagram for CoinPostsolveMatrix:



Collaboration 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.

8.62.1 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`.

8.62.2 Constructor & Destructor Documentation

8.62.2.1 `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`.

8.62.2.2 `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.

8.62.2.3 `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.

8.62.3 Member Function Documentation

8.62.3.1 `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.

8.62.4 Member Data Documentation

8.62.4.1 `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.

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

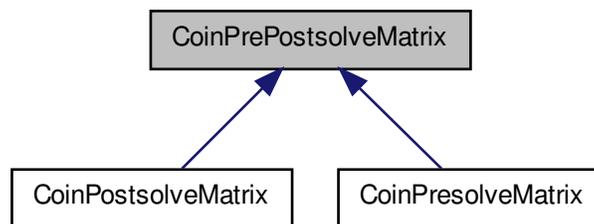
- [CoinPresolveMatrix.hpp](#)
- [CoinPresolvePsdebug.hpp](#)

8.63 CoinPrePostsolveMatrix Class Reference

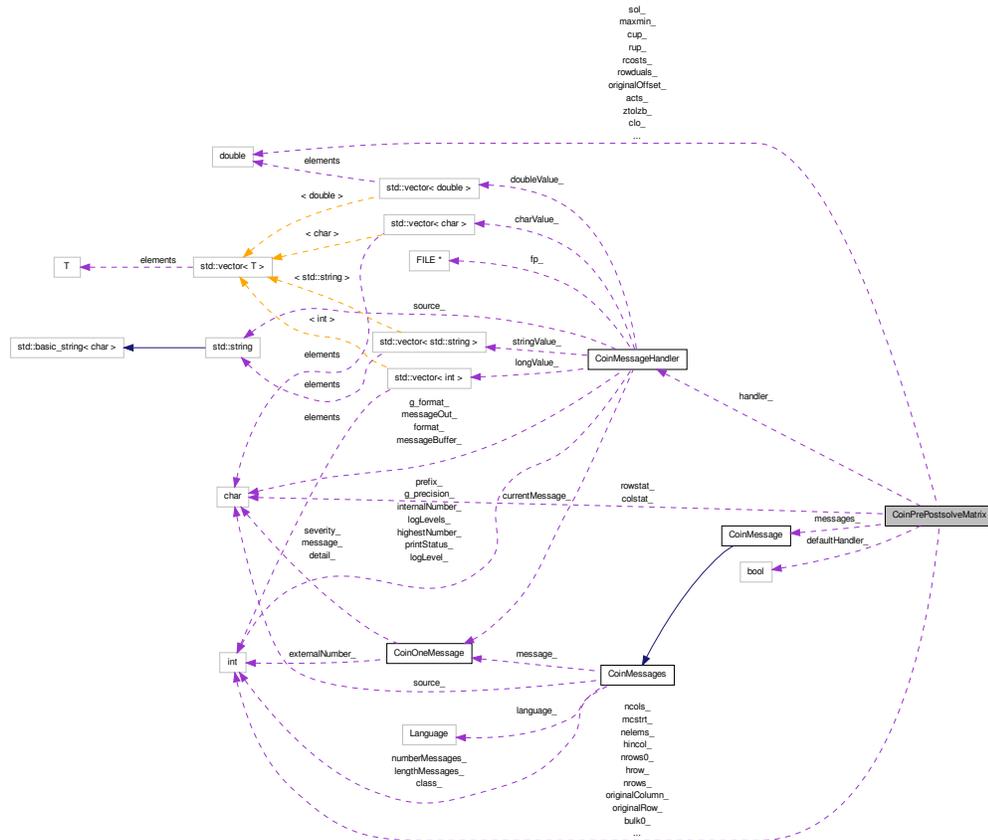
Collects all the information about the problem that is needed in both presolve and postsolve.

```
#include <CoinPresolveMatrix.hpp>
```

Inheritance diagram for CoinPrePostsolveMatrix:



Collaboration diagram for CoinPrePostsolveMatrix:



Public Types

- enum [Status](#)
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 bulkRatio)
ClpOsi constructor.
- **~CoinPrePostsolveMatrix** ()
Destructor.

Functions to work with variable status

Functions to work with the *CoinPrePostsolveMatrix::Status* enum and related vectors.

- 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 **rowIsBasic** (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 **columnIsBasic** (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_`, `coels_`.
- int * `hincol_`
Vector of column lengths.
- int * `hrow_`
Row indices (positional correspondence with `coels_`).
- double * `coels_`
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 *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.

Message handling

Uses the standard COIN approach: a default handler is installed, and the [CoinPrePost-solveMatrix](#) 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.

8.63.1 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.

8.63.2 Member Enumeration Documentation

8.63.2.1 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](#).

Definition at line 313 of file CoinPresolveMatrix.hpp.

8.63.3 Constructor & Destructor Documentation

8.63.3.1 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.

8.63.3.2 CoinPrePostsolveMatrix::CoinPrePostsolveMatrix (const OsiSolverInterface * *si*, int *ncols_*, int *nrows_*, CoinBigIndex *nelems_*)

Generic OSI constructor.

See OSI code for the definition.

8.63.3.3 `CoinPrePostsolveMatrix::CoinPrePostsolveMatrix (const ClpSimplex * si, int ncols_, int nrows_, CoinBigIndex nelems_, double bulkRatio)`

ClpOsi constructor.

See Clp code for the definition.

8.63.4 Member Function Documentation

8.63.4.1 `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.

8.63.4.2 `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`.

8.63.5 Member Data Documentation

8.63.5.1 `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 $2 * nelems0_$.

Definition at line 568 of file `CoinPresolveMatrix.hpp`.

8.63.5.2 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.

8.63.5.3 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.

8.63.5.4 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.

8.63.5.5 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.

8.63.5.6 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.

8.63.5.7 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.

8.63.5.8 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.

8.63.5.9 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.

8.63.5.10 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.

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

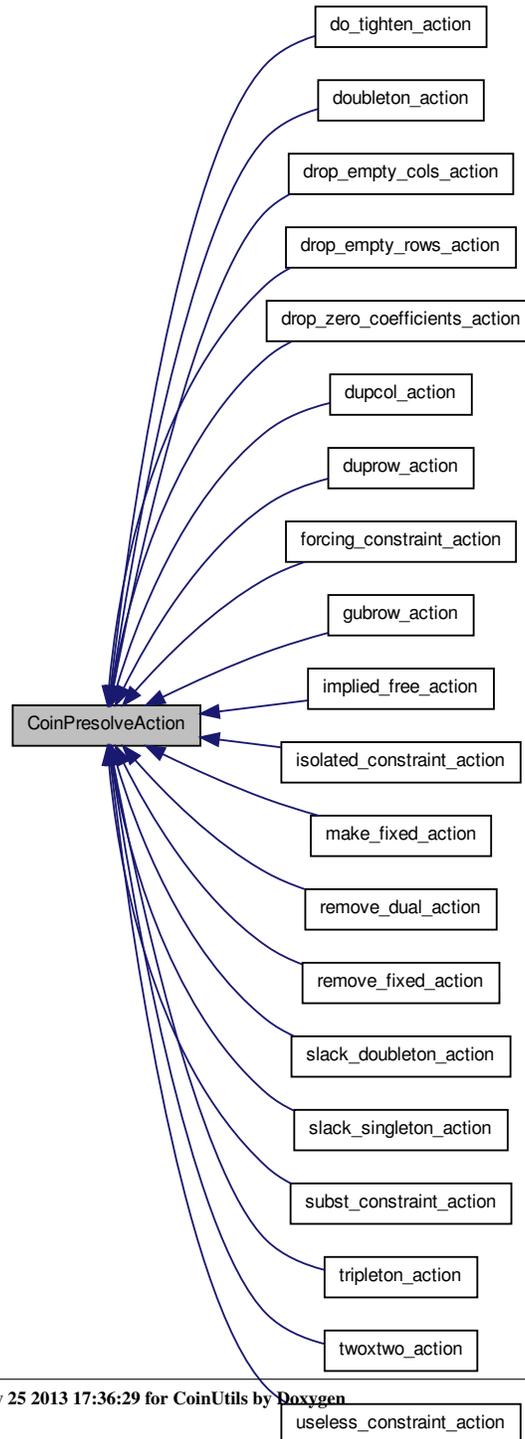
- [CoinPresolveMatrix.hpp](#)

8.64 CoinPresolveAction Class Reference

Abstract base class of all presolve routines.

```
#include <CoinPresolveMatrix.hpp>
```

Inheritance diagram for CoinPresolveAction:



Collaboration 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.

8.64.1 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`.

8.64.2 Constructor & Destructor Documentation

8.64.2.1 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.

8.64.2.2 virtual CoinPresolveAction::~CoinPresolveAction () [inline, virtual]

Virtual destructor.

Definition at line 195 of file CoinPresolveMatrix.hpp.

8.64.3 Member Function Documentation

8.64.3.1 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 x1C, 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.

8.64.3.2 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 [doubleton_action](#), [remove_dual_action](#), [dupcol_action](#), [duprow_action](#), [gubrow_action](#), [twoxtwo_action](#), [drop_empty_cols_action](#), [drop_empty_](#)

[rows_action](#), [remove_fixed_action](#), [make_fixed_action](#), [forcing_constraint_action](#), [implied_free_action](#), [isolated_constraint_action](#), [slack_doubleton_action](#), [slack_singleton_action](#), [subst_constraint_action](#), [do_tighten_action](#), [tripleton_action](#), [useless_constraint_action](#), and [drop_zero_coefficients_action](#).

8.64.4 Member Data Documentation

8.64.4.1 `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:

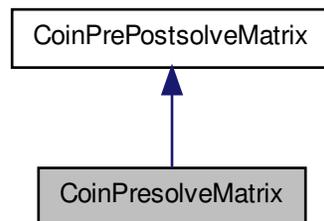
- [CoinPresolveMatrix.hpp](#)

8.65 CoinPresolveMatrix Class Reference

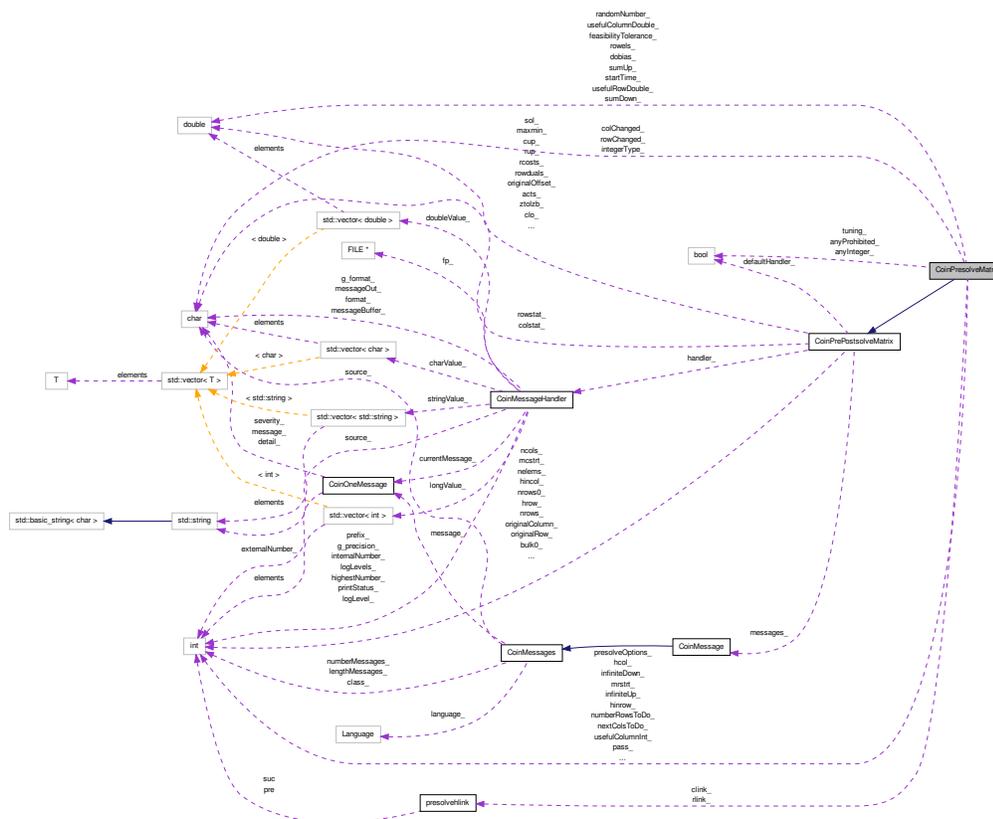
Augments [CoinPrePostsolveMatrix](#) with information about the problem that is only needed during presolve.

```
#include <CoinPresolveMatrix.hpp>
```

Inheritance diagram for `CoinPresolveMatrix`:



Collaboration 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 [colInfinite](#) (int i) const

Has column infinite ub (originally).

- void **unsetColInfinite** (int i)
Mark column as not infinite ub (originally).
- void **setColInfinite** (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.

8.65.1 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`.

8.65.2 Constructor & Destructor Documentation

8.65.2.1 `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`.

8.65.2.2 `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.

8.65.2.3 `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.

8.65.3 Member Function Documentation

8.65.3.1 `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.

8.65.3.2 `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`.

**8.65.3.3 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.

**8.65.3.4 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.

8.65.3.5 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`.

8.65.3.6 bool CoinPresolveMatrix::anyInteger () const [inline]

Check if there are any integer variables.

Consults the `anyInteger_` flag

Definition at line 978 of file `CoinPresolveMatrix.hpp`.

8.65.3.7 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.

8.65.3.8 void CoinPresolveMatrix::initColsToDo ()

Initialise the column ToDo lists.

Places all columns in the `colsToDo_` list except for columns marked as prohibited (*viz.* `colChanged_`).

8.65.3.9 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.

8.65.3.10 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`.

8.65.3.11 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`.

8.65.3.12 void CoinPresolveMatrix::initRowsToDo ()

Initialise the row ToDo lists.

Places all rows in the [rowsToDo_](#) list except for rows marked as prohibited (*viz.* [rowChanged_](#)).

8.65.3.13 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.

8.65.3.14 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.

8.65.3.15 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.

8.65.4 Friends And Related Function Documentation

8.65.4.1 void assignPresolveToPostsolve (CoinPresolveMatrix *& *preObj*) [friend]

Initialize a [CoinPostsolveMatrix](#) object, destroying the [CoinPresolveMatrix](#) object.

See [CoinPostsolveMatrix::assignPresolveToPostsolve](#).

8.65.5 Member Data Documentation

8.65.5.1 `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`.

8.65.5.2 `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`.

8.65.5.3 `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`.

8.65.5.4 `int CoinPresolveMatrix::maxSubstLevel_`

Maximum substitution level.

Used to control the execution of `subst` from `implied_free`

Definition at line 1087 of file `CoinPresolveMatrix.hpp`.

8.65.5.5 `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.

8.65.5.6 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.

8.65.5.7 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.

8.65.5.8 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.

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

- [CoinPresolveMatrix.hpp](#)
- [CoinPresolvePsdebug.hpp](#)

8.66 CoinPresolveMonitor Class Reference

Monitor a row or column for modification.

```
#include <CoinPresolveMonitor.hpp>
```


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.

8.66.1 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 [CoinPostsolve](#) matrix 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](#).

8.66.2 Constructor & Destructor Documentation

8.66.2.1 [CoinPresolveMonitor::CoinPresolveMonitor \(\)](#)

Default constructor.

Creates an empty monitor.

8.66.2.2 [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.

8.66.2.3 `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.

8.66.3 Member Function Documentation

8.66.3.1 `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`.

8.66.3.2 `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:

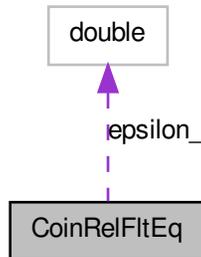
- `CoinPresolveMonitor.hpp`

8.67 CoinRelFltEq Class Reference

Equality to a scaled tolerance.

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

Collaboration diagram for CoinRelFltEq:



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.

8.67.1 Detailed Description

Equality to a scaled tolerance. Operands are considered equal if their difference is within a scaled epsilon calculated as $\text{epsilon}_* \cdot (1 + \text{CoinMax}(|f1|, |f2|))$.

Definition at line 110 of file CoinFloatEqual.hpp.

8.67.2 Constructor & Destructor Documentation

8.67.2.1 CoinRelFltEq::CoinRelFltEq () [inline]

Default constructor.

Default tolerance is 1.0e-10.

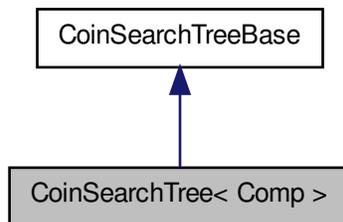
Definition at line 134 of file CoinFloatEqual.hpp.

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

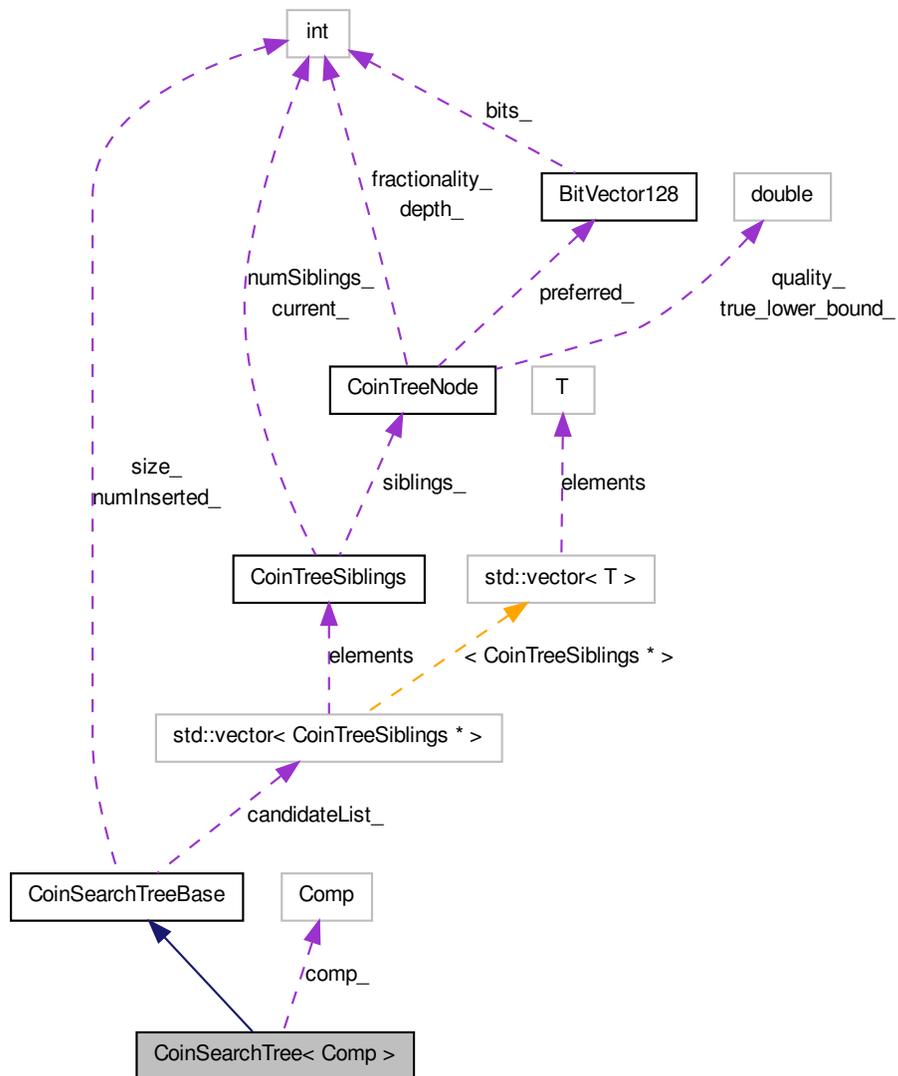
- [CoinFloatEqual.hpp](#)

8.68 CoinSearchTree< Comp > Class Template Reference

Inheritance diagram for CoinSearchTree< Comp >:



Collaboration diagram for CoinSearchTree< Comp >:



Protected Member Functions

- virtual void [fixTop](#) ()
After changing data in the top node, fix the heap.

8.68.1 Detailed Description

template<class Comp> class CoinSearchTree< Comp >

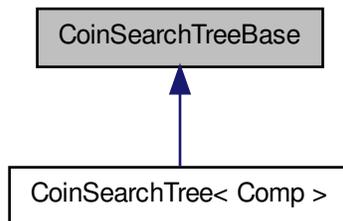
Definition at line 331 of file CoinSearchTree.hpp.

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

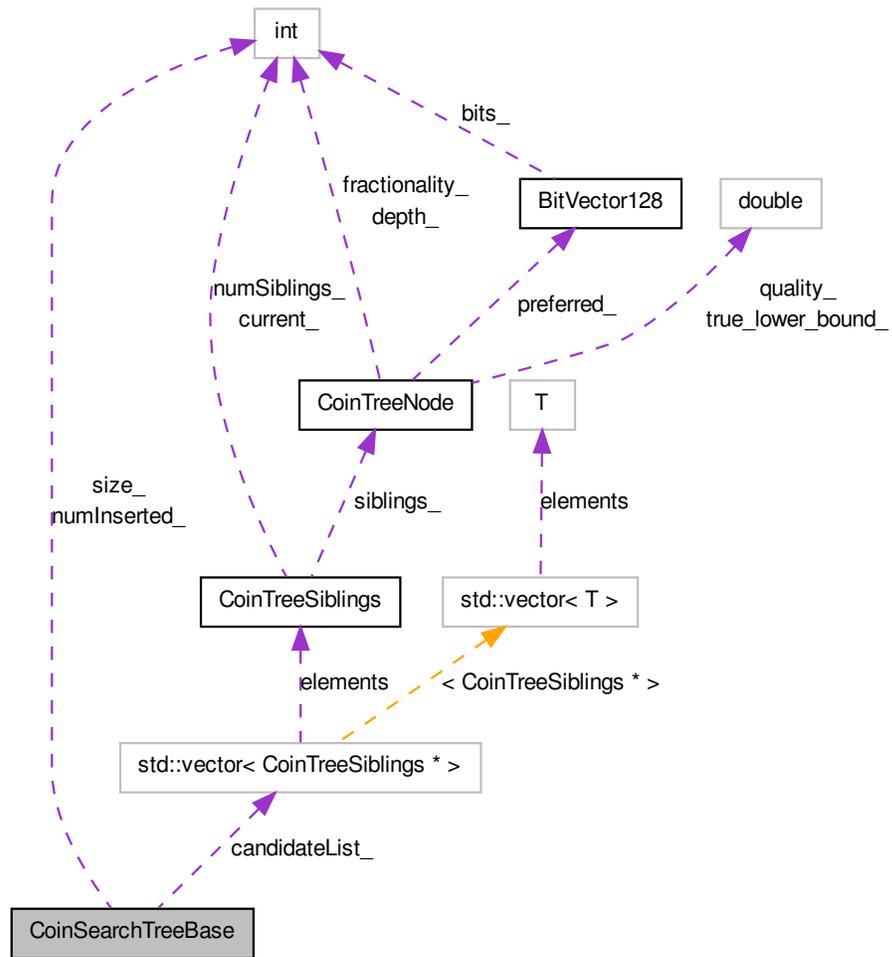
- CoinSearchTree.hpp

8.69 CoinSearchTreeBase Class Reference

Inheritance diagram for CoinSearchTreeBase:



Collaboration diagram for CoinSearchTreeBase:



Public Member Functions

- `void pop ()`

pop will advance the next pointer among the siblings on the top and then moves the top to its correct position.

8.69.1 Detailed Description

Definition at line 217 of file CoinSearchTree.hpp.

8.69.2 Member Function Documentation

8.69.2.1 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.

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

- CoinSearchTree.hpp

8.70 CoinSearchTreeCompareBest Struct Reference

Best first search.

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

8.70.1 Detailed Description

Best first search.

Definition at line 207 of file CoinSearchTree.hpp.

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

- CoinSearchTree.hpp

8.71 CoinSearchTreeCompareBreadth Struct Reference

8.71.1 Detailed Description

Definition at line 197 of file CoinSearchTree.hpp.

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

- CoinSearchTree.hpp

8.72 CoinSearchTreeCompareDepth Struct Reference

Depth First Search.

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

8.72.1 Detailed Description

Depth First Search.

Definition at line 178 of file CoinSearchTree.hpp.

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

- CoinSearchTree.hpp

8.73 CoinSearchTreeComparePreferred Struct Reference

Function objects to compare search tree nodes.

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

8.73.1 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.

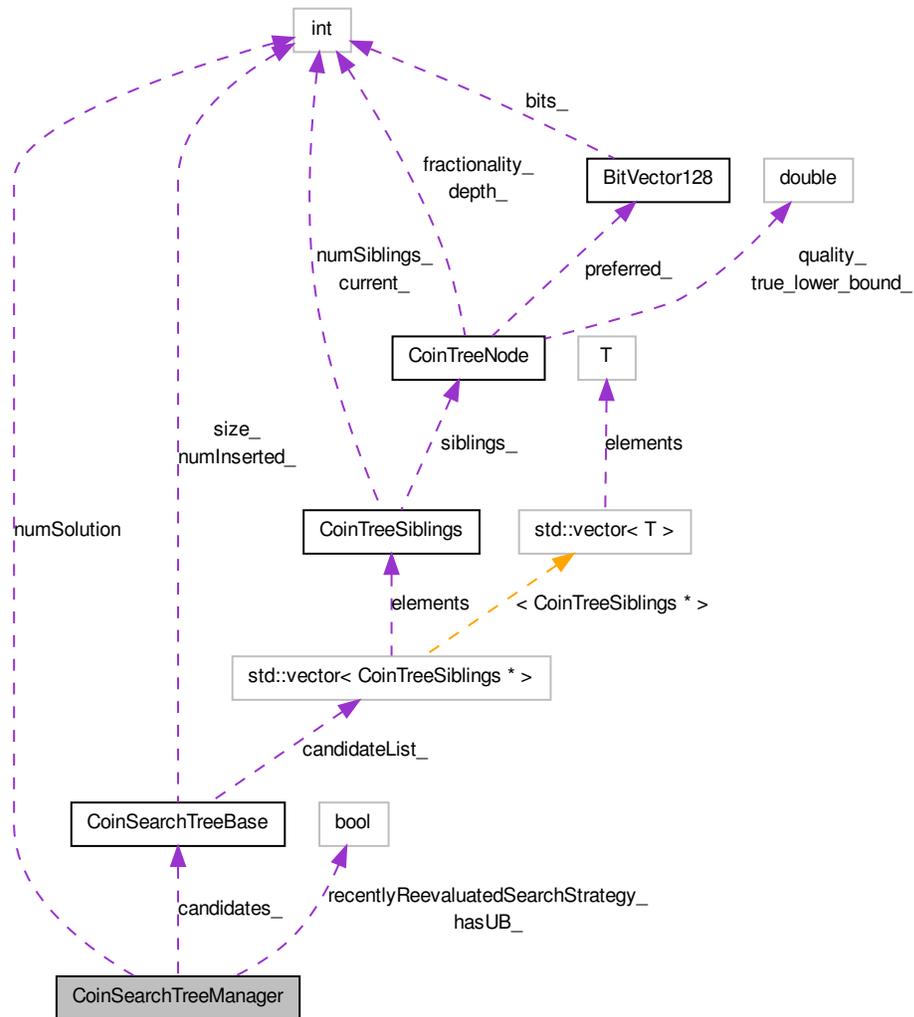
Definition at line 152 of file CoinSearchTree.hpp.

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

- CoinSearchTree.hpp

8.74 CoinSearchTreeManager Class Reference

Collaboration diagram for CoinSearchTreeManager:



8.74.1 Detailed Description

Definition at line 404 of file CoinSearchTree.hpp.

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

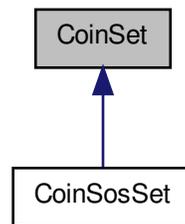
- CoinSearchTree.hpp

8.75 CoinSet Class Reference

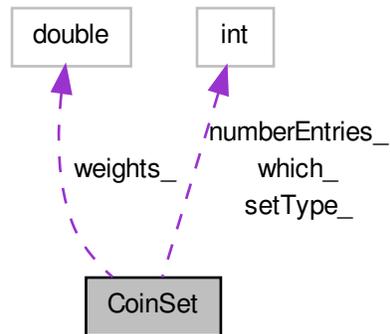
Very simple class for containing data on set.

```
#include <CoinMpsIO.hpp>
```

Inheritance diagram for CoinSet:



Collaboration 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.

8.75.1 Detailed Description

Very simple class for containing data on set.

Definition at line 221 of file CoinMpsIO.hpp.

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

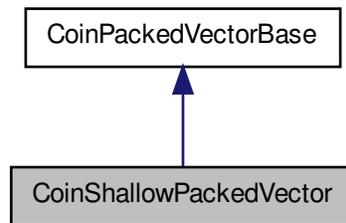
- CoinMpsIO.hpp

8.76 CoinShallowPackedVector Class Reference

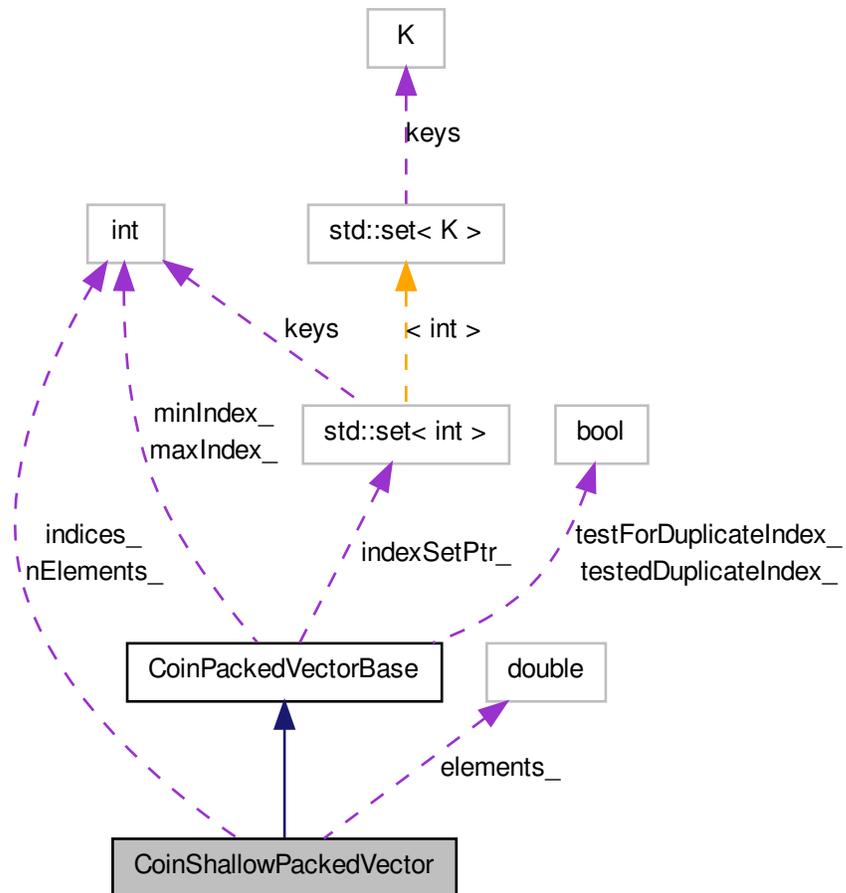
Shallow Sparse Vector.

```
#include <CoinShallowPackedVector.hpp>
```

Inheritance diagram for CoinShallowPackedVector:



Collaboration 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.

8.76.1 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.

8.76.2 Constructor & Destructor Documentation

8.76.2.1 CoinShallowPackedVector::CoinShallowPackedVector (bool *testForDuplicateIndex* = *true*)

Default constructor.

8.76.2.2 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.

8.76.2.3 CoinShallowPackedVector::CoinShallowPackedVector (const CoinPackedVectorBase &)

Copy constructor from the base class.

8.76.2.4 CoinShallowPackedVector::CoinShallowPackedVector (const CoinShallowPackedVector &)

Copy constructor.

8.76.2.5 virtual CoinShallowPackedVector::~~CoinShallowPackedVector () [inline, virtual]

Destructor.

Definition at line 122 of file CoinShallowPackedVector.hpp.

8.76.3 Member Function Documentation

8.76.3.1 CoinShallowPackedVector& CoinShallowPackedVector::operator= (const CoinShallowPackedVector & x)

Assignment operator.

8.76.3.2 CoinShallowPackedVector& CoinShallowPackedVector::operator= (const CoinPackedVectorBase & x)

Assignment operator from a [CoinPackedVectorBase](#).

Reimplemented from [CoinPackedVectorBase](#).

8.76.4 Friends And Related Function Documentation

8.76.4.1 void CoinShallowPackedVectorUnitTest () [friend]

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

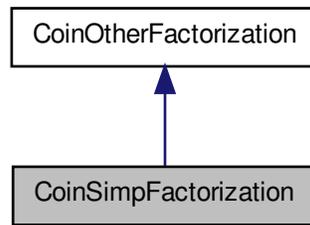
The only reason for it not to be a member method is that this way it doesn't have to be compiled into the library. And that's a gain, because the library should be compiled with optimization on, but this method should be compiled with debugging.

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

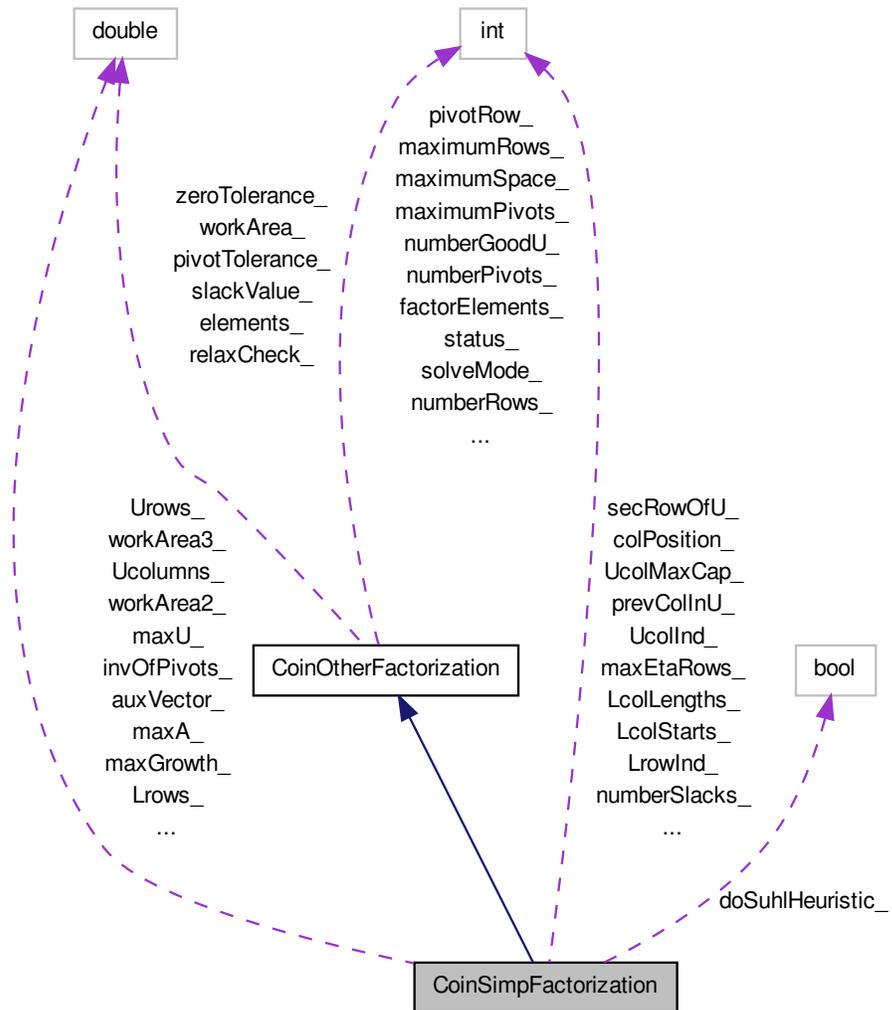
- CoinShallowPackedVector.hpp

8.77 CoinSimpFactorization Class Reference

Inheritance diagram for CoinSimpFactorization:



Collaboration 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 numberOfRows, int numberOfColumns, `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 numberOfColumns)
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 Shul 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 first basis

8.77.1 Detailed Description

Definition at line 38 of file CoinSimpFactorization.hpp.

8.77.2 Member Function Documentation

8.77.2.1 `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](#).

8.77.2.2 `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 regionSparse starts as zero and is zero at end.

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

Implements [CoinOtherFactorization](#).

8.77.2.3 `int CoinSimpFactorization::checkPivot (double saveFromU, double oldPivot) const [protected]`

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

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

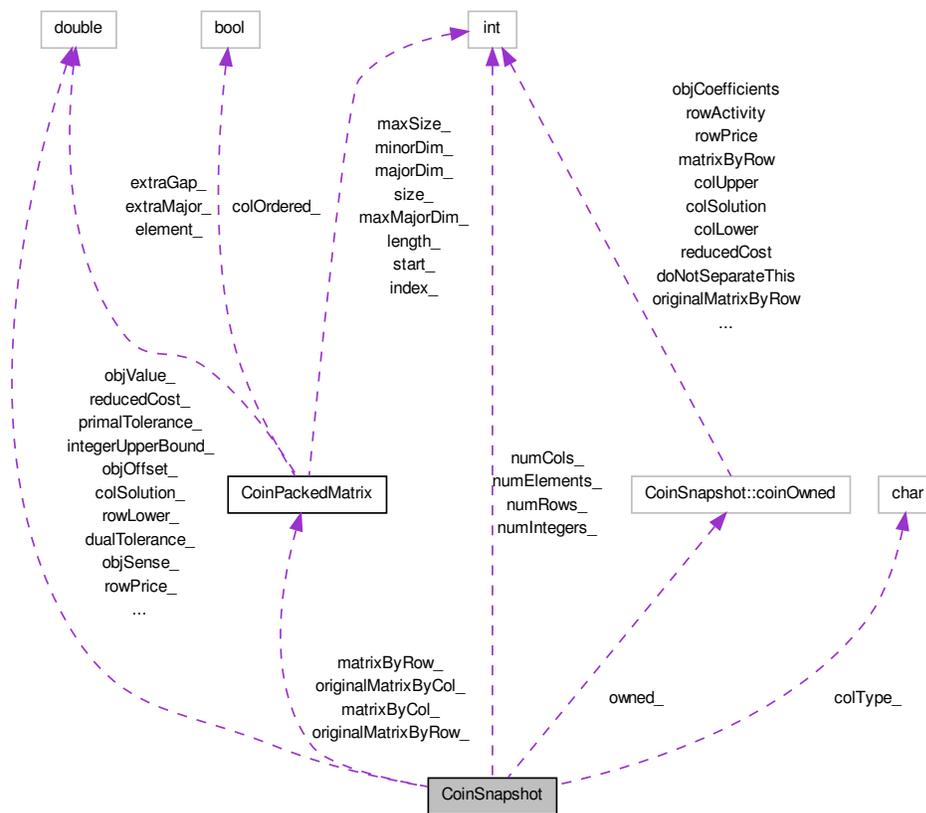
- CoinSimpFactorization.hpp

8.78 CoinSnapshot Class Reference

NON Abstract Base Class for interfacing with cut generators or branching code or .

```
#include <CoinSnapshot.hpp>
```

Collaboration diagram for CoinSnapshot:



Classes

- struct **coinOwned**

To say whether arrays etc are owned by [CoinSnapshot](#).

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 colIndex) const`
Return true if column is integer.
- `bool isIntegerNonBinary (int colIndex) const`
Return true if variable is general integer.
- `bool isFreeBinary (int colIndex) 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. $\sum c_{sub\ j} * x_{sub\ j} - objValue = objOffset$.
- 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 an 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.

8.78.1 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.

8.78.2 Member Function Documentation

8.78.2.1 `double CoinSnapshot::getObjValue () const [inline]`

Get objective function value - including any offset i.e.

$\text{sum } c_{\text{sub } j} * x_{\text{sub } j} - \text{objValue} = \text{objOffset}$

Definition at line 157 of file CoinSnapshot.hpp.

8.78.2.2 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

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

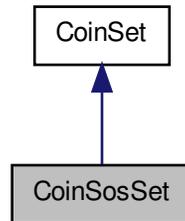
- CoinSnapshot.hpp

8.79 CoinSosSet Class Reference

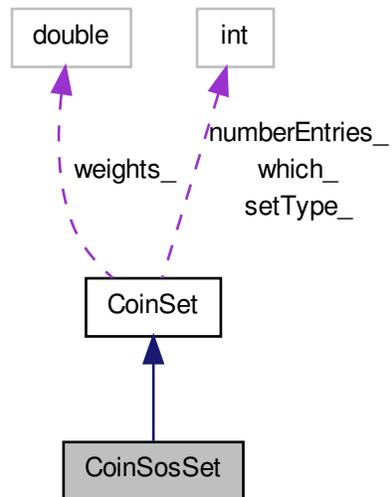
Very simple class for containing SOS set.

```
#include <CoinMpsIO.hpp>
```

Inheritance diagram for CoinSosSet:



Collaboration diagram for CoinSosSet:



Public Member Functions

Constructor and destructor

- [CoinSosSet](#) (int numberEntries, const int *which, const double *weights, int type)

Constructor.

- virtual [~CoinSosSet](#) ()

Destructor.

8.79.1 Detailed Description

Very simple class for containing SOS set.

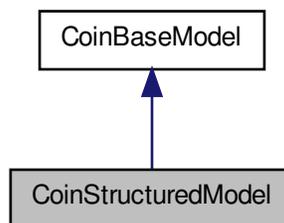
Definition at line 286 of file CoinMpsIO.hpp.

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

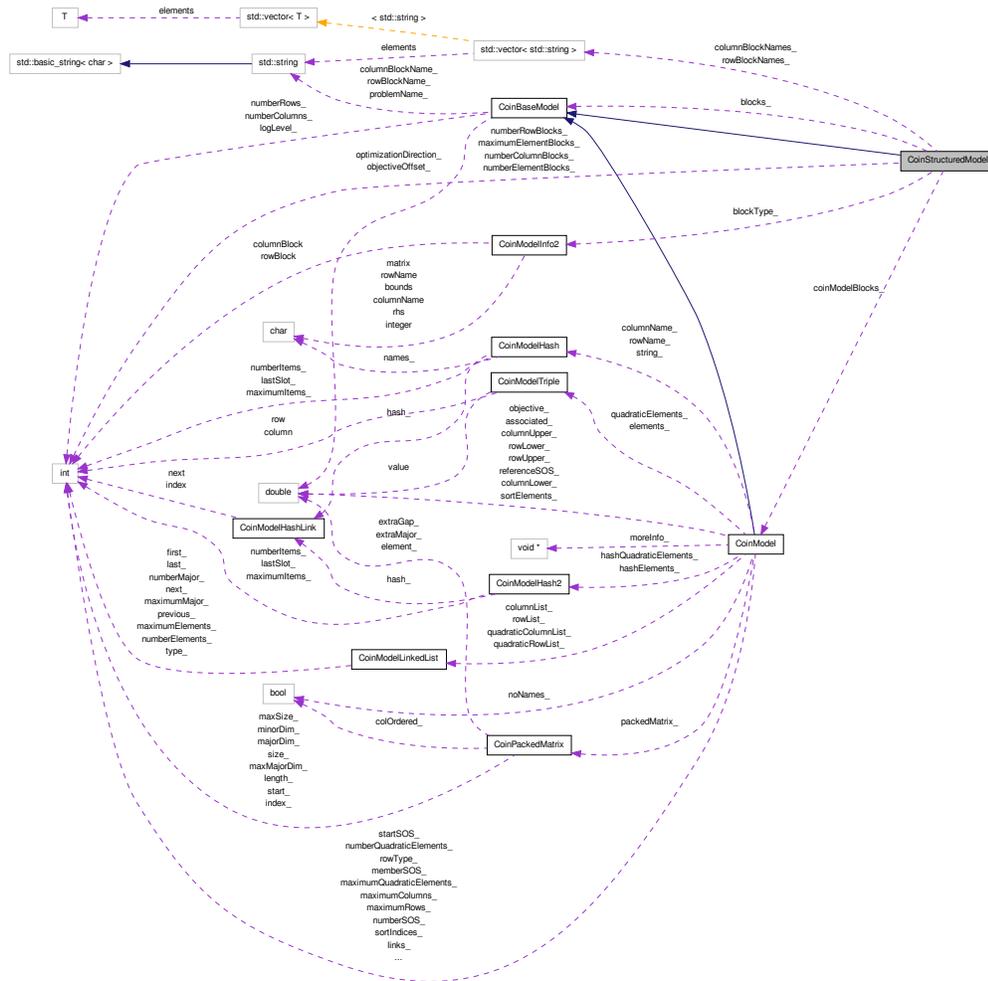
- CoinMpsIO.hpp

8.80 CoinStructuredModel Class Reference

Inheritance diagram for CoinStructuredModel:



Collaboration 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 given as parameters 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.

8.80.1 Detailed Description

Definition at line 36 of file CoinStructuredModel.hpp.

8.80.2 Constructor & Destructor Documentation

8.80.2.1 CoinStructuredModel::CoinStructuredModel ()

Default constructor.

8.80.2.2 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

8.80.2.3 CoinStructuredModel::CoinStructuredModel (const CoinStructuredModel &)

The copy constructor.

8.80.3 Member Function Documentation

8.80.3.1 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)

8.80.3.2 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)

8.80.3.3 `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

8.80.3.4 `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

compression can be set to three values to indicate what kind of file should be written

- 0: plain text (default)
- 1: gzip compressed (.gz is appended to filename)
- 2: bzip2 compressed (.bz2 is appended to filename) (TODO)

If the library was not compiled with the requested compression then writeMps falls back to writing a plain text file.

formatType specifies the precision to used for values in the MPS file

- 0: normal precision (default)
- 1: extra accuracy
- 2: IEEE hex

numberAcross 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

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

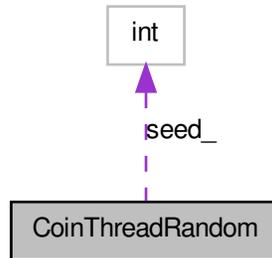
- CoinStructuredModel.hpp

8.81 CoinThreadRandom Class Reference

Class for thread specific random numbers.

```
#include <CoinHelperFunctions.hpp>
```

Collaboration diagram for CoinThreadRandom:



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.

8.81.1 Detailed Description

Class for thread specific random numbers.

Definition at line 952 of file CoinHelperFunctions.hpp.

8.81.2 Constructor & Destructor Documentation

8.81.2.1 CoinThreadRandom::CoinThreadRandom () [inline]

Default constructor.

Definition at line 958 of file CoinHelperFunctions.hpp.

8.81.2.2 CoinThreadRandom::CoinThreadRandom (int seed) [inline]

Constructor with seed.

Definition at line 961 of file CoinHelperFunctions.hpp.

8.81.3 Member Function Documentation

8.81.3.1 void CoinThreadRandom::setSeed (int seed) [inline]

Set seed.

Definition at line 985 of file CoinHelperFunctions.hpp.

8.81.3.2 unsigned int CoinThreadRandom::getSeed () const [inline]

Get seed.

Definition at line 990 of file CoinHelperFunctions.hpp.

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

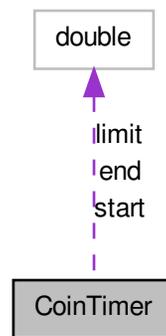
- CoinHelperFunctions.hpp

8.82 CoinTimer Class Reference

This class implements a timer that also implements a tracing functionality.

```
#include <CoinTime.hpp>
```

Collaboration diagram for CoinTimer:



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.

8.82.1 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.

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

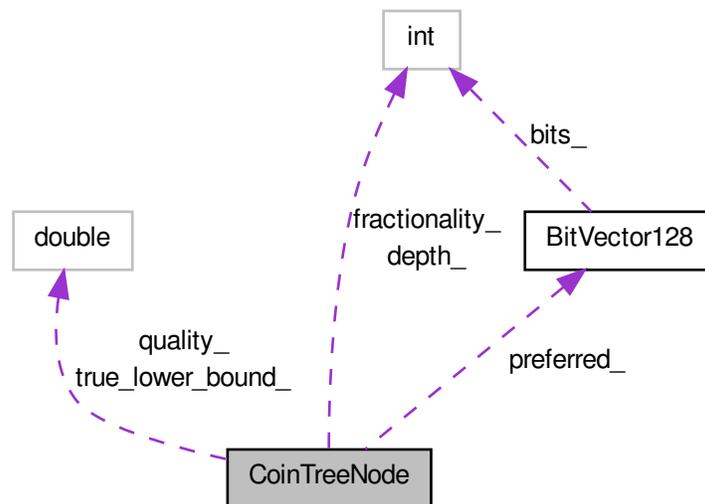
- CoinTime.hpp

8.83 CoinTreeNode Class Reference

A class from which the real tree nodes should be derived from.

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

Collaboration diagram for CoinTreeNode:



8.83.1 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.

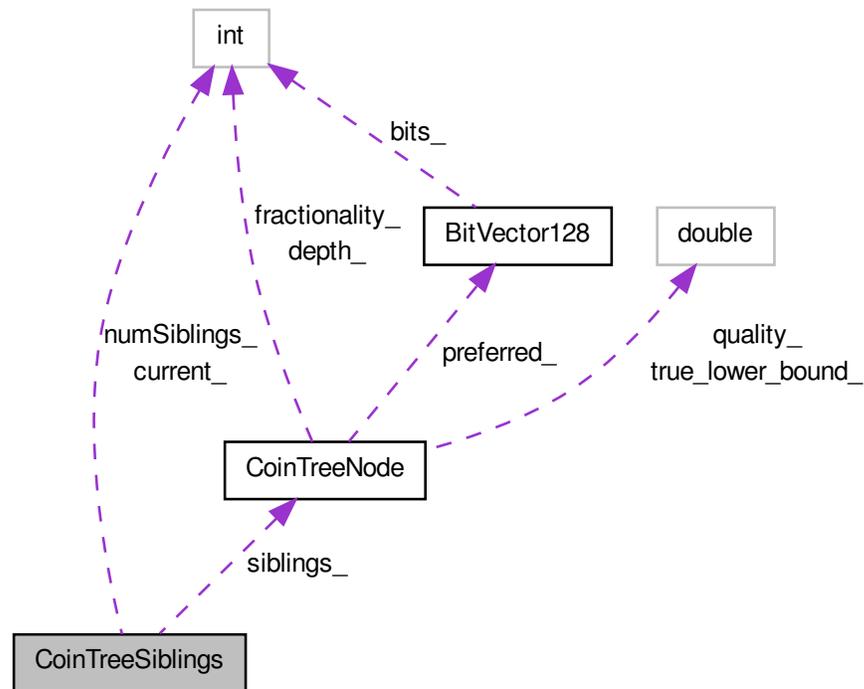
Definition at line 42 of file CoinSearchTree.hpp.

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

- CoinSearchTree.hpp

8.84 CoinTreeSiblings Class Reference

Collaboration diagram for CoinTreeSiblings:



Public Member Functions

- bool [advanceNode](#) ()
returns false if cannot be advanced

8.84.1 Detailed Description

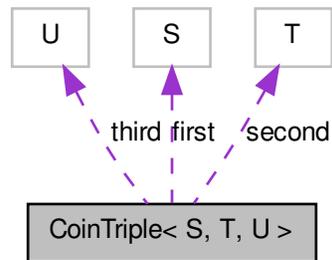
Definition at line 110 of file CoinSearchTree.hpp.

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

- CoinSearchTree.hpp

8.85 CoinTriple< S, T, U > Class Template Reference

Collaboration diagram for CoinTriple< S, T, U >:



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.

8.85.1 Detailed Description

template<class S, class T, class U> class CoinTriple< S, T, U >

Definition at line 459 of file CoinSort.hpp.

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

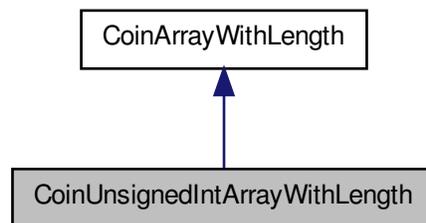
- CoinSort.hpp

8.86 CoinUnsignedIntArrayWithLength Class Reference

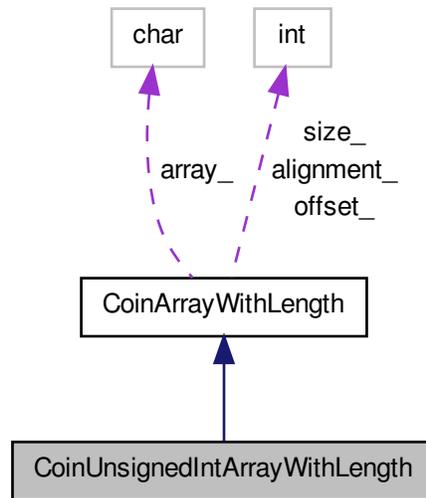
unsigned int * version

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

Inheritance diagram for CoinUnsignedIntArrayWithLength:



Collaboration 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.

8.86.1 Detailed Description

unsigned int * version

Definition at line 887 of file CoinIndexedVector.hpp.

8.86.2 Constructor & Destructor Documentation

8.86.2.1 [CoinUnsignedIntArrayWithLength::CoinUnsignedIntArrayWithLength](#) (const [CoinUnsignedIntArrayWithLength](#) & rhs) [inline]

Copy constructor.

Definition at line 929 of file CoinIndexedVector.hpp.

8.86.3 Member Function Documentation

8.86.3.1 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:

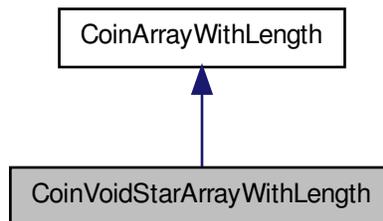
- CoinIndexedVector.hpp

8.87 CoinVoidStarArrayWithLength Class Reference

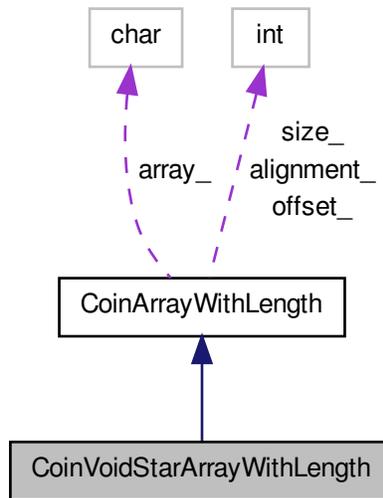
void * version

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

Inheritance diagram for CoinVoidStarArrayWithLength:



Collaboration 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.

8.87.1 Detailed Description

void * version

Definition at line 941 of file CoinIndexedVector.hpp.

8.87.2 Constructor & Destructor Documentation

8.87.2.1 [CoinVoidStarArrayWithLength::CoinVoidStarArrayWithLength](#) (const [CoinVoidStarArrayWithLength](#) & rhs) [inline]

Copy constructor.

Definition at line 983 of file CoinIndexedVector.hpp.

8.87.3 Member Function Documentation

8.87.3.1 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:

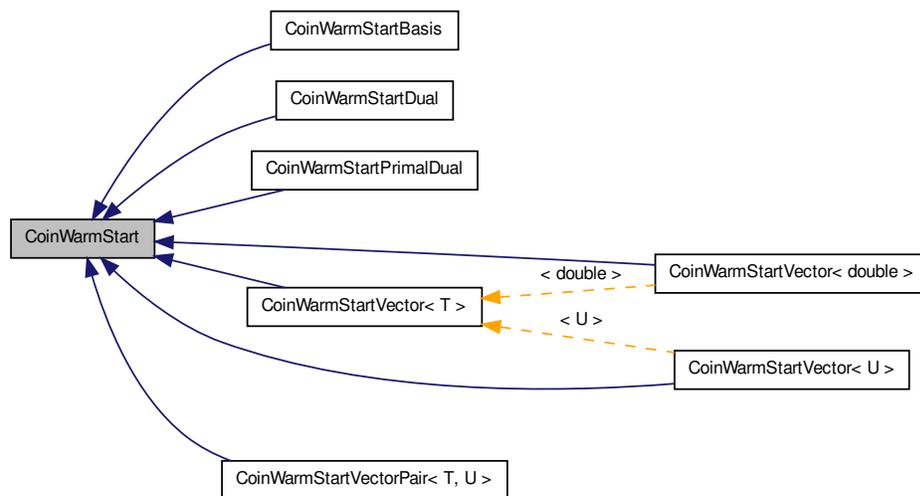
- CoinIndexedVector.hpp

8.88 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'

8.88.1 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`.

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

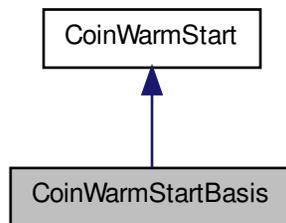
- [CoinWarmStart.hpp](#)

8.89 CoinWarmStartBasis Class Reference

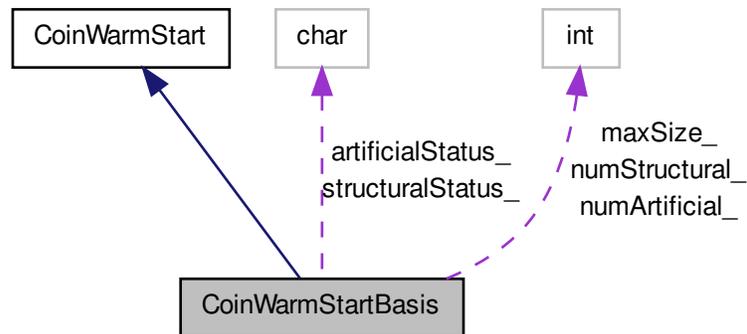
The default COIN simplex (basis-oriented) warm start class.

```
#include <CoinWarmStartBasis.hpp>
```

Inheritance diagram for `CoinWarmStartBasis`:



Collaboration diagram for CoinWarmStartBasis:



Public Types

- enum `Status` { `isFree` = 0x00, `basic` = 0x01, `atUpperBound` = 0x02, `atLowerBound` = 0x03 }
- 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 old-CWS) 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.

8.89.1 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).

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

8.89.2 Member Enumeration Documentation

8.89.2.1 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`.

8.89.3 Constructor & Destructor Documentation

8.89.3.1 CoinWarmStartBasis::CoinWarmStartBasis ()

Default constructor.

Creates a warm start object representing an empty basis (0 rows, 0 columns).

8.89.3.2 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.

8.89.4 Member Function Documentation

8.89.4.1 int CoinWarmStartBasis::numberBasicStructurals () const

Return the number of basic structurals.

A fast test for an all-slack basis.

8.89.4.2 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.

8.89.4.3 `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](#).

8.89.4.4 `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](#).

8.89.4.5 `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`.

8.89.4.6 `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.)

8.89.4.7 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.

8.89.4.8 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.

8.89.4.9 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.

8.89.4.10 `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.

8.89.4.11 `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[]`.

8.89.5 Member Data Documentation

8.89.5.1 `char* CoinWarmStartBasis::structuralStatus_ [protected]`

The status of the structural variables.

Definition at line 340 of file `CoinWarmStartBasis.hpp`.

8.89.5.2 `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:

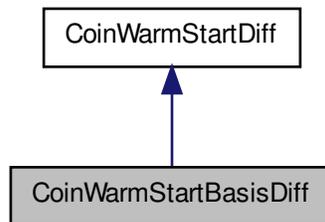
- CoinWarmStartBasis.hpp

8.90 CoinWarmStartBasisDiff Class Reference

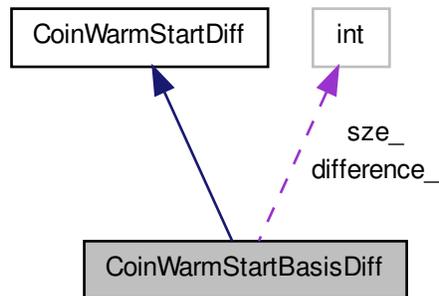
A 'diff' between two [CoinWarmStartBasis](#) objects.

```
#include <CoinWarmStartBasis.hpp>
```

Inheritance diagram for CoinWarmStartBasisDiff:



Collaboration 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!

8.90.1 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_`

Definition at line 395 of file `CoinWarmStartBasis.hpp`.

8.90.2 Constructor & Destructor Documentation

8.90.2.1 [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`.

8.90.2.2 [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.

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

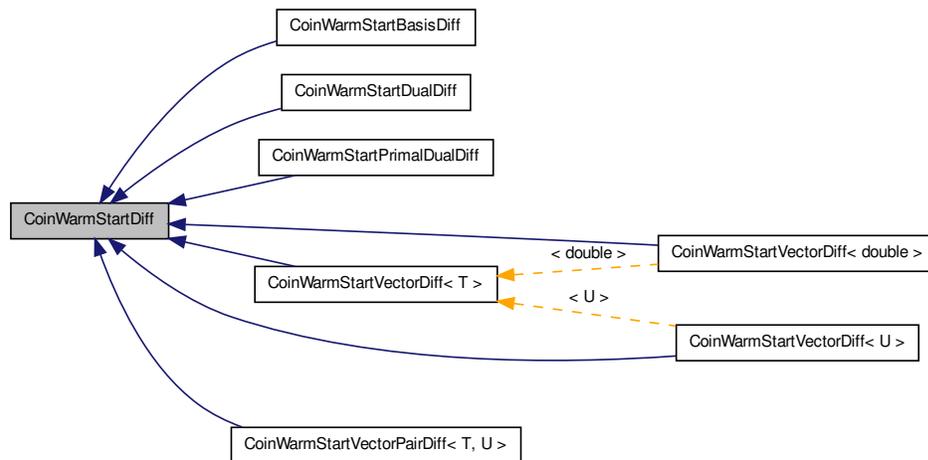
- CoinWarmStartBasis.hpp

8.91 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'

8.91.1 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`.

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

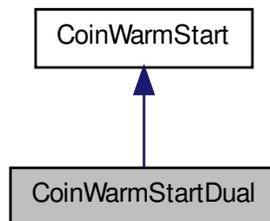
- [CoinWarmStart.hpp](#)

8.92 CoinWarmStartDual Class Reference

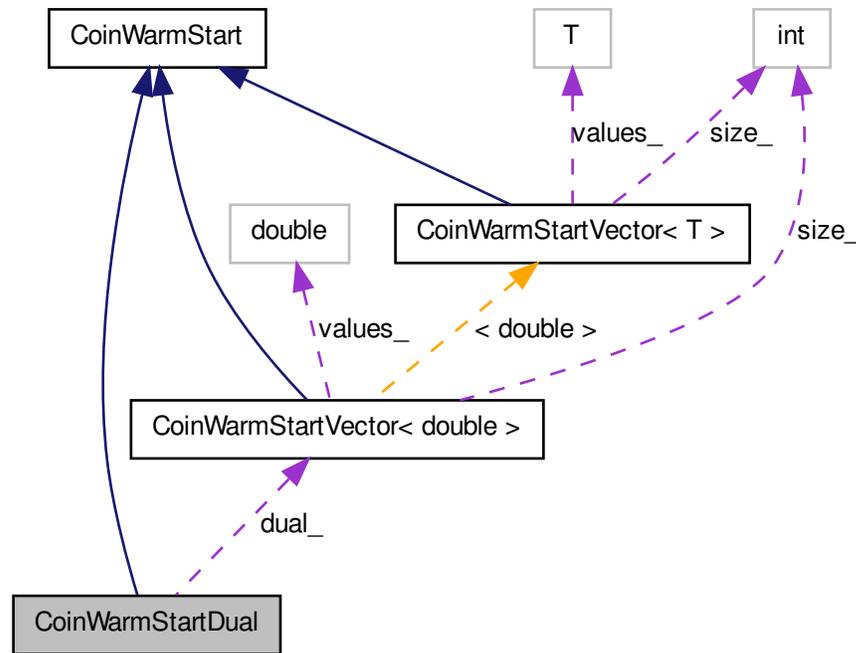
WarmStart information that is only a dual vector.

```
#include <CoinWarmStartDual.hpp>
```

Inheritance diagram for `CoinWarmStartDual`:



Collaboration 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.
- `virtual CoinWarmStart * clone () const`
'Virtual constructor'

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.

8.92.1 Detailed Description

WarmStart information that is only a dual vector.

Definition at line 18 of file `CoinWarmStartDual.hpp`.

8.92.2 Member Function Documentation

8.92.2.1 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 22 of file `CoinWarmStartDual.hpp`.

8.92.2.2 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](#).

8.92.2.3 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:

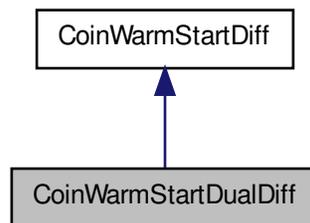
- [CoinWarmStartDual.hpp](#)

8.93 CoinWarmStartDualDiff Class Reference

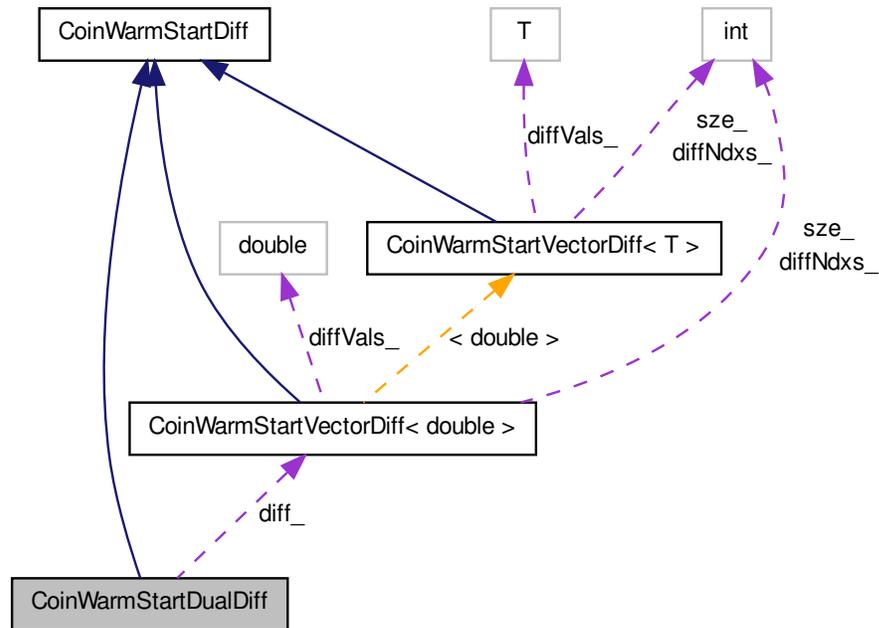
A 'diff' between two [CoinWarmStartDual](#) objects.

```
#include <CoinWarmStartDual.hpp>
```

Inheritance diagram for [CoinWarmStartDualDiff](#):



Collaboration 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.

8.93.1 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`.

8.93.2 Constructor & Destructor Documentation

8.93.2.1 [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`.

8.93.2.2 [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 they make *their* copy constructor protected or private.

Definition at line 142 of file CoinWarmStartDual.hpp.

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

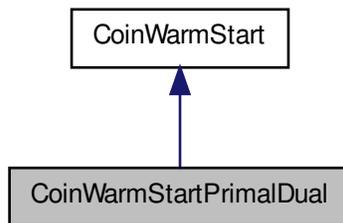
- CoinWarmStartDual.hpp

8.94 CoinWarmStartPrimalDual Class Reference

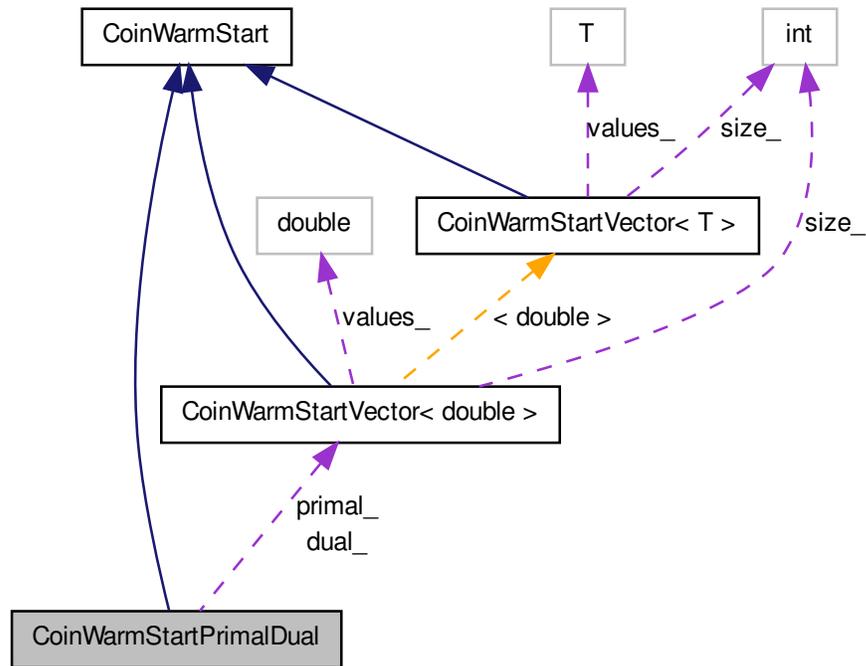
WarmStart information that is only a dual vector.

```
#include <CoinWarmStartPrimalDual.hpp>
```

Inheritance diagram for CoinWarmStartPrimalDual:



Collaboration 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.
- void `clear` ()
Clear the data.
- virtual `CoinWarmStart * clone` () const
'Virtual constructor'

PrimalDual warm start 'diff' methods

- virtual `CoinWarmStartDiff * generateDiff` (const `CoinWarmStart *const old-CWS`) 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.

8.94.1 Detailed Description

WarmStart information that is only a dual vector.

Definition at line 18 of file CoinWarmStartPrimalDual.hpp.

8.94.2 Member Function Documentation

8.94.2.1 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.

8.94.2.2 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.

8.94.2.3 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](#).

8.94.2.4 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:

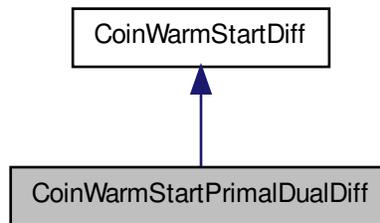
- CoinWarmStartPrimalDual.hpp

8.95 CoinWarmStartPrimalDualDiff Class Reference

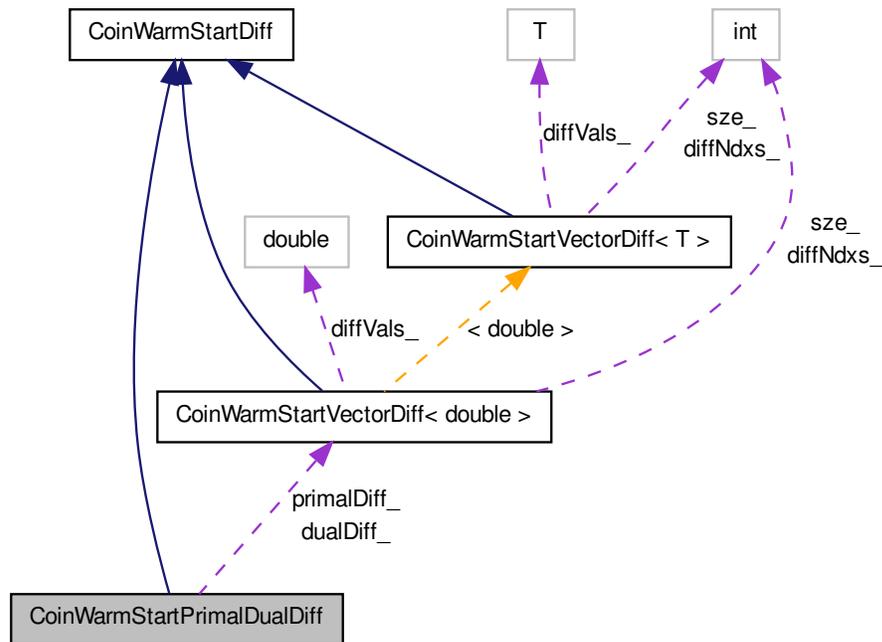
A ‘diff’ between two [CoinWarmStartPrimalDual](#) objects.

```
#include <CoinWarmStartPrimalDual.hpp>
```

Inheritance diagram for CoinWarmStartPrimalDualDiff:



Collaboration 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.

8.95.1 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`.

8.95.2 Constructor & Destructor Documentation

8.95.2.1 `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`.

8.95.2.2 `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.

8.95.3 Member Function Documentation

8.95.3.1 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.

8.95.3.2 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.

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

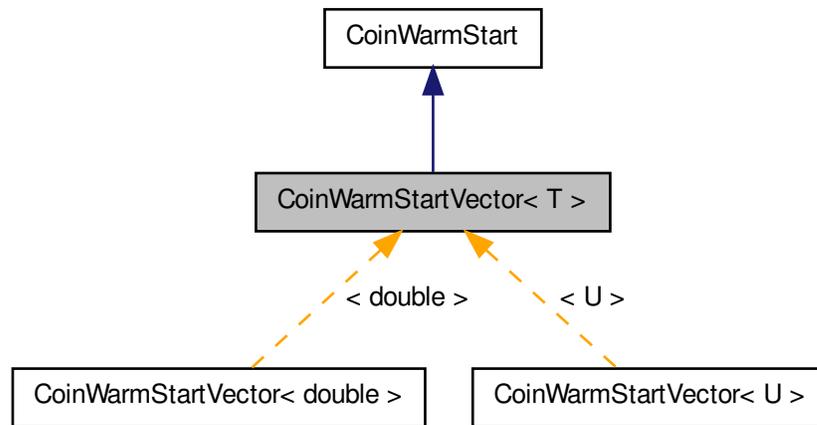
- CoinWarmStartPrimalDual.hpp

8.96 CoinWarmStartVector< T > Class Template Reference

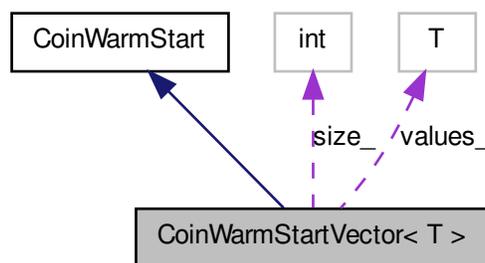
WarmStart information that is only a vector.

```
#include <CoinWarmStartVector.hpp>
```

Inheritance diagram for CoinWarmStartVector< T >:



Collaboration 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.
- virtual CoinWarmStart * `clone` () const
'Virtual constructor'
- void `clear` ()
Clear the data.

Vector warm start 'diff' methods

- virtual CoinWarmStartDiff * `generateDiff` (const CoinWarmStart *const old-CWS) 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.

8.96.1 Detailed Description

`template<typename T> class CoinWarmStartVector< T >`

WarmStart information that is only a vector.

Definition at line 26 of file CoinWarmStartVector.hpp.

8.96.2 Member Function Documentation

8.96.2.1 `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.

8.96.2.2 `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.

8.96.2.3 `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.

8.96.2.4 `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:

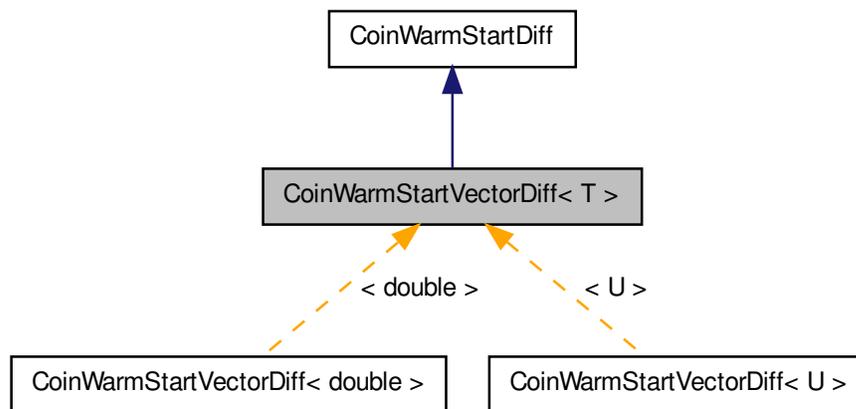
- CoinWarmStartVector.hpp

8.97 CoinWarmStartVectorDiff< T > Class Template Reference

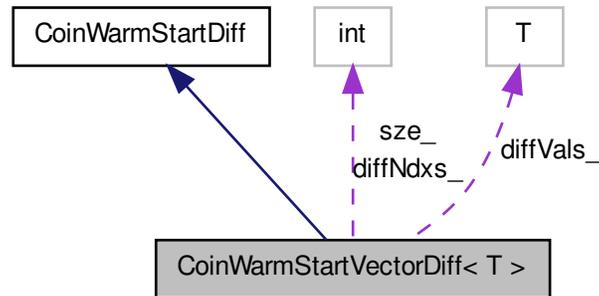
A 'diff' between two [CoinWarmStartVector](#) objects.

```
#include <CoinWarmStartVector.hpp>
```

Inheritance diagram for CoinWarmStartVectorDiff< T >:



Collaboration diagram for CoinWarmStartVectorDiff< T >:



Public Member Functions

- virtual [CoinWarmStartDiff](#) * [clone](#) () const
'Virtual constructor'
- virtual [CoinWarmStartVectorDiff](#) & [operator=](#) (const [CoinWarmStartVectorDiff](#)< T > &rhs)
Assignment.
- virtual [~CoinWarmStartVectorDiff](#) ()
Destructor.
- [CoinWarmStartVectorDiff](#) ()
Default constructor.
- [CoinWarmStartVectorDiff](#) (const [CoinWarmStartVectorDiff](#)< T > &rhs)
Copy constructor.
- [CoinWarmStartVectorDiff](#) (int [sze](#), const unsigned int *const [diffNdxs](#), const T *const [diffVals](#))
Standard constructor.
- void [clear](#) ()
Clear the data.

8.97.1 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`.

8.97.2 Constructor & Destructor Documentation

8.97.2.1 **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`.

8.97.3 Member Function Documentation

8.97.3.1 **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.

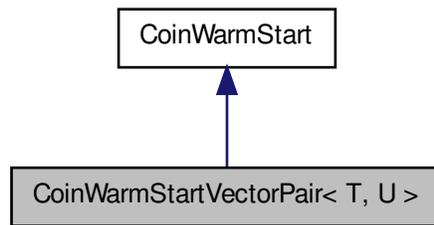
Definition at line 204 of file `CoinWarmStartVector.hpp`.

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

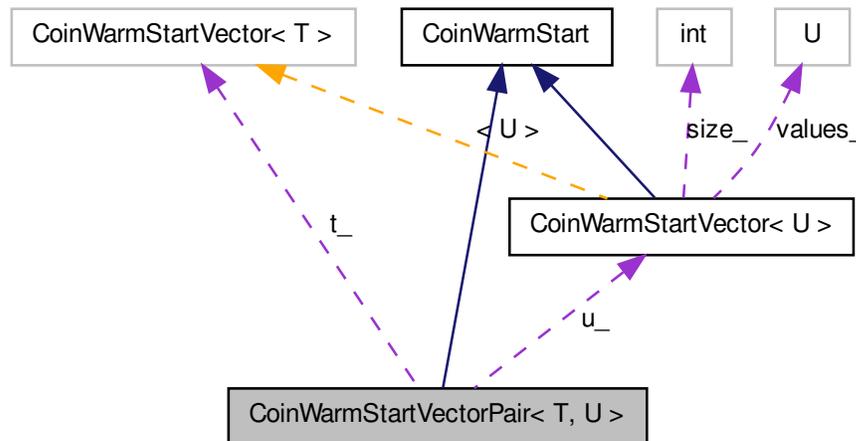
- `CoinWarmStartVector.hpp`

8.98 CoinWarmStartVectorPair< T, U > Class Template Reference

Inheritance diagram for CoinWarmStartVectorPair< T, U >:



Collaboration diagram for CoinWarmStartVectorPair< T, U >:



Public Member Functions

- virtual `CoinWarmStart` * `clone` () const
'Virtual constructor'

8.98.1 Detailed Description

`template<typename T, typename U> class CoinWarmStartVectorPair< T, U >`

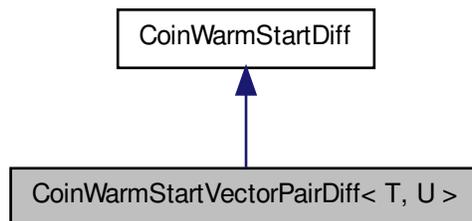
Definition at line 229 of file `CoinWarmStartVector.hpp`.

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

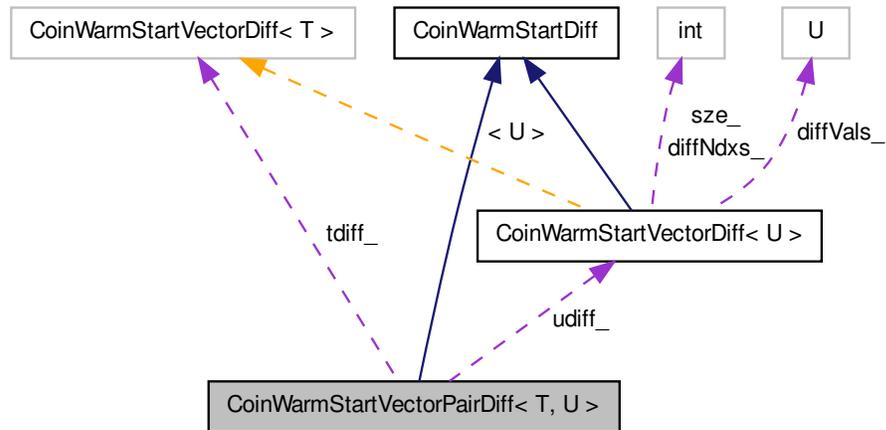
- `CoinWarmStartVector.hpp`

8.99 CoinWarmStartVectorPairDiff< T, U > Class Template Reference

Inheritance diagram for `CoinWarmStartVectorPairDiff< T, U >`:



Collaboration diagram for CoinWarmStartVectorPairDiff< T, U >:



Public Member Functions

- virtual `CoinWarmStartDiff * clone () const`
'Virtual constructor'

8.99.1 Detailed Description

`template<typename T, typename U> class CoinWarmStartVectorPairDiff< T, U >`

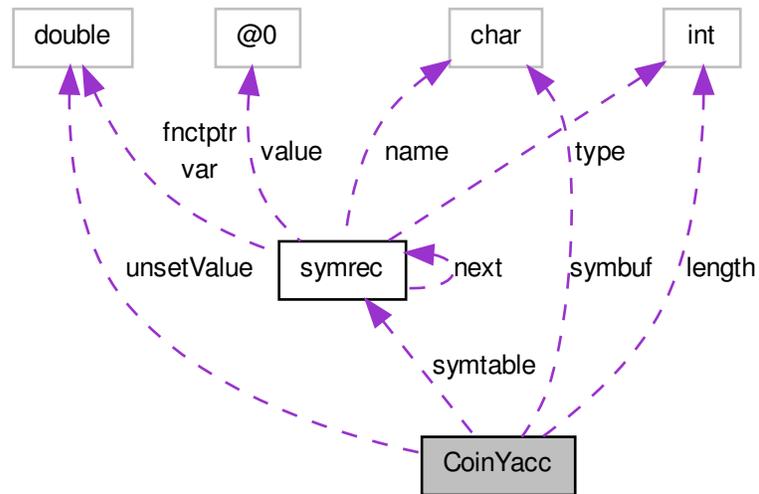
Definition at line 283 of file CoinWarmStartVector.hpp.

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

- CoinWarmStartVector.hpp

8.100 CoinYacc Class Reference

Collaboration diagram for CoinYacc:



8.100.1 Detailed Description

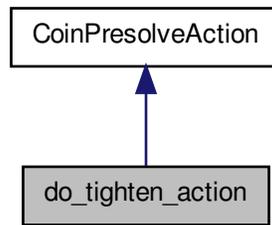
Definition at line 151 of file `CoinModelUseful.hpp`.

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

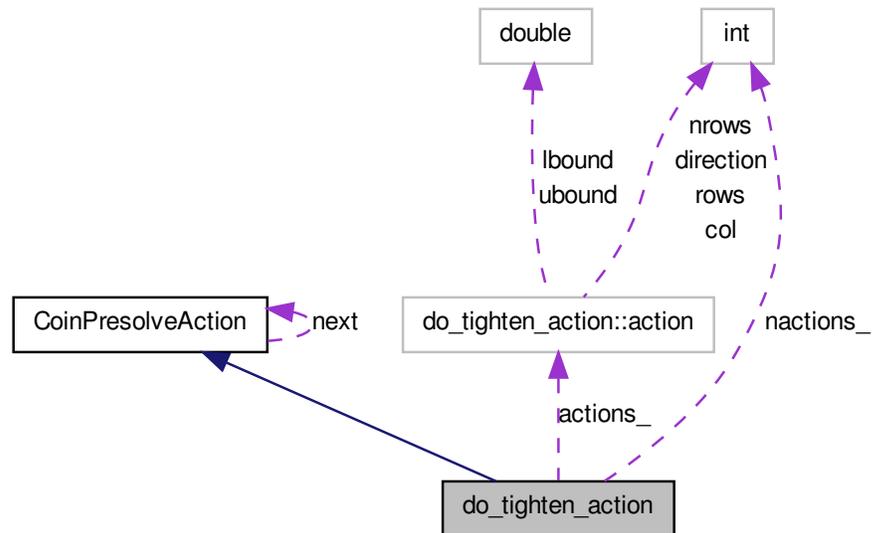
- `CoinModelUseful.hpp`

8.101 do_tighten_action Class Reference

Inheritance diagram for do_tighten_action:



Collaboration diagram for do_tighten_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.

8.101.1 Detailed Description

Definition at line 19 of file CoinPresolveTighten.hpp.

8.101.2 Member Function Documentation

8.101.2.1 `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](#).

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

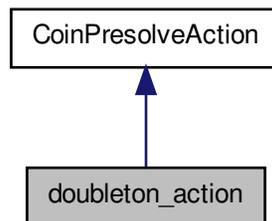
- CoinPresolveTighten.hpp

8.102 doubleton_action Class Reference

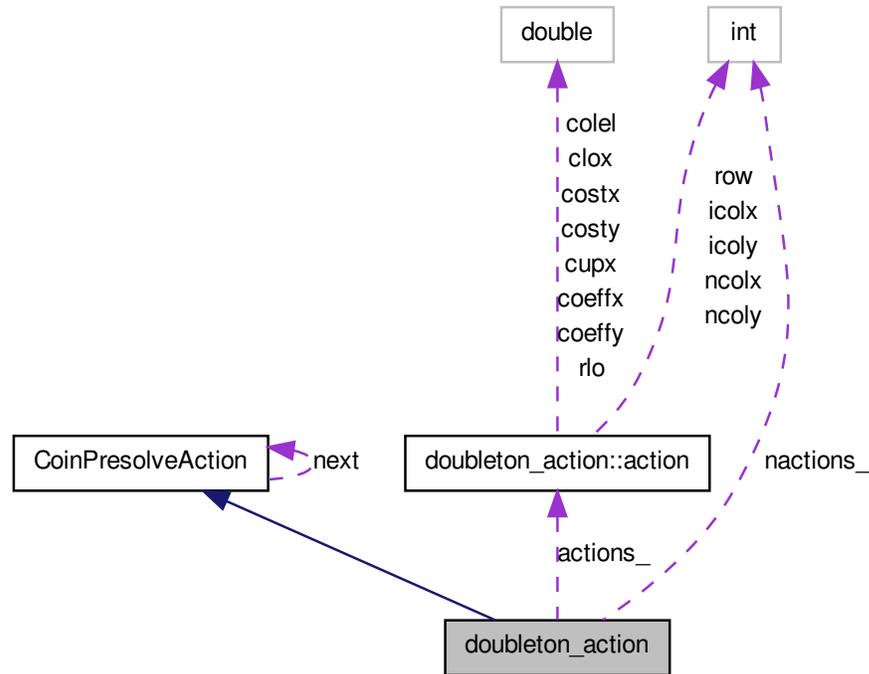
Solve $ax+by=c$ for y and substitute y out of the problem.

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

Inheritance diagram for doubleton_action:



Collaboration 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.

8.102.1 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`.

8.102.2 Member Function Documentation

8.102.2.1 `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`.

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

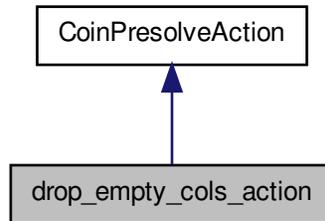
- `CoinPresolveDoubleton.hpp`

8.103 `drop_empty_cols_action` Class Reference

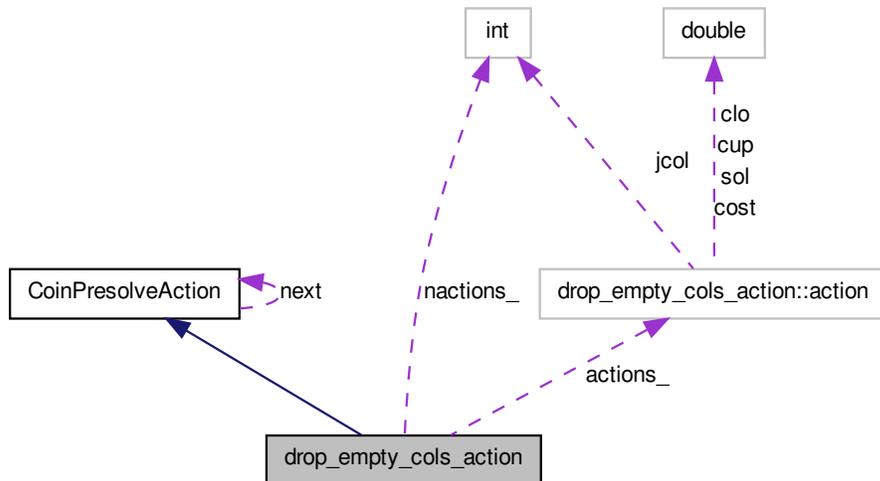
Physically removes empty columns in presolve, and reinserts empty columns in post-solve.

```
#include <CoinPresolveEmpty.hpp>
```

Inheritance diagram for drop_empty_cols_action:



Collaboration diagram for drop_empty_cols_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.

8.103.1 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_`.

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

8.103.2 Member Function Documentation

8.103.2.1 `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`.

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

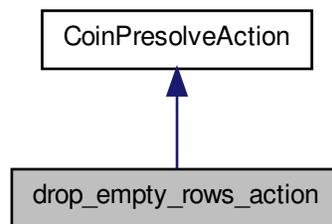
- `CoinPresolveEmpty.hpp`

8.104 drop_empty_rows_action Class Reference

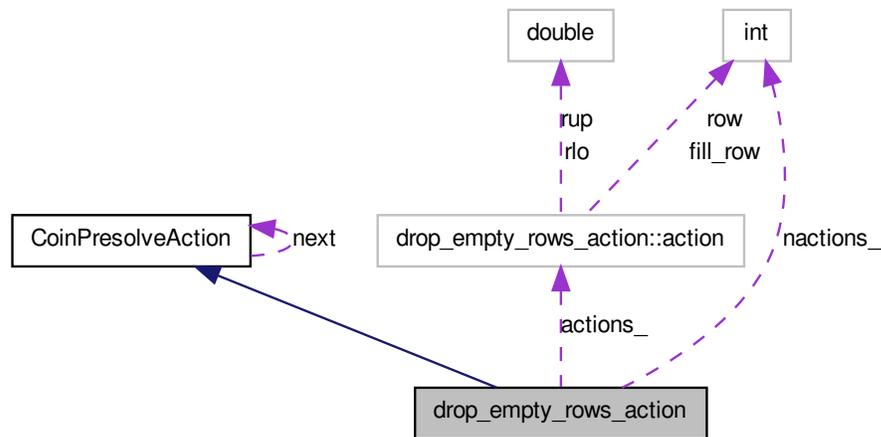
Physically removes empty rows in presolve, and reinserts empty rows in postsolve.

```
#include <CoinPresolveEmpty.hpp>
```

Inheritance diagram for drop_empty_rows_action:



Collaboration diagram for drop_empty_rows_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.

8.104.1 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.

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

8.104.2 Member Function Documentation

8.104.2.1 `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`.

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

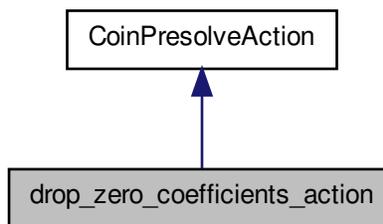
- `CoinPresolveEmpty.hpp`

8.105 `drop_zero_coefficients_action` Class Reference

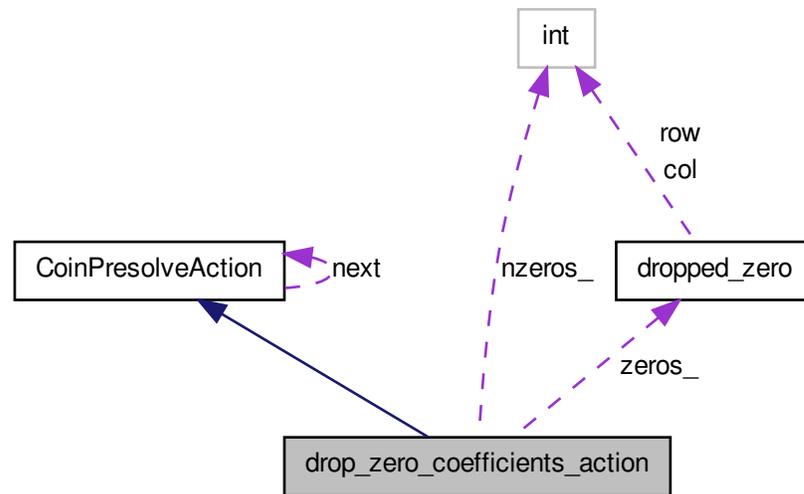
Removal of explicit zeros.

```
#include <CoinPresolveZeros.hpp>
```

Inheritance diagram for `drop_zero_coefficients_action`:



Collaboration 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.

8.105.1 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`.

8.105.2 Member Function Documentation

8.105.2.1 `const char* dropped_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`.

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

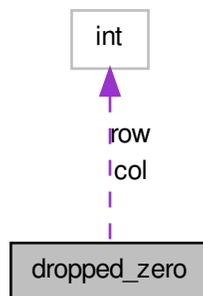
- [CoinPresolveZeros.hpp](#)

8.106 `dropped_zero` Struct Reference

Tracking information for an explicit zero coefficient.

```
#include <CoinPresolveZeros.hpp>
```

Collaboration diagram for `dropped_zero`:



8.106.1 Detailed Description

Tracking information for an explicit zero coefficient.

Definition at line 22 of file CoinPresolveZeros.hpp.

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

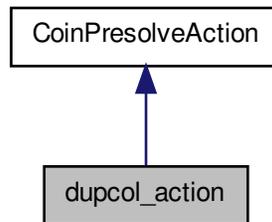
- [CoinPresolveZeros.hpp](#)

8.107 dupcol_action Class Reference

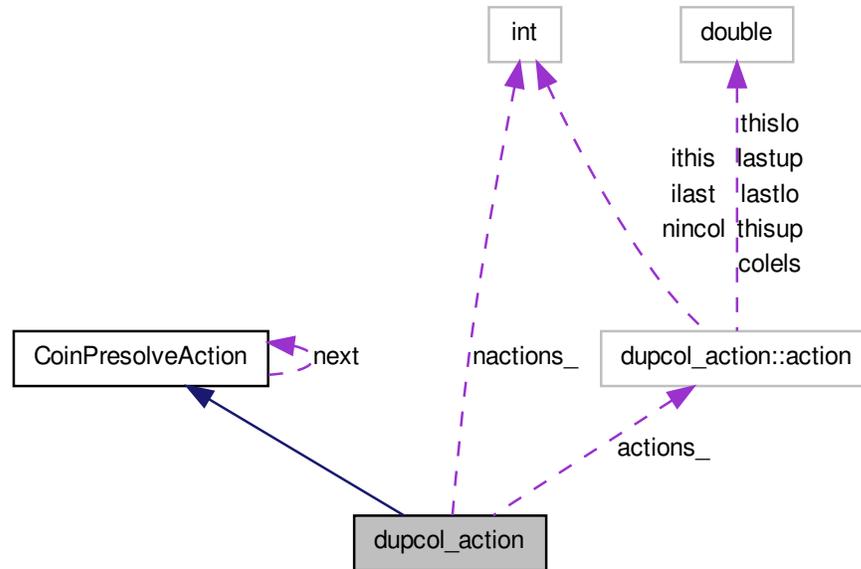
Detect and remove duplicate columns.

```
#include <CoinPresolveDupcol.hpp>
```

Inheritance diagram for dupcol_action:



Collaboration diagram for dupcol_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.

8.107.1 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.

8.107.2 Member Function Documentation

8.107.2.1 `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](#).

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

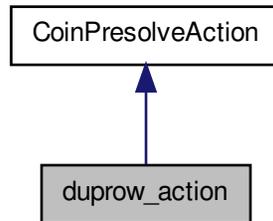
- [CoinPresolveDupcol.hpp](#)

8.108 duprow_action Class Reference

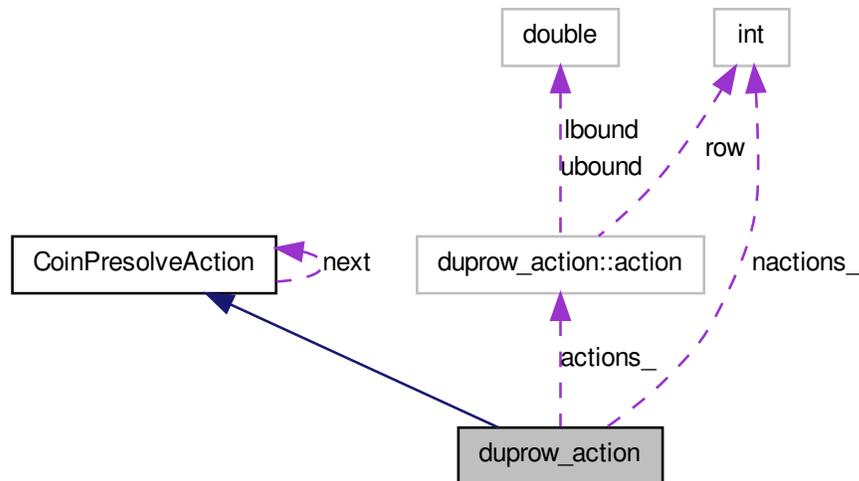
Detect and remove duplicate rows.

```
#include <CoinPresolveDupcol.hpp>
```

Inheritance diagram for duprow_action:



Collaboration diagram for duprow_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.

8.108.1 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](#).

8.108.2 Member Function Documentation

8.108.2.1 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](#).

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

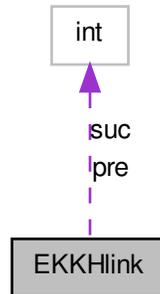
- [CoinPresolveDupcol.hpp](#)

8.109 EKKHlink Struct Reference

This deals with Factorization and Updates This is ripped off from OSL!!!!!!!!!!

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

Collaboration diagram for EKKHlink:



8.109.1 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.

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

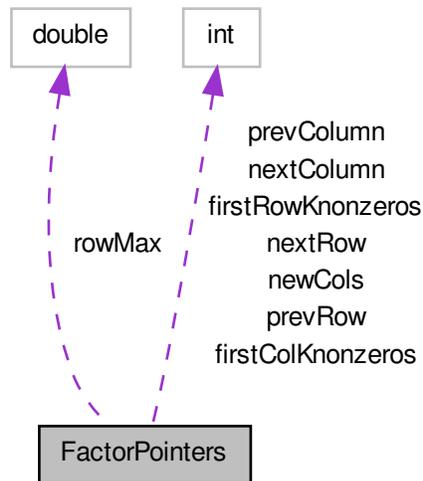
- CoinOslFactorization.hpp

8.110 FactorPointers Class Reference

pointers used during factorization

```
#include <CoinSimpFactorization.hpp>
```

Collaboration diagram for FactorPointers:



8.110.1 Detailed Description

pointers used during factorization

Definition at line 22 of file `CoinSimpFactorization.hpp`.

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

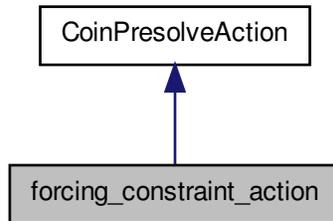
- `CoinSimpFactorization.hpp`

8.111 forcing_constraint_action Class Reference

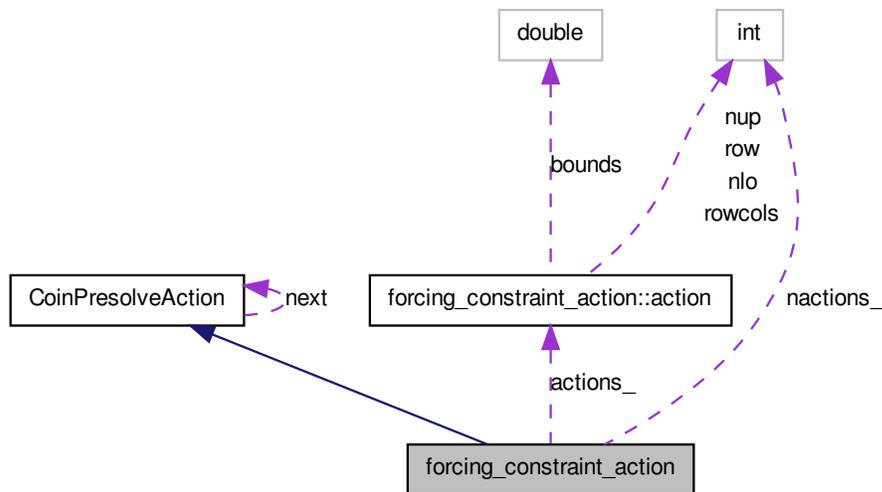
Detect and process forcing constraints and useless constraints.

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

Inheritance diagram for forcing_constraint_action:



Collaboration diagram for forcing_constraint_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.

8.111.1 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`.

8.111.2 Member Function Documentation

8.111.2.1 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](#).

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

- [CoinPresolveForcing.hpp](#)

8.112 gubrow_action Class Reference

Detect and remove entries whose sum is known.

```
#include <CoinPresolveDupcol.hpp>
```


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.

8.112.1 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`.

8.112.2 Member Function Documentation

8.112.2.1 `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`.

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

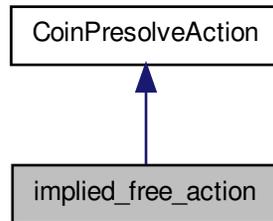
- `CoinPresolveDupcol.hpp`

8.113 `implied_free_action` Class Reference

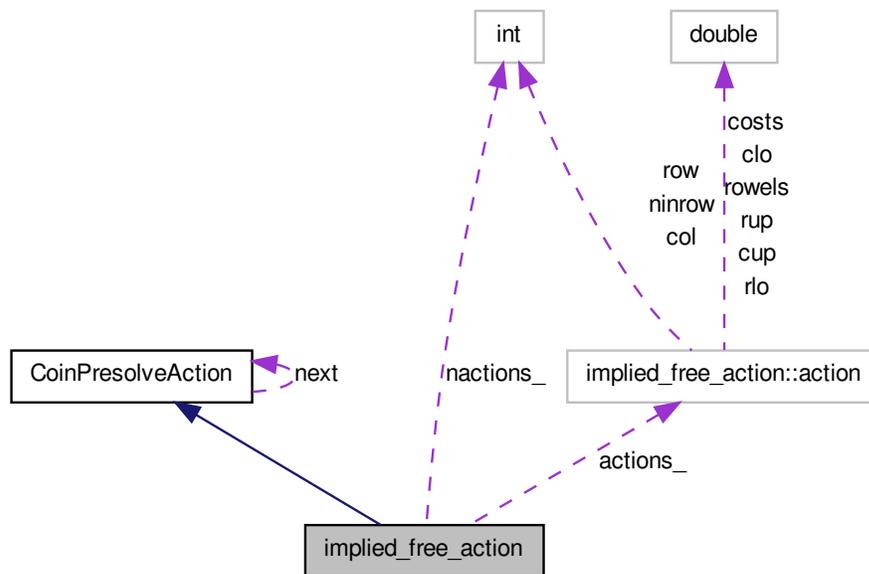
Detect and process implied free variables.

```
#include <CoinPresolveImpliedFree.hpp>
```

Inheritance diagram for implied_free_action:



Collaboration diagram for implied_free_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.

8.113.1 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`.

8.113.2 Member Function Documentation

8.113.2.1 `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.

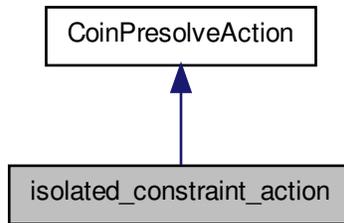
Implements [CoinPresolveAction](#).

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

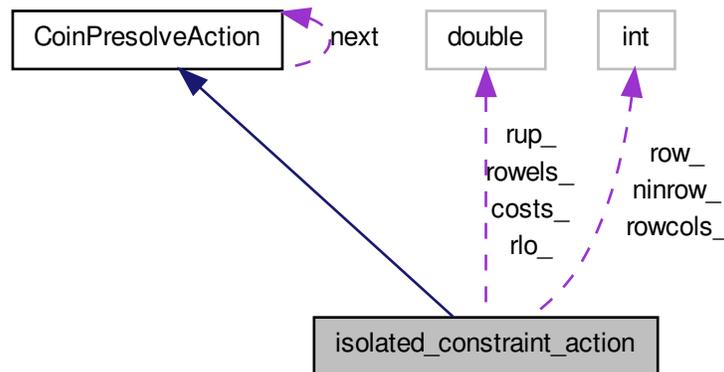
- [CoinPresolveImpliedFree.hpp](#)

8.114 isolated_constraint_action Class Reference

Inheritance diagram for isolated_constraint_action:



Collaboration 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.

8.114.1 Detailed Description

Definition at line 11 of file `CoinPresolveIsolated.hpp`.

8.114.2 Member Function Documentation

8.114.2.1 `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`.

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

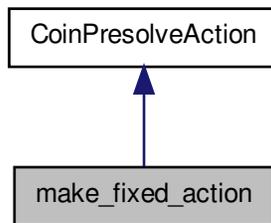
- `CoinPresolveIsolated.hpp`

8.115 `make_fixed_action` Class Reference

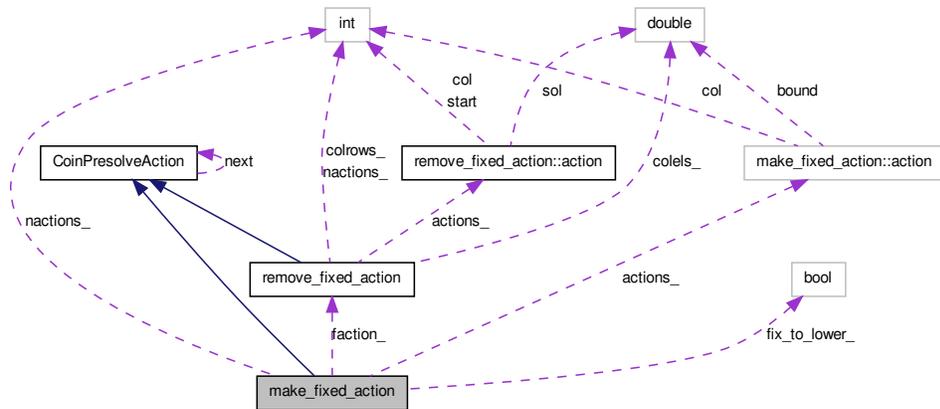
Fix a variable at a specified bound.

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

Inheritance diagram for make_fixed_action:



Collaboration diagram for make_fixed_action:



Classes

- struct **action**

Structure to preserve the bound overwritten when fixing a variable.

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 nfcols, 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.

8.115.1 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`.

8.115.2 Member Function Documentation

8.115.2.1 `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.

8.115.2.2 `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](#).

8.115.3 Friends And Related Function Documentation

8.115.3.1 `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.

8.115.3.2 `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:

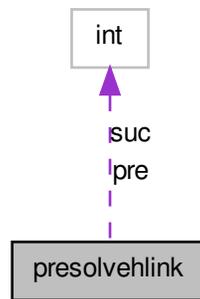
- CoinPresolveFixed.hpp

8.116 presolvehlink Class Reference

Links to aid in packed matrix modification.

```
#include <CoinPresolveMatrix.hpp>
```

Collaboration diagram for presolvehlink:



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

8.116.1 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 `mstrt_` 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`.

8.116.2 Friends And Related Function Documentation

8.116.2.1 `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`.

8.116.2.2 `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`.

8.116.23 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.

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

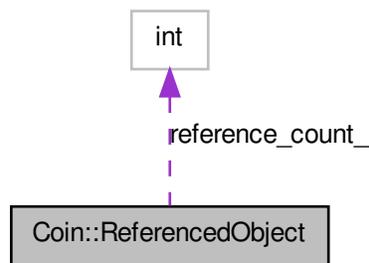
- [CoinPresolveMatrix.hpp](#)

8.117 Coin::ReferencedObject Class Reference

[ReferencedObject](#) class.

```
#include <CoinSmartPtr.hpp>
```

Collaboration diagram for Coin::ReferencedObject:



8.117.1 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(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 157 of file `CoinSmartPtr.hpp`.

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

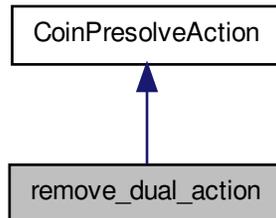
- `CoinSmartPtr.hpp`

8.118 `remove_dual_action` Class Reference

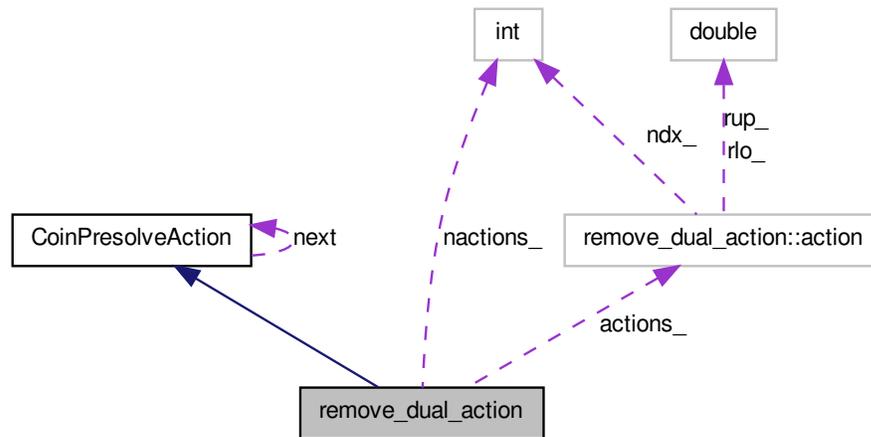
Attempt to fix variables by bounding reduced costs.

```
#include <CoinPresolveDual.hpp>
```

Inheritance diagram for `remove_dual_action`:



Collaboration diagram for remove_dual_action:



Classes

- struct **action**
Postsolve (bound restore) instruction.

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.

8.118.1 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`.

8.118.2 Member Function Documentation

- 8.118.2.1** `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.

- 8.118.2.2** `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:

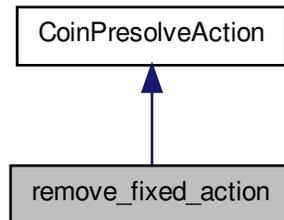
- `CoinPresolveDual.hpp`

8.119 `remove_fixed_action` Class Reference

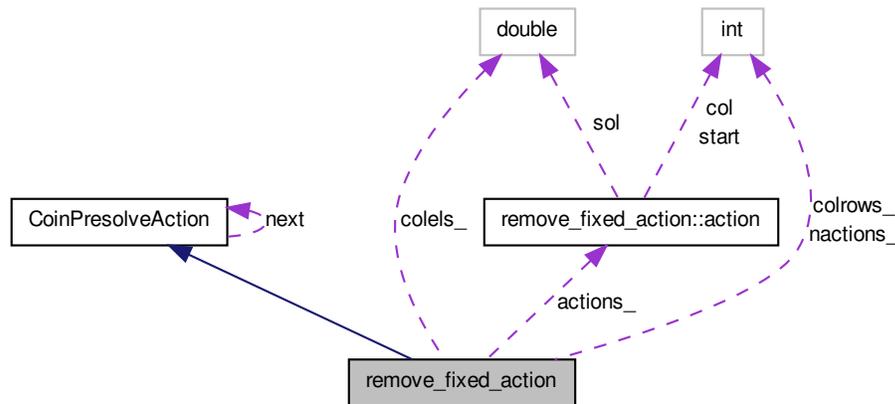
Excise fixed variables from the model.

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

Inheritance diagram for `remove_fixed_action`:



Collaboration 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 `nfcols`, const `CoinPresolveAction` *`next`)
Excise the specified columns.

Public Attributes

- int * `colrows_`
Array of row indices for coefficients of excised columns.
- double * `colels_`
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.

8.119.1 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 $-a_{ij} * sol_j$ 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`.

8.119.2 Member Function Documentation

8.119.2.1 `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`).

8.119.3 Friends And Related Function Documentation

8.119.3.1 `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.

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

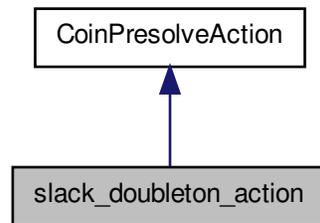
- `CoinPresolveFixed.hpp`

8.120 `slack_doubleton_action` Class Reference

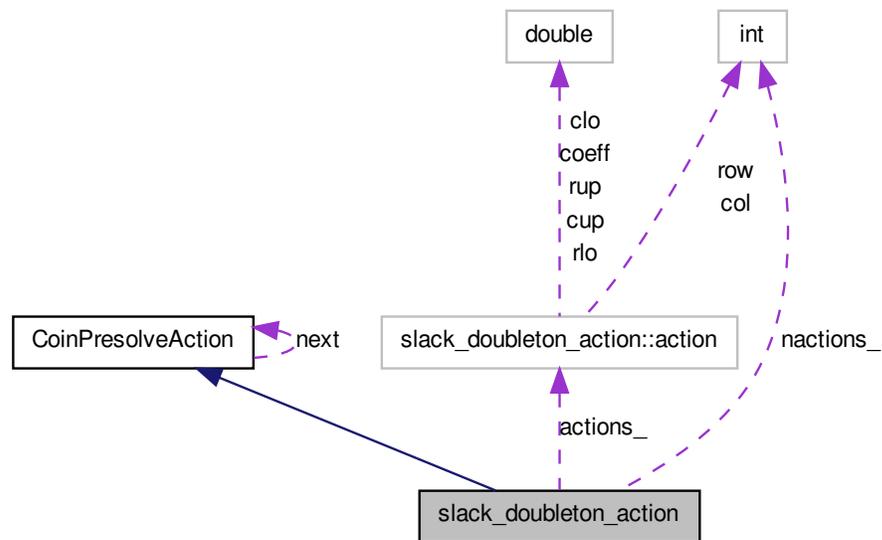
Convert an explicit bound constraint to a column bound.

```
#include <CoinPresolveSingleton.hpp>
```

Inheritance diagram for slack_doubleton_action:



Collaboration diagram for slack_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.

Static Public Member Functions

- static const [CoinPresolveAction](#) * [presolve](#) ([CoinPresolveMatrix](#) *prob, const [CoinPresolveAction](#) *next, bool ¬Finished)
Convert explicit bound constraints to column bounds.

8.120.1 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](#).

8.120.2 Member Function Documentation

8.120.2.1 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](#).

8.120.2.2 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.

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

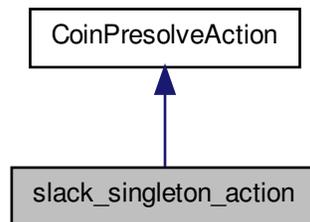
- [CoinPresolveSingleton.hpp](#)

8.121 slack_singleton_action Class Reference

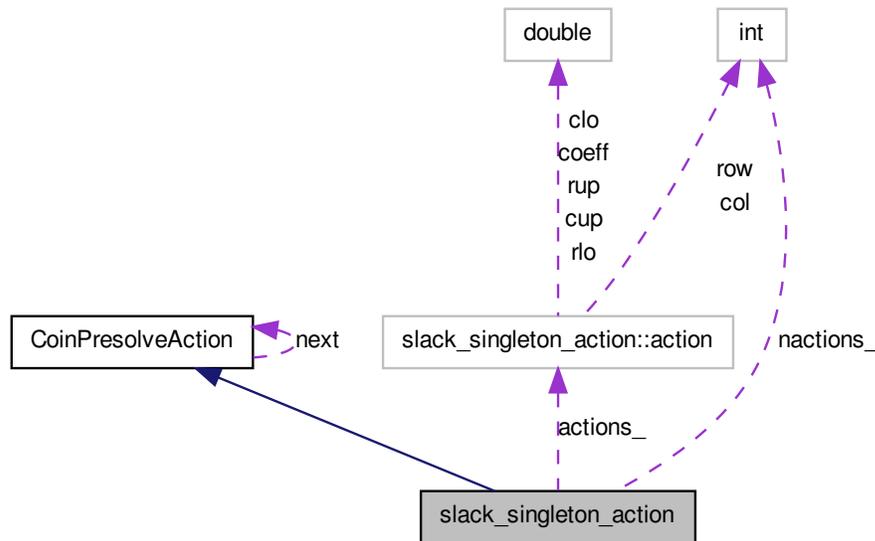
For variables with one entry.

```
#include <CoinPresolveSingleton.hpp>
```

Inheritance diagram for slack_singleton_action:



Collaboration diagram for slack_singleton_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.

8.121.1 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.

8.121.2 Member Function Documentation

8.121.2.1 `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.

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

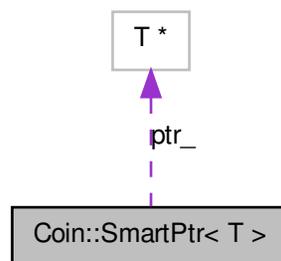
- [CoinPresolveSingleton.hpp](#)

8.122 Coin::SmartPtr< T > Class Template Reference

Template class for Smart Pointers.

```
#include <CoinSmartPtr.hpp>
```

Collaboration diagram for Coin::SmartPtr< T >:



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 [SmartPtr](#)s.
- `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 [SmartPtr](#)s.
- `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.

8.122.1 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 SmartPtrs 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 SmartPtrs 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 319 of file CoinSmartPtr.hpp.

8.122.2 Constructor & Destructor Documentation

8.122.2.1 `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.

8.122.3 Member Function Documentation

8.122.3.1 `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.

8.122.3.2 `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.

8.122.3.3 `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.

8.122.3.4 `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.

8.122.3.5 `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.

8.122.4 Friends And Related Function Documentation

8.122.4.1 `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.

8.122.4.2 `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.

8.122.4.3 `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.

8.122.4.4 `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:

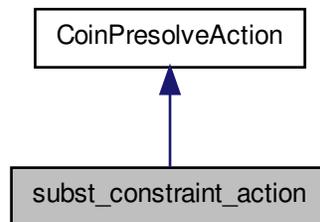
- CoinSmartPtr.hpp

8.123 subst_constraint_action Class Reference

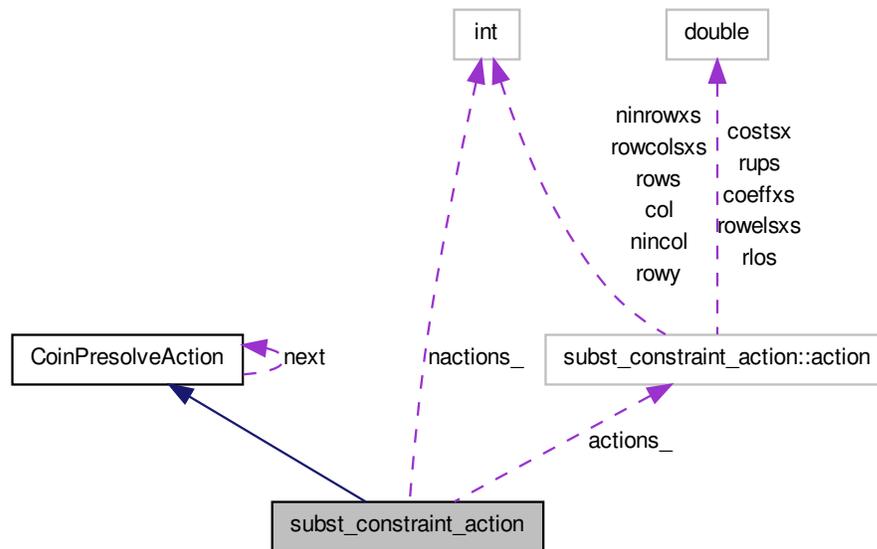
Detect and process implied free variables.

```
#include <CoinPresolveSubst.hpp>
```

Inheritance diagram for subst_constraint_action:



Collaboration diagram for subst_constraint_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.

8.123.1 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`.

8.123.2 Member Function Documentation

8.123.2.1 `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](#).

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

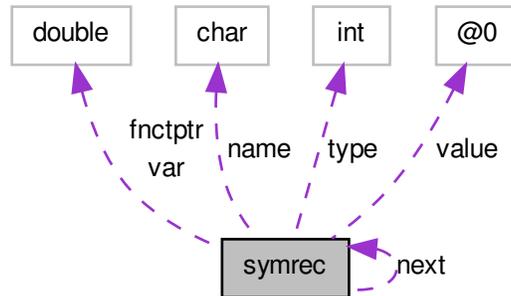
- [CoinPresolveSubst.hpp](#)

8.124 symrec Struct Reference

For string evaluation.

```
#include <CoinModelUseful.hpp>
```

Collaboration diagram for symrec:



8.124.1 Detailed Description

For string evaluation.

Definition at line 137 of file `CoinModelUseful.hpp`.

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

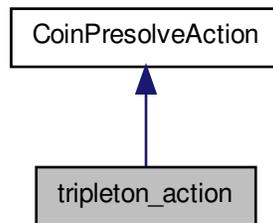
- `CoinModelUseful.hpp`

8.125 tripleton_action Class Reference

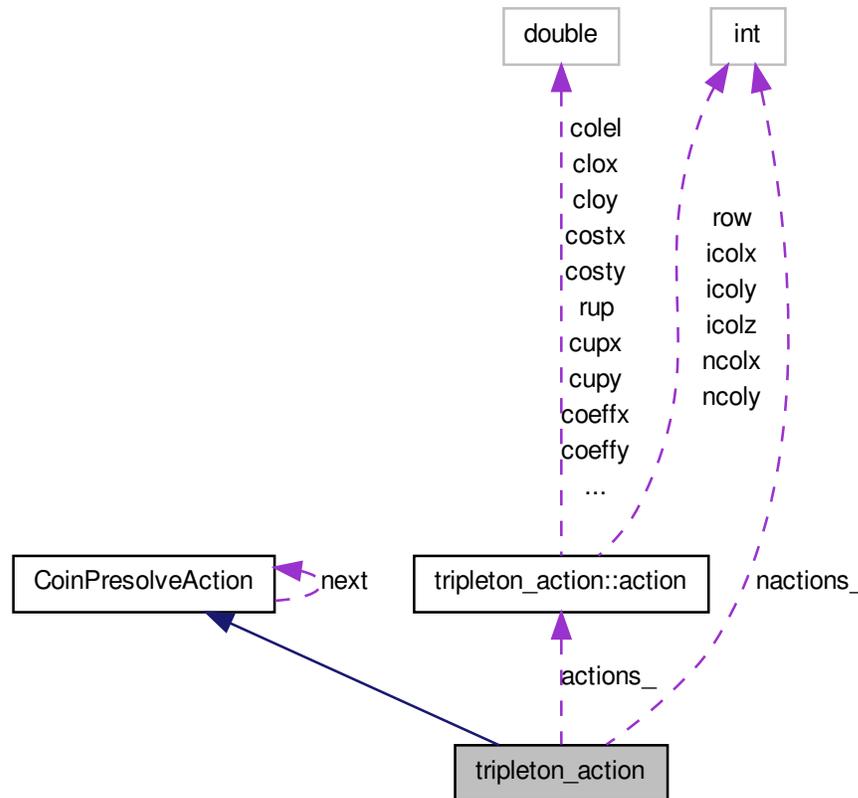
We are only going to do this if it does not increase number of elements?.

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

Inheritance diagram for tripleton_action:



Collaboration 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.

8.125.1 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`.

8.125.2 Member Function Documentation

8.125.2.1 `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`.

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

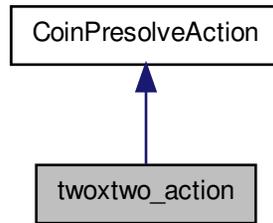
- `CoinPresolveTripletion.hpp`

8.126 `twoxtwo_action` Class Reference

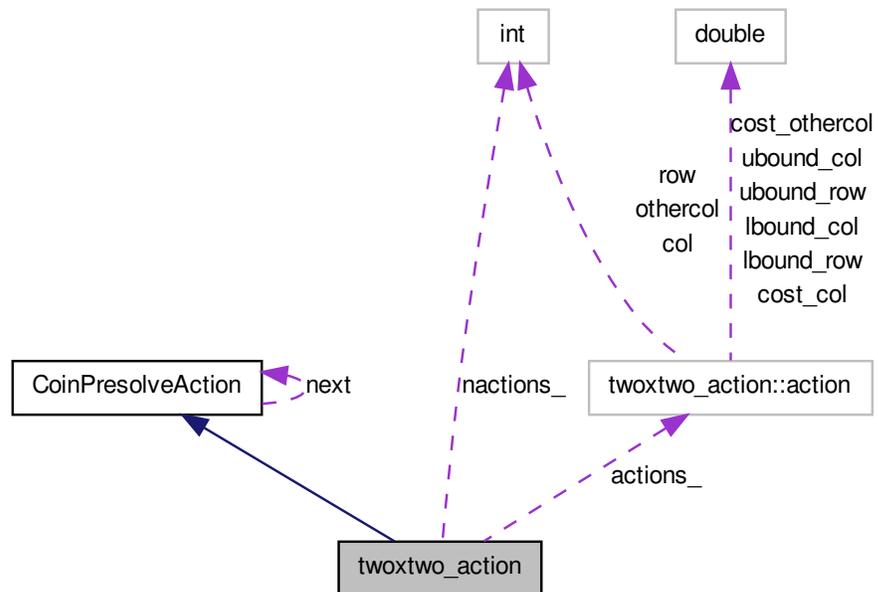
Detect interesting 2 by 2 blocks.

```
#include <CoinPresolveDupcol.hpp>
```

Inheritance diagram for twoxtwo_action:



Collaboration diagram for twoxtwo_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.

8.126.1 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`.

8.126.2 Member Function Documentation

8.126.2.1 `const char* twoxtwo_action::name () const` **[virtual]**

A name for debug printing.

It is expected that the name is not stored in the transform itself.

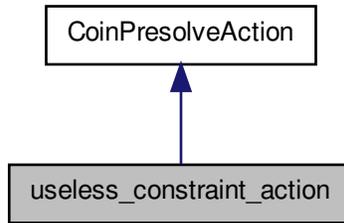
Implements `CoinPresolveAction`.

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

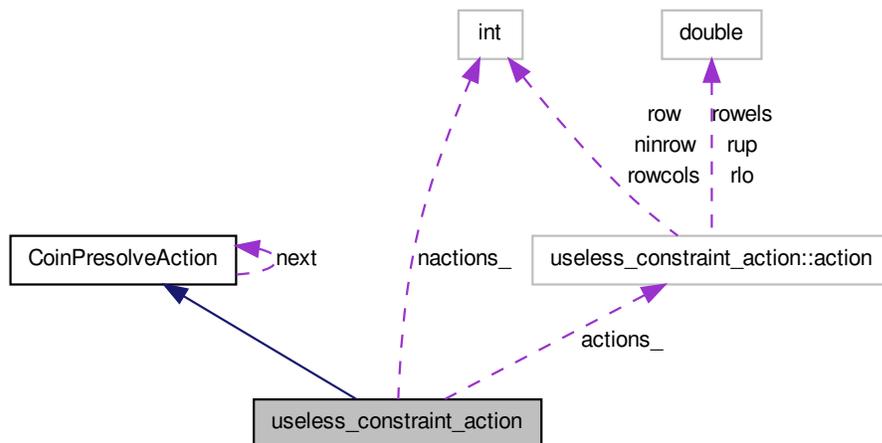
- [CoinPresolveDupcol.hpp](#)

8.127 useless_constraint_action Class Reference

Inheritance diagram for useless_constraint_action:



Collaboration diagram for useless_constraint_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.

Related Functions

(Note that these are not member functions.)

- `const CoinPresolveAction * testRedundant (CoinPresolveMatrix *prob, const CoinPresolveAction *next)`
Scan constraints looking for useless constraints.

8.127.1 Detailed Description

Definition at line 10 of file `CoinPresolveUseless.hpp`.

8.127.2 Member Function Documentation

8.127.2.1 `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](#).

8.127.3 Friends And Related Function Documentation

8.127.3.1 `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:

- `CoinPresolveUseless.hpp`

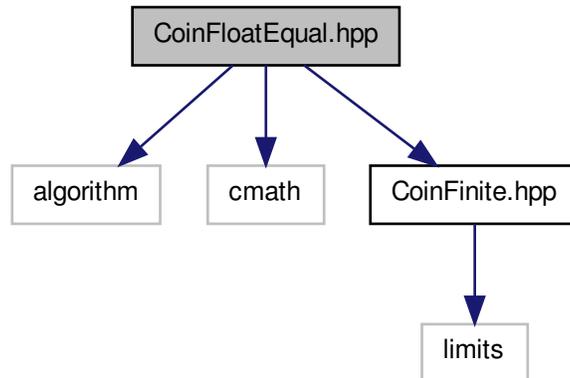
9 File Documentation

9.1 `CoinFloatEqual.hpp` File Reference

Function objects for testing equality of real numbers.

```
#include <algorithm>
#include <cmath>
#include "CoinFinite.hpp"
```

Include dependency graph for CoinFloatEqual.hpp:



Classes

- class [CoinAbsFltEq](#)
Equality to an absolute tolerance.
- class [CoinRelFltEq](#)
Equality to a scaled tolerance.

9.1.1 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.

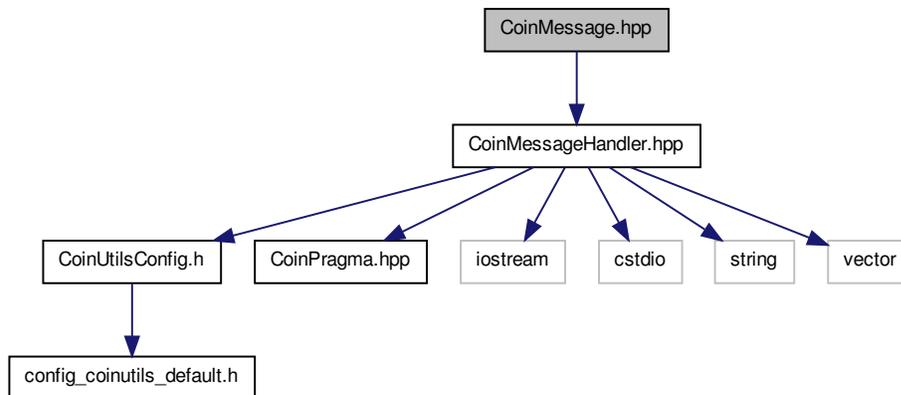
Definition in file [CoinFloatEqual.hpp](#).

9.2 CoinMessage.hpp File Reference

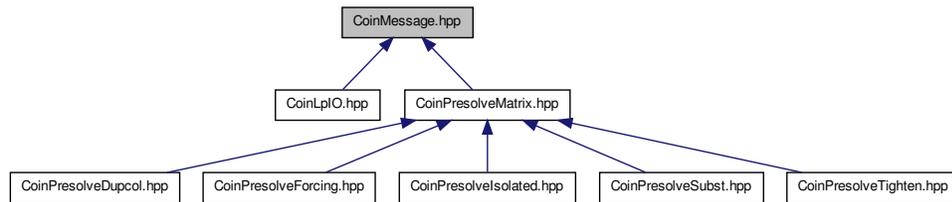
This file contains the enum for the standard set of Coin messages and a class definition whose sole purpose is to supply a constructor.

```
#include "CoinMessageHandler.hpp"
```

Include dependency graph for CoinMessage.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [CoinMessage](#)
The standard set of Coin messages.

Enumerations

- enum [COIN_Message](#)
Symbolic names for the standard set of COIN messages.

9.2.1 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](#).

9.3 CoinMessageHandler.hpp File Reference

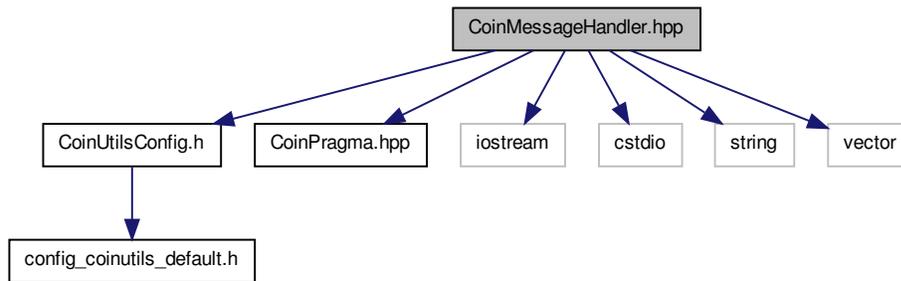
This is a first attempt at a message handler.

```

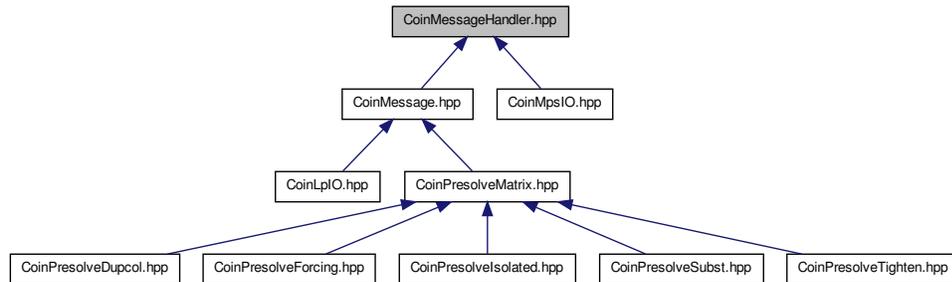
#include "CoinUtilsConfig.h"
#include "CoinPragma.hpp"
#include <iostream>
  
```

```
#include <cstdio>
#include <string>
#include <vector>
```

Include dependency graph for CoinMessageHandler.hpp:



This graph shows which files directly or indirectly include this file:



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.

Defines

- #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).

Functions

- bool [CoinMessageHandlerUnitTest](#) ()
A function that tests the methods in the [CoinMessageHandler](#) class.

9.3.1 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

Definition in file [CoinMessageHandler.hpp](#).

9.3.2 Define Documentation

9.3.2.1 #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.

9.3.3 Function Documentation

9.3.3.1 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.

9.4 CoinParam.hpp File Reference

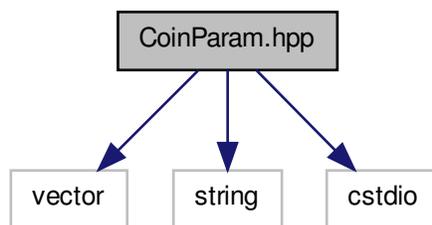
Declaration of a class for command line parameters.

```
#include <vector>
```

```
#include <string>
```

```
#include <cstdio>
```

Include dependency graph for CoinParam.hpp:



Classes

- class [CoinParam](#)

A base class for 'keyword value' command line parameters.

Namespaces

- namespace [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 &shortCnt`)
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 *shortCnt=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 numQuery)
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.

9.4.1 Detailed Description

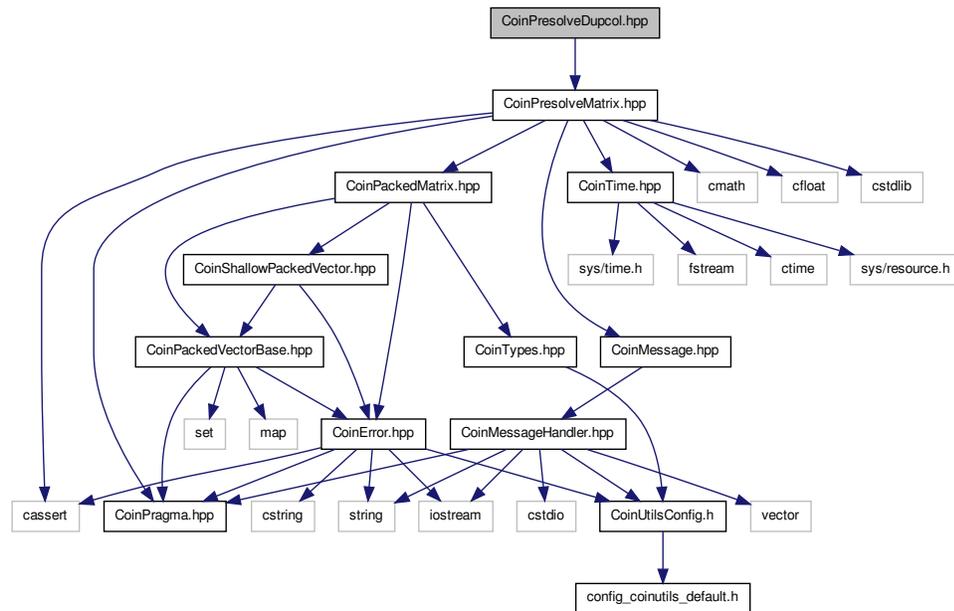
Declaration of a class for command line parameters.

Definition in file `CoinParam.hpp`.

9.5 CoinPresolveDupcol.hpp File Reference

```
#include "CoinPresolveMatrix.hpp"
```

Include dependency graph for CoinPresolveDupcol.hpp:



Classes

- class [dupcol_action](#)
Detect and remove duplicate columns.
- struct **dupcol_action::action**
- class [duprow_action](#)
Detect and remove duplicate rows.
- struct **duprow_action::action**
- class [gubrow_action](#)
Detect and remove entries whose sum is known.
- struct **gubrow_action::action**
- class [twoxtwo_action](#)
Detect interesting 2 by 2 blocks.
- struct **twoxtwo_action::action**

9.5.1 Detailed Description

Definition in file [CoinPresolveDupcol.hpp](#).

9.6 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.
- struct **drop_empty_cols_action::action**
- class [drop_empty_rows_action](#)
Physically removes empty rows in presolve, and reinserts empty rows in postsolve.
- struct **drop_empty_rows_action::action**

9.6.1 Detailed Description

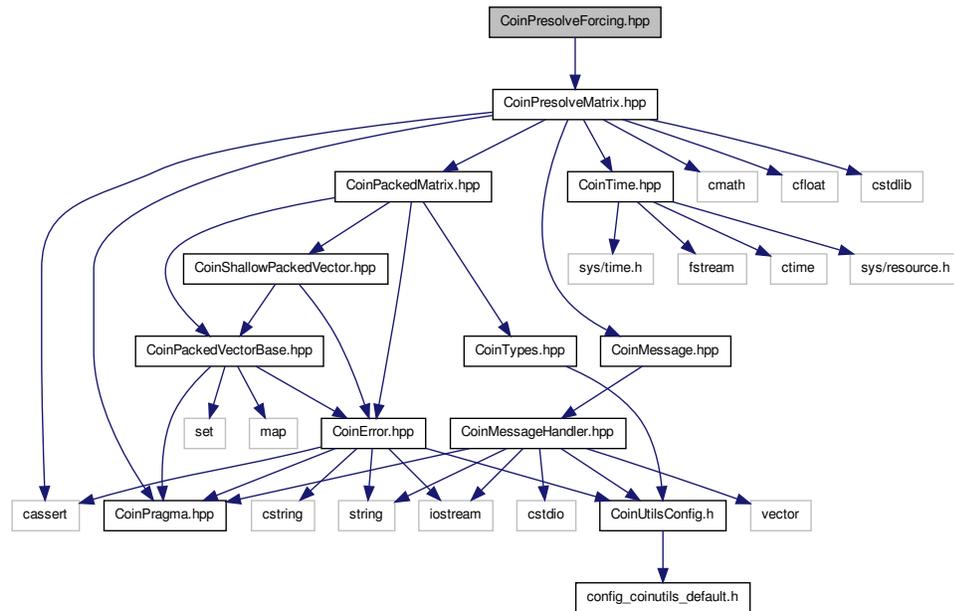
Drop/reinsert empty rows/columns.

Definition in file [CoinPresolveEmpty.hpp](#).

9.7 CoinPresolveForcing.hpp File Reference

```
#include "CoinPresolveMatrix.hpp"
```

Include dependency graph for CoinPresolveForcing.hpp:



Classes

- class [forcing_constraint_action](#)
Detect and process forcing constraints and useless constraints.
- struct [forcing_constraint_action::action](#)

9.7.1 Detailed Description

Definition in file [CoinPresolveForcing.hpp](#).

9.8 CoinPresolveImpliedFree.hpp File Reference

Classes

- class [implied_free_action](#)

Detect and process implied free variables.

- struct **implied_free_action::action**

9.8.1 Detailed Description

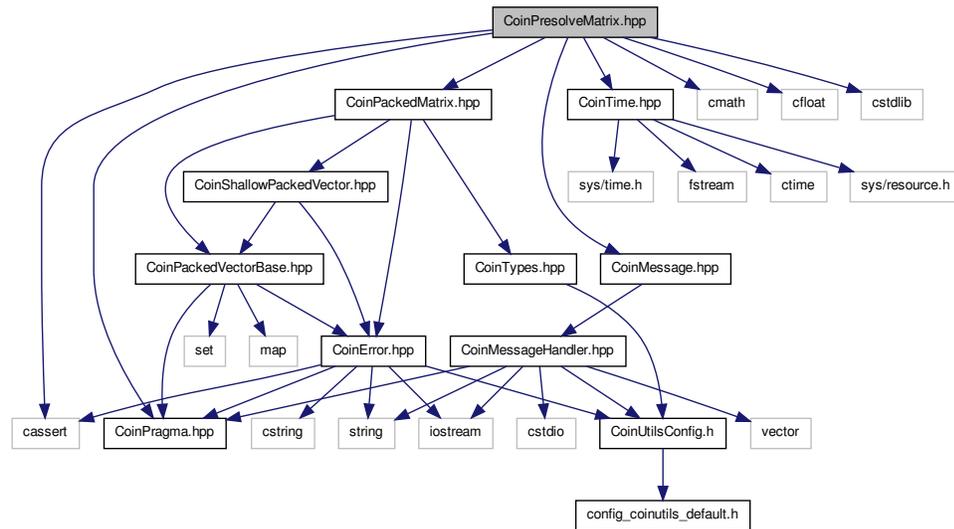
Definition in file [CoinPresolveImpliedFree.hpp](#).

9.9 CoinPresolveMatrix.hpp File Reference

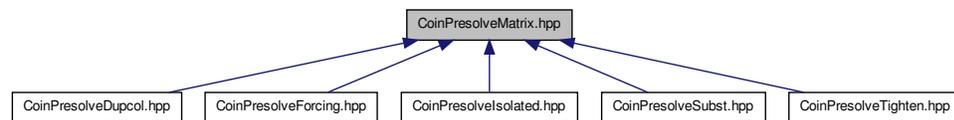
Declarations for [CoinPresolveMatrix](#) and [CoinPostsolveMatrix](#) and their common base class [CoinPrePostsolveMatrix](#).

```
#include "CoinPragma.hpp"
#include "CoinPackedMatrix.hpp"
#include "CoinMessage.hpp"
#include "CoinTime.hpp"
#include <cmath>
#include <cassert>
#include <cfloat>
#include <cstdlib>
```

Include dependency graph for CoinPresolveMatrix.hpp:



This graph shows which files directly or indirectly include this file:



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.

Defines

- #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.

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.

Variables

- const double [ZTOLDP](#) = 1e-12
Zero tolerance.
- const double [ZTOLDP2](#) = 1e-10
Alternate zero tolerance.

9.9.1 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](#).

9.9.2 Variable Documentation

9.9.2.1 `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](#).

9.9.2.2 `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](#).

9.10 CoinPresolveSingleton.hpp File Reference

Classes

- class [slack_doubleton_action](#)
Convert an explicit bound constraint to a column bound.
- struct [slack_doubleton_action::action](#)
- class [slack_singleton_action](#)
For variables with one entry.
- struct [slack_singleton_action::action](#)

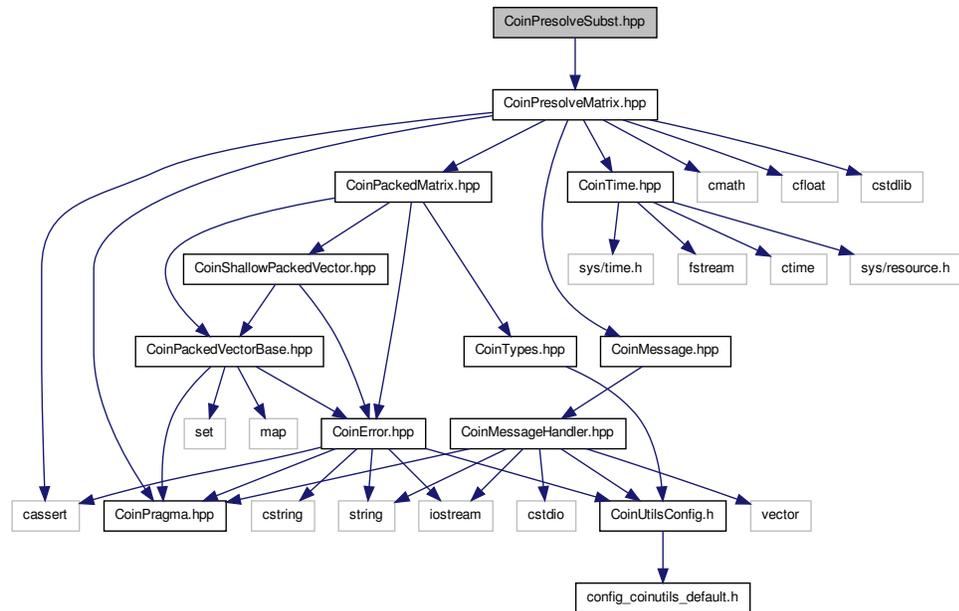
9.10.1 Detailed Description

Definition in file [CoinPresolveSingleton.hpp](#).

9.11 CoinPresolveSubst.hpp File Reference

```
#include "CoinPresolveMatrix.hpp"
```

Include dependency graph for CoinPresolveSubst.hpp:



Classes

- class [subst_constraint_action](#)
Detect and process implied free variables.
- struct [subst_constraint_action::action](#)

9.11.1 Detailed Description

Definition in file [CoinPresolveSubst.hpp](#).

9.12 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.

9.12.1 Detailed Description

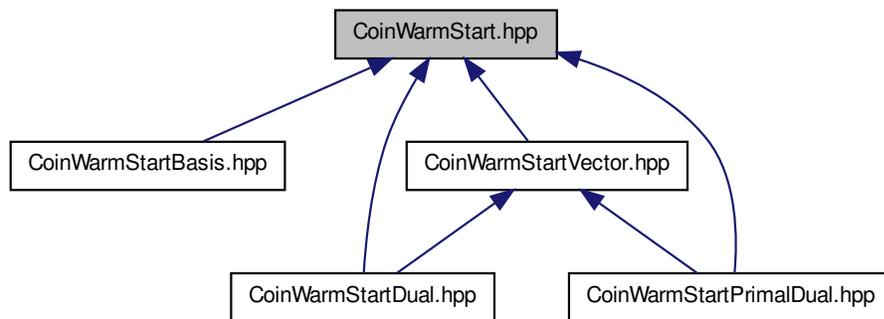
Drop/reintroduce explicit zeros.

Definition in file [CoinPresolveZeros.hpp](#).

9.13 CoinWarmStart.hpp File Reference

Copyright (C) 2000 -- 2003, International Business Machines Corporation and others.

This graph shows which files directly or indirectly include this file:



Classes

- class [CoinWarmStart](#)
Abstract base class for warm start information.
- class [CoinWarmStartDiff](#)
Abstract base class for warm start 'diff' objects.

9.13.1 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](#).

Index

- ~CoinPresolveAction
 - CoinPresolveAction, 370
- ~CoinShallowPackedVector
 - CoinShallowPackedVector, 407
- ~SmartPtr
 - Coin::SmartPtr, 541
- _EKKfactinfo, 35
- acts_
 - CoinPrePostsolveMatrix, 365
- add
 - CoinFactorization, 101
- addBlock
 - CoinStructuredModel, 436
- addColumn
 - CoinFactorization, 101
- addEasy
 - CoinModelLinkedList, 228
- addHard
 - CoinModelLinkedList, 228, 229
- addMessage
 - CoinMessages, 191
- addRow
 - CoinFactorization, 101
- anyInteger
 - CoinPresolveMatrix, 383
- anyInteger_
 - CoinPresolveMatrix, 386
- anyProhibited_
 - CoinPresolveMatrix, 388
- appendCol
 - CoinPackedMatrix, 289, 290
- appendCols
 - CoinPackedMatrix, 290
- appendMajor
 - CoinPackedMatrix, 305
- appendMajorVector
 - CoinPackedMatrix, 298, 299
- appendMinor
 - CoinPackedMatrix, 305
- appendMinorFast
 - CoinPackedMatrix, 299
- appendMinorVector
 - CoinPackedMatrix, 299
- appendRow
 - CoinPackedMatrix, 290, 291
- appendRows
 - CoinPackedMatrix, 291
- applyDiff
 - CoinWarmStartBasis, 459
 - CoinWarmStartDual, 468
 - CoinWarmStartPrimalDual, 475
 - CoinWarmStartVector, 482
- are_invalid_names
 - CoinLpIO, 162
- artificialStatus_
 - CoinWarmStartBasis, 461
- assign
 - CoinWarmStartPrimalDual, 474
- assignBasisStatus
 - CoinWarmStartBasis, 461
- assignDual
 - CoinWarmStartDual, 468
- assignMatrix
 - CoinPackedMatrix, 295
- assignPresolveToPostsolve
 - CoinPostsolveMatrix, 350
 - CoinPresolveMatrix, 385
- assignVector
 - CoinPackedVector, 315
 - CoinWarmStartVector, 481
- atLowerBound
 - CoinWarmStartBasis, 457
- atUpperBound
 - CoinWarmStartBasis, 457
- basic
 - CoinWarmStartBasis, 457
- BitVector128, 40
- borrowVector
 - CoinIndexedVector, 139
- bottomAppendPackedMatrix
 - CoinPackedMatrix, 292
- bulk0_
 - CoinPrePostsolveMatrix, 363

- card_previous_names_
 - CoinLpIO, 169
- charValue
 - CoinMessageHandler, 183
- checkAndTell
 - CoinPresolveMonitor, 391
- checkColNames
 - CoinLpIO, 168
- checkPivot
 - CoinSimpFactorization, 421
- checkRowNames
 - CoinLpIO, 168
- cleanMatrix
 - CoinPackedMatrix, 293
- clear
 - CoinWarmStartPrimalDual, 474
 - CoinWarmStartPrimalDualDiff, 479
 - CoinWarmStartVector, 482
 - CoinWarmStartVectorDiff, 485
- clone
 - CoinWarmStartPrimalDualDiff, 479
- Coin::ReferencedObject, 522
- Coin::SmartPtr, 536
 - ~SmartPtr, 541
 - GetRawPtr, 541
 - IsNull, 541
 - IsValid, 541
 - operator*, 542
 - operator->, 542
 - operator==, 542
- COIN_NUM_LOG
 - CoinMessageHandler.hpp, 559
- CoinAbsFltEq, 40
 - CoinAbsFltEq, 42
- CoinArbitraryArrayWithLength, 42
 - CoinArbitraryArrayWithLength, 45
 - operator=, 45
- CoinArrayWithLength, 45
 - CoinArrayWithLength, 50
 - operator=, 50
- CoinBaseModel, 51
 - logLevel_, 54
- CoinBigIndexArrayWithLength, 54
 - CoinBigIndexArrayWithLength, 57
 - operator=, 57
- CoinBuild, 57
 - CoinBuild, 60
- CoinDenseFactorization, 61
 - replaceColumn, 65
 - updateColumnFT, 65
- CoinDenseVector, 66
 - resize, 69
 - setVector, 69
- CoinDoubleArrayWithLength, 70
 - CoinDoubleArrayWithLength, 72
 - operator=, 73
- CoinError, 73
 - CoinErrorUnitTest, 76
- CoinErrorUnitTest
 - CoinError, 76
- CoinExternalVectorFirstGreater_2, 76
- CoinExternalVectorFirstGreater_3, 77
- CoinExternalVectorFirstLess_2, 78
- CoinExternalVectorFirstLess_3, 79
- CoinFactorization, 80
 - add, 101
 - addColumn, 101
 - addRow, 101
 - factorize, 99, 100
 - factorizePart1, 100
 - getColumnSpaceIterate, 102
 - getColumnSpaceIterateR, 102
 - lengthU_, 104
 - numberTrials_, 104
 - preProcess, 101
 - replaceColumn, 100
 - replaceColumnPFI, 103
 - restoreFactorization, 99
 - updateColumnFT, 100
 - updateColumnRFT, 102
 - updateColumnTransposeU, 102
 - updateColumnTransposeUByColumn, 103
 - updateColumnTransposeUDensish, 103
 - updateColumnTransposeUSparse, 103
 - updateColumnTransposeUSparsish, 102
 - updateTwoColumnsFT, 101
- CoinFactorizationDoubleArrayWithLength, 104

- CoinFactorizationDoubleArray-
 WithLength, 107
- operator=, 107
- CoinFactorizationLongDoubleArrayWithLength, scan, 141
 107
- CoinFactorizationLongDoubleAr-
 rayWithLength, 110
- operator=, 110
- CoinFileInput, 110
 - CoinFileInput, 113
 - create, 114
 - fileAbsPath, 115
 - fileCoinReadable, 115
 - gets, 114
 - read, 114
- CoinFileIOBase, 116
 - CoinFileIOBase, 118
- CoinFileOutput, 118
 - CoinFileOutput, 121
 - COMPRESS_BZIP2, 120
 - COMPRESS_GZIP, 120
 - COMPRESS_NONE, 120
 - Compression, 120
 - compressionSupported, 121
 - create, 121
 - puts, 122
 - write, 121
- CoinFirstAbsGreater_2, 122
- CoinFirstAbsGreater_3, 123
- CoinFirstAbsLess_2, 123
- CoinFirstAbsLess_3, 124
- CoinFirstGreater_2, 125
- CoinFirstGreater_3, 125
- CoinFirstLess_2, 126
- CoinFirstLess_3, 126
- CoinFloatEqual.hpp, 554
- CoinIndexedVector, 128
 - borrowVector, 139
 - CoinIndexedVector, 138
 - CoinIndexedVectorUnitTest, 142
 - copy, 139
 - denseVector, 139
 - getIndices, 138
 - operator=, 139
 - operator==, 141
 - quickAdd, 140
 - quickAddNonZero, 140
 - reserve, 142
 - returnVector, 139
 - setVector, 140
 - sort, 142
 - zero, 141
- CoinIndexedVectorUnitTest
 - CoinIndexedVector, 142
- CoinIntArrayWithLength, 142
 - CoinIntArrayWithLength, 145
 - operator=, 145
- CoinLpIO, 145
 - are_invalid_names, 162
 - card_previous_names_, 169
 - checkColNames, 168
 - checkRowNames, 168
 - columnIndex, 160
 - columnName, 160
 - defaultHandler_, 169
 - find_obj, 166
 - findHash, 165
 - first_is_number, 166
 - freePreviousNames, 158
 - getPreviousColNames, 159
 - getPreviousRowNames, 159
 - getRightHandSide, 158
 - getRowNames, 159
 - getRowRange, 159
 - getRowSense, 158
 - hash_, 170
 - insertHash, 165
 - integerType_, 169
 - is_comment, 166
 - is_free, 167
 - is_inf, 167
 - is_invalid_name, 161
 - is_keyword, 167
 - is_sense, 167
 - is_subject_to, 166
 - maxHash_, 170
 - names_, 169
 - numberHash_, 170
 - out_coeff, 166
 - passInMessageHandler, 165
 - previous_names_, 169

- read_monom_obj, 167
- read_monom_row, 168
- readLp, 164
- rowIndex, 160
- rowName, 160
- setDecimals, 161
- setDefaultRowNames, 162
- setEpsilon, 160
- setInfinity, 160
- setLpDataRowAndColNames, 162
- setLpDataWithoutRowAndColNames, 161
- setNumberAcross, 161
- startHash, 165
- stopHash, 165
- writeLp, 163, 164
- CoinLpIO::CoinHashLink, 128
- CoinMessage, 170
 - CoinMessage, 173
- CoinMessage.hpp, 556
- CoinMessageHandler, 173
 - charValue, 183
 - CoinMessageHandlerUnitTest, 186
 - doubleValue, 182
 - finish, 186
 - intValue, 182
 - logLevel, 181
 - message, 184
 - numberCharFields, 183
 - numberDoubleFields, 182
 - numberIntFields, 182
 - numberStringFields, 183
 - operator<<, 184, 185
 - operator=, 181
 - printing, 186
 - printStatus_, 186
 - setLogLevel, 181, 182
 - stringValue, 183
- CoinMessageHandler.hpp, 557
 - COIN_NUM_LOG, 559
 - CoinMessageHandlerUnitTest, 560
- CoinMessageHandlerUnitTest
 - CoinMessageHandler, 186
 - CoinMessageHandler.hpp, 560
- CoinMessages, 187
 - addMessage, 191
 - CoinMessages, 190
 - Language, 190
 - language, 191
 - lengthMessages_, 192
 - operator=, 191
 - replaceMessage, 191
 - setDetailMessages, 191
- CoinModel, 192
 - CoinModel, 207
 - convertMatrix, 210
 - countPlusMinusOne, 212
 - deleteCol, 208
 - deleteColumn, 208
 - deleteRow, 208
 - differentModel, 210
 - expandKnapsack, 215
 - firstInColumn, 211
 - firstInQuadraticColumn, 212
 - firstInRow, 211
 - getColumn, 208
 - getElementAsString, 210
 - getRow, 208
 - lastInColumn, 211
 - lastInQuadraticColumn, 212
 - lastInRow, 211
 - loadBlock, 213, 214
 - next, 212
 - pack, 209
 - packCols, 209
 - packColumns, 209
 - packRows, 208
 - pointer, 210
 - position, 211
 - previous, 212
 - reorder, 214
 - writeMps, 209
- CoinModelHash, 216
 - CoinModelHash, 218
- CoinModelHash2, 218
 - CoinModelHash2, 220
- CoinModelHashLink, 221
- CoinModelInfo2, 222
- CoinModelLink, 222
 - CoinModelLink, 224, 225
- CoinModelLinkedList, 225
 - addEasy, 228

- addHard, 228, 229
- CoinModelLinkedList, 228
- create, 228
- deleteSame, 229
- updateDeleted, 229
- CoinModelTriple, 229
- CoinMpsCardReader, 230
 - mpsType, 235
 - nextGmsField, 235
- CoinMpsIO, 235
 - CoinMpsIOUnitTest, 254
 - columnIndex, 249
 - columnName, 249
 - copyInIntegerInformation, 250
 - dealWithFileName, 253
 - defaultHandler_, 254
 - getRightHandSide, 248
 - getRowRange, 248
 - getRowSense, 247
 - integerColumns, 248
 - isInteger, 248
 - objectiveOffset, 249
 - passInMessageHandler, 253
 - readBasis, 250
 - readConicMps, 252
 - readGms, 251
 - readMps, 250
 - readQuadraticMps, 252
 - releaseRedundantInformation, 253
 - rowIndex, 249
 - rowName, 249
 - writeMps, 251
- CoinMpsIO::CoinHashLink, 127
- CoinMpsIOUnitTest
 - CoinMpsIO, 254
- CoinOneMessage, 254
 - CoinOneMessage, 257
 - operator=, 257
 - setExternalNumber, 257
- CoinOslFactorization, 258
 - factorize, 263
 - indices, 264
 - replaceColumn, 263
 - updateColumnFT, 264
- CoinOtherFactorization, 264
 - replaceColumn, 271
 - setSolveMode, 271
 - solveMode, 271
 - solveMode_, 272
 - updateColumnFT, 271
- CoinPackedMatrix, 272
 - appendCol, 289, 290
 - appendCols, 290
 - appendMajor, 305
 - appendMajorVector, 298, 299
 - appendMinor, 305
 - appendMinorFast, 299
 - appendMinorVector, 299
 - appendRow, 290, 291
 - appendRows, 291
 - assignMatrix, 295
 - bottomAppendPackedMatrix, 292
 - cleanMatrix, 293
 - CoinPackedMatrix, 284, 285
 - CoinPackedMatrixUnitTest, 305
 - colOrdered_, 306
 - compress, 293
 - copyOf, 294
 - copyReuseArrays, 294
 - countOrthoLength, 297
 - deleteCols, 292
 - deleteMajorVectors, 300
 - deleteMinorVectors, 301
 - deleteRows, 292
 - dumpMatrix, 298
 - element_, 306
 - eliminateDuplicates, 293
 - extraGap_, 306
 - extraMajor_, 306
 - getCoefficient, 293
 - getElements, 287
 - getExtraGap, 285
 - getExtraMajor, 285
 - getIndices, 287
 - getMajorDim, 297
 - getMajorIndices, 289
 - getMaxMajorDim, 298
 - getMinorDim, 298
 - getMutableElements, 302
 - getMutableIndices, 303
 - getMutableVectorLengths, 303
 - getMutableVectorStarts, 303

- getNumCols, 286
- getNumElements, 286
- getNumRows, 287
- getSizeVectorLengths, 287
- getSizeVectorStarts, 287
- getVector, 288
- getVectorLast, 288
- getVectorLengths, 288
- getVectorSize, 288
- getVectorStarts, 288
- hasGaps, 286
- index_, 306
- isColOrdered, 286
- isEquivalent, 302
- isEquivalent2, 302
- length_, 307
- majorAppendOrthoOrdered, 300
- majorAppendSameOrdered, 299
- minorAppendOrthoOrdered, 300
- minorAppendSameOrdered, 300
- modifyCoefficient, 292
- nullElementArray, 303
- nullIndexArray, 304
- nullLengthArray, 304
- nullStartArray, 303
- operator=, 295
- replaceVector, 292
- reserve, 286
- reverseOrderedCopyOf, 295
- reverseOrdering, 295
- rightAppendPackedMatrix, 291
- setDimensions, 289
- setExtraGap, 289
- setExtraMajor, 289
- setMajorDim, 297
- setMinorDim, 298
- start_, 307
- submatrixOf, 293
- submatrixOfWithDuplicates, 294
- swap, 296
- times, 296
- timesMajor, 301
- timesMinor, 301
- transpose, 296
- transposeTimes, 296, 297
- verifyMtx, 304
- CoinPackedMatrixUnitTest
 - CoinPackedMatrix, 305
- CoinPackedVector, 307
 - assignVector, 315
 - CoinPackedVector, 314
 - CoinPackedVectorUnitTest, 317
 - getOriginalPosition, 315
 - operator=, 315
 - reserve, 316
 - setVector, 316
 - sort, 316
 - sortOriginalOrder, 316
 - truncate, 316
- CoinPackedVectorBase, 317
 - CoinPackedVectorBase, 321
 - compare, 323
 - denseVector, 322
 - findIndex, 323
 - isEquivalent, 323
 - isExistingIndex, 323
 - operator==, 323
 - setTestForDuplicateIndex, 322
 - setTestForDuplicateIndexWhen-
True, 322
 - testForDuplicateIndex, 322
- CoinPackedVectorUnitTest
 - CoinPackedVector, 317
- CoinPair, 324
- CoinParam, 325
 - CoinParam, 333
 - CoinParamFunc, 332
 - CoinParamType, 332
 - getCommand, 336
 - getDoubleField, 336
 - getIntField, 336
 - getStringField, 335
 - kwdIndex, 334
 - lookupParam, 337
 - matches, 335
 - matchName, 335
 - matchParam, 336
 - printGenericHelp, 339
 - printHelp, 339
 - printIt, 338
 - printLongHelp, 334
 - setDisplay, 335

- setInputSrc, 335
 - setKwdVal, 334
 - setLongHelp, 334
 - shortOrHelpMany, 338
 - shortOrHelpOne, 338
- CoinParam.hpp, 560
- CoinParamFunc
 - CoinParam, 332
- CoinParamType
 - CoinParam, 332
- CoinParamUtils, 28
 - getCommand, 30
 - getDoubleField, 30
 - getIntField, 30
 - getStringField, 29
 - lookupParam, 31
 - matchParam, 30
 - printGenericHelp, 33
 - printHelp, 33
 - printIt, 32
 - setInputSrc, 29
 - shortOrHelpMany, 32
 - shortOrHelpOne, 32
- CoinPartitionedVector, 340
 - CoinPartitionedVector, 344
 - operator=, 344
 - print, 344
 - sort, 344
- CoinPostsolveMatrix, 345
 - assignPresolveToPostsolve, 350
 - CoinPostsolveMatrix, 349, 350
 - link_, 350
- CoinPrePostsolveMatrix, 351
 - acts_, 365
 - bulk0_, 363
 - CoinPrePostsolveMatrix, 362
 - colstat_, 365
 - maxmin_, 364
 - originalColumn_, 363
 - originalRow_, 364
 - rcosts_, 365
 - rowduals_, 364
 - rowstat_, 365
 - setMessageHandler, 363
 - setObjSense, 363
 - sol_, 364
- Status, 362
- CoinPresolveAction, 366
 - ~CoinPresolveAction, 370
 - CoinPresolveAction, 370
 - name, 370
 - next, 371
 - throwCoinError, 370
- CoinPresolveDupcol.hpp, 562
- CoinPresolveEmpty.hpp, 564
- CoinPresolveForcing.hpp, 564
- CoinPresolveImpliedFree.hpp, 565
- CoinPresolveMatrix, 371
 - anyInteger, 383
 - anyInteger_, 386
 - anyProhibited_, 388
 - assignPresolveToPostsolve, 385
 - CoinPresolveMatrix, 381, 382
 - colChanged_, 386
 - colProhibited2, 384
 - colUsed, 384
 - initColsToDo, 384
 - initRowsToDo, 384
 - isInteger, 383
 - maxSubstLevel_, 386
 - pass_, 386
 - presolveOptions_, 387
 - recomputeSums, 383
 - rowChanged_, 387
 - rowProhibited2, 385
 - rowUsed, 385
 - setMatrix, 382
 - setVariableType, 382, 383
 - status_, 386
 - stepColsToDo, 384
 - stepRowsToDo, 385
- CoinPresolveMatrix.hpp, 566
 - ZTOLDP, 569
 - ZTOLDP2, 569
- CoinPresolveMonitor, 388
 - checkAndTell, 391
 - CoinPresolveMonitor, 390, 391
- CoinPresolveSingleton.hpp, 569
- CoinPresolveSubst.hpp, 570
- CoinPresolveZeros.hpp, 571
- CoinRelFltEq, 391
 - CoinRelFltEq, 393

- CoinSearchTree, 393
- CoinSearchTreeBase, 395
 - pop, 397
- CoinSearchTreeCompareBest, 397
- CoinSearchTreeCompareBreadth, 397
- CoinSearchTreeCompareDepth, 398
- CoinSearchTreeComparePreferred, 398
- CoinSearchTreeManager, 399
- CoinSet, 400
- CoinShallowPackedVector, 402
 - ~CoinShallowPackedVector, 407
 - CoinShallowPackedVector, 407
 - CoinShallowPackedVectorUnitTest, 408
 - operator=, 408
- CoinShallowPackedVectorUnitTest
 - CoinShallowPackedVector, 408
- CoinSimpFactorization, 409
 - checkPivot, 421
 - replaceColumn, 421
 - updateColumnFT, 421
- CoinSnapshot, 422
 - getObjValue, 428
 - loadProblem, 429
- CoinSosSet, 429
- CoinStructuredModel, 431
 - addBlock, 436
 - CoinStructuredModel, 436
 - writeMps, 437
- CoinThreadRandom, 437
 - CoinThreadRandom, 439
 - getSeed, 440
 - setSeed, 439
- CoinTimer, 440
- CoinTreeNode, 442
- CoinTreeSiblings, 443
- CoinTriple, 444
- CoinUnsignedIntArrayWithLength, 445
 - CoinUnsignedIntArrayWithLength, 447
 - operator=, 448
- CoinVoidStarArrayWithLength, 448
 - CoinVoidStarArrayWithLength, 450
 - operator=, 451
- CoinWarmStart, 451
- CoinWarmStart.hpp, 571
- CoinWarmStartBasis, 452
 - applyDiff, 459
 - artificialStatus_, 461
 - assignBasisStatus, 461
 - atLowerBound, 457
 - atUpperBound, 457
 - basic, 457
 - CoinWarmStartBasis, 458
 - compressRows, 459
 - deleteColumns, 460
 - deleteRows, 460
 - generateDiff, 458
 - getStructuralStatus, 458
 - isFree, 457
 - mergeBasis, 460
 - numberBasicStructurals, 458
 - resize, 459
 - setSize, 459
 - Status, 457
 - structuralStatus_, 461
- CoinWarmStartBasisDiff, 462
 - CoinWarmStartBasisDiff, 464
- CoinWarmStartDiff, 465
- CoinWarmStartDual, 466
 - applyDiff, 468
 - assignDual, 468
 - generateDiff, 468
- CoinWarmStartDualDiff, 469
 - CoinWarmStartDualDiff, 471
- CoinWarmStartPrimalDual, 472
 - applyDiff, 475
 - assign, 474
 - clear, 474
 - generateDiff, 475
- CoinWarmStartPrimalDualDiff, 475
 - clear, 479
 - clone, 479
 - CoinWarmStartPrimalDualDiff, 478
- CoinWarmStartVector, 479
 - applyDiff, 482
 - assignVector, 481
 - clear, 482
 - generateDiff, 482
- CoinWarmStartVectorDiff, 483
 - clear, 485
 - CoinWarmStartVectorDiff, 485

- CoinWarmStartVectorPair, 486
- CoinWarmStartVectorPairDiff, 487
- CoinYacc, 489
- colChanged_
 - CoinPresolveMatrix, 386
- colOrdered_
 - CoinPackedMatrix, 306
- colProhibited2
 - CoinPresolveMatrix, 384
- colstat_
 - CoinPrePostsolveMatrix, 365
- columnIndex
 - CoinLpIO, 160
 - CoinMpsIO, 249
- columnName
 - CoinLpIO, 160
 - CoinMpsIO, 249
- colUsed
 - CoinPresolveMatrix, 384
- compare
 - CoinPackedVectorBase, 323
- compress
 - CoinPackedMatrix, 293
- COMPRESS_BZIP2
 - CoinFileOutput, 120
- COMPRESS_GZIP
 - CoinFileOutput, 120
- COMPRESS_NONE
 - CoinFileOutput, 120
- Compression
 - CoinFileOutput, 120
- compressionSupported
 - CoinFileOutput, 121
- compressRows
 - CoinWarmStartBasis, 459
- convertMatrix
 - CoinModel, 210
- copy
 - CoinIndexedVector, 139
- copyInIntegerInformation
 - CoinMpsIO, 250
- copyOf
 - CoinPackedMatrix, 294
- copyReuseArrays
 - CoinPackedMatrix, 294
- countOrthoLength
 - CoinPackedMatrix, 297
- countPlusMinusOne
 - CoinModel, 212
- create
 - CoinFileInput, 114
 - CoinFileOutput, 121
 - CoinModelLinkedList, 228
- dealWithFileName
 - CoinMpsIO, 253
- defaultHandler_
 - CoinLpIO, 169
 - CoinMpsIO, 254
- deleteCol
 - CoinModel, 208
- deleteCols
 - CoinPackedMatrix, 292
- deleteColumn
 - CoinModel, 208
- deleteColumns
 - CoinWarmStartBasis, 460
- deleteMajorVectors
 - CoinPackedMatrix, 300
- deleteMinorVectors
 - CoinPackedMatrix, 301
- deleteRow
 - CoinModel, 208
- deleteRows
 - CoinPackedMatrix, 292
 - CoinWarmStartBasis, 460
- deleteSame
 - CoinModelLinkedList, 229
- denseVector
 - CoinIndexedVector, 139
 - CoinPackedVectorBase, 322
- differentModel
 - CoinModel, 210
- do_tighten_action, 490
 - name, 492
- doubleton_action, 492
 - name, 494
- doubleton_action::action, 39
- doubleValue
 - CoinMessageHandler, 182
- drop_empty_cols_action, 494
 - name, 496

- drop_empty_rows_action, 497
 - name, 498
- drop_zero_coefficients_action, 499
 - name, 501
- dropped_zero, 501
- dumpMatrix
 - CoinPackedMatrix, 298
- dupcol_action, 502
 - name, 504
- duprow_action, 504
 - name, 506
- EKKHlink, 506
- element_
 - CoinPackedMatrix, 306
- eliminateDuplicates
 - CoinPackedMatrix, 293
- expandKnapsack
 - CoinModel, 215
- extraGap_
 - CoinPackedMatrix, 306
- extraMajor_
 - CoinPackedMatrix, 306
- factorize
 - CoinFactorization, 99, 100
 - CoinOslFactorization, 263
- factorizePart1
 - CoinFactorization, 100
- FactorPointers, 507
- fileAbsPath
 - CoinFileInput, 115
- fileCoinReadable
 - CoinFileInput, 115
- find_obj
 - CoinLpIO, 166
- findHash
 - CoinLpIO, 165
- findIndex
 - CoinPackedVectorBase, 323
- finish
 - CoinMessageHandler, 186
- first_is_number
 - CoinLpIO, 166
- firstInColumn
 - CoinModel, 211
- firstInQuadraticColumn
 - CoinModel, 212
- firstInRow
 - CoinModel, 211
- forcing_constraint_action, 508
 - name, 510
- forcing_constraint_action::action, 37
- freePreviousNames
 - CoinLpIO, 158
- generateDiff
 - CoinWarmStartBasis, 458
 - CoinWarmStartDual, 468
 - CoinWarmStartPrimalDual, 475
 - CoinWarmStartVector, 482
- getCoefficient
 - CoinPackedMatrix, 293
- getColumn
 - CoinModel, 208
- getColumnSpaceIterate
 - CoinFactorization, 102
- getColumnSpaceIterateR
 - CoinFactorization, 102
- getCommand
 - CoinParam, 336
 - CoinParamUtils, 30
- getDoubleField
 - CoinParam, 336
 - CoinParamUtils, 30
- getElementAsString
 - CoinModel, 210
- getElements
 - CoinPackedMatrix, 287
- getExtraGap
 - CoinPackedMatrix, 285
- getExtraMajor
 - CoinPackedMatrix, 285
- getIndices
 - CoinIndexedVector, 138
 - CoinPackedMatrix, 287
- getIntField
 - CoinParam, 336
 - CoinParamUtils, 30
- getMajorDim
 - CoinPackedMatrix, 297
- getMajorIndices

- CoinPackedMatrix, 289
- getMaxMajorDim
 - CoinPackedMatrix, 298
- getMinorDim
 - CoinPackedMatrix, 298
- getMutableElements
 - CoinPackedMatrix, 302
- getMutableIndices
 - CoinPackedMatrix, 303
- getMutableVectorLengths
 - CoinPackedMatrix, 303
- getMutableVectorStarts
 - CoinPackedMatrix, 303
- getNumCols
 - CoinPackedMatrix, 286
- getNumElements
 - CoinPackedMatrix, 286
- getNumRows
 - CoinPackedMatrix, 287
- getObjValue
 - CoinSnapshot, 428
- getOriginalPosition
 - CoinPackedVector, 315
- getPreviousColNames
 - CoinLpIO, 159
- getPreviousRowNames
 - CoinLpIO, 159
- GetRawPtr
 - Coin::SmartPtr, 541
- getRightHandSide
 - CoinLpIO, 158
 - CoinMpsIO, 248
- getRow
 - CoinModel, 208
- getRowNames
 - CoinLpIO, 159
- getRowRange
 - CoinLpIO, 159
 - CoinMpsIO, 248
- getRowSense
 - CoinLpIO, 158
 - CoinMpsIO, 247
- gets
 - CoinFileInput, 114
- getSeed
 - CoinThreadRandom, 440
- getSizeVectorLengths
 - CoinPackedMatrix, 287
- getSizeVectorStarts
 - CoinPackedMatrix, 287
- getStringField
 - CoinParam, 335
 - CoinParamUtils, 29
- getStructuralStatus
 - CoinWarmStartBasis, 458
- getVector
 - CoinPackedMatrix, 288
- getVectorLast
 - CoinPackedMatrix, 288
- getVectorLengths
 - CoinPackedMatrix, 288
- getVectorSize
 - CoinPackedMatrix, 288
- getVectorStarts
 - CoinPackedMatrix, 288
- gubrow_action, 510
 - name, 512
- hasGaps
 - CoinPackedMatrix, 286
- hash_
 - CoinLpIO, 170
- implied_free_action, 512
 - name, 514
- index_
 - CoinPackedMatrix, 306
- indices
 - CoinOslFactorization, 264
- initColsToDo
 - CoinPresolveMatrix, 384
- initRowsToDo
 - CoinPresolveMatrix, 384
- insertHash
 - CoinLpIO, 165
- integerColumns
 - CoinMpsIO, 248
- integerType_
 - CoinLpIO, 169
- intValue
 - CoinMessageHandler, 182
- is_comment

- CoinLpIO, 166
- is_free
 - CoinLpIO, 167
- is_inf
 - CoinLpIO, 167
- is_invalid_name
 - CoinLpIO, 161
- is_keyword
 - CoinLpIO, 167
- is_sense
 - CoinLpIO, 167
- is_subject_to
 - CoinLpIO, 166
- isColOrdered
 - CoinPackedMatrix, 286
- isEquivalent
 - CoinPackedMatrix, 302
 - CoinPackedVectorBase, 323
- isEquivalent2
 - CoinPackedMatrix, 302
- isExistingIndex
 - CoinPackedVectorBase, 323
- isFree
 - CoinWarmStartBasis, 457
- isInteger
 - CoinMpsIO, 248
 - CoinPresolveMatrix, 383
- IsNull
 - Coin::SmartPtr, 541
- isolated_constraint_action, 515
 - name, 516
- IsValid
 - Coin::SmartPtr, 541
- kwdIndex
 - CoinParam, 334
- Language
 - CoinMessages, 190
- language
 - CoinMessages, 191
- lastInColumn
 - CoinModel, 211
- lastInQuadraticColumn
 - CoinModel, 212
- lastInRow
 - CoinModel, 211
- length_
 - CoinPackedMatrix, 307
- lengthMessages_
 - CoinMessages, 192
- lengthU_
 - CoinFactorization, 104
- link_
 - CoinPostsolveMatrix, 350
- loadBlock
 - CoinModel, 213, 214
- loadProblem
 - CoinSnapshot, 429
- logLevel
 - CoinMessageHandler, 181
- logLevel_
 - CoinBaseModel, 54
- lookupParam
 - CoinParam, 337
 - CoinParamUtils, 31
- majorAppendOrthoOrdered
 - CoinPackedMatrix, 300
- majorAppendSameOrdered
 - CoinPackedMatrix, 299
- make_fixed
 - make_fixed_action, 519
- make_fixed_action, 516
 - make_fixed, 519
 - postsolve, 519
 - presolve, 519
 - transferCosts, 519
- matches
 - CoinParam, 335
- matchName
 - CoinParam, 335
- matchParam
 - CoinParam, 336
 - CoinParamUtils, 30
- maxHash_
 - CoinLpIO, 170
- maxmin_
 - CoinPrePostsolveMatrix, 364
- maxSubstLevel_
 - CoinPresolveMatrix, 386
- mergeBasis

- CoinWarmStartBasis, 460
- message
 - CoinMessageHandler, 184
- minorAppendOrthoOrdered
 - CoinPackedMatrix, 300
- minorAppendSameOrdered
 - CoinPackedMatrix, 300
- modifyCoefficient
 - CoinPackedMatrix, 292
- mpsType
 - CoinMpsCardReader, 235
- MtxManip
 - presolve_delete_from_col, 22
 - presolve_delete_from_col2, 22
 - presolve_delete_from_major, 21
 - presolve_delete_from_major2, 22
 - presolve_delete_from_row, 22
 - presolve_delete_many_from_major, 21
 - presolve_expand_major, 18
 - presolve_find_col, 19
 - presolve_find_col1, 20
 - presolve_find_minor, 18
 - presolve_find_minor1, 19
 - presolve_find_minor2, 20
 - presolve_find_minor3, 20
 - presolve_find_row, 19
 - presolve_find_row1, 19
 - presolve_find_row2, 20
 - presolve_find_row3, 21
- name
 - CoinPresolveAction, 370
 - do_tighten_action, 492
 - doubleton_action, 494
 - drop_empty_cols_action, 496
 - drop_empty_rows_action, 498
 - drop_zero_coefficients_action, 501
 - dupcol_action, 504
 - duprow_action, 506
 - forcing_constraint_action, 510
 - gubrow_action, 512
 - implied_free_action, 514
 - isolated_constraint_action, 516
 - slack_doubleton_action, 533
 - slack_singleton_action, 536
 - subst_constraint_action, 545
 - tripleton_action, 549
 - twoxtwo_action, 551
 - useless_constraint_action, 553
- names_
 - CoinLpIO, 169
- next
 - CoinModel, 212
 - CoinPresolveAction, 371
- nextGmsField
 - CoinMpsCardReader, 235
- nullElementArray
 - CoinPackedMatrix, 303
- nullIndexArray
 - CoinPackedMatrix, 304
- nullLengthArray
 - CoinPackedMatrix, 304
- nullStartArray
 - CoinPackedMatrix, 303
- numberBasicStructurals
 - CoinWarmStartBasis, 458
- numberCharFields
 - CoinMessageHandler, 183
- numberDoubleFields
 - CoinMessageHandler, 182
- numberHash_
 - CoinLpIO, 170
- numberIntFields
 - CoinMessageHandler, 182
- numberStringFields
 - CoinMessageHandler, 183
- numberTrials_
 - CoinFactorization, 104
- objectiveOffset
 - CoinMpsIO, 249
- operator<<
 - CoinMessageHandler, 184, 185
- operator*
 - Coin::SmartPtr, 542
- operator->
 - Coin::SmartPtr, 542
- operator=
 - CoinArbitraryArrayWithLength, 45
 - CoinArrayWithLength, 50
 - CoinBigIndexArrayWithLength, 57

- CoinDoubleArrayWithLength, 73
- CoinFactorizationDoubleArrayWithLength, 107
- CoinFactorizationLongDoubleArrayWithLength, 110
- CoinIndexedVector, 139
- CoinIntArrayWithLength, 145
- CoinMessageHandler, 181
- CoinMessages, 191
- CoinOneMessage, 257
- CoinPackedMatrix, 295
- CoinPackedVector, 315
- CoinPartitionedVector, 344
- CoinShallowPackedVector, 408
- CoinUnsignedIntArrayWithLength, 448
- CoinVoidStarArrayWithLength, 451
- operator==
 - Coin::SmartPtr, 542
 - CoinIndexedVector, 141
 - CoinPackedVectorBase, 323
- originalColumn_
 - CoinPrePostsolveMatrix, 363
- originalRow_
 - CoinPrePostsolveMatrix, 364
- out_coeff
 - CoinLpIO, 166
- pack
 - CoinModel, 209
- packCols
 - CoinModel, 209
- packColumns
 - CoinModel, 209
- packRows
 - CoinModel, 208
- pass_
 - CoinPresolveMatrix, 386
- passInMessageHandler
 - CoinLpIO, 165
 - CoinMpsIO, 253
- pointer
 - CoinModel, 210
- pop
 - CoinSearchTreeBase, 397
- position
 - CoinModel, 211
- postsolve
 - make_fixed_action, 519
 - remove_dual_action, 527
- preProcess
 - CoinFactorization, 101
- presolve
 - make_fixed_action, 519
 - remove_dual_action, 527
 - remove_fixed_action, 531
 - slack_doubleton_action, 533
- Presolve Debug Functions, 24
- Presolve Matrix Manipulation Functions, 16
- Presolve Utility Functions, 23
- presolve_check_duals
 - PresolveDebugFunctions, 27
- presolve_check_free_list
 - PresolveDebugFunctions, 26
- presolve_check_reduced_costs
 - PresolveDebugFunctions, 27
- presolve_check_sol
 - PresolveDebugFunctions, 27
- presolve_consistent
 - PresolveDebugFunctions, 26
- presolve_delete_from_col
 - MtxManip, 22
- presolve_delete_from_col2
 - MtxManip, 22
- presolve_delete_from_major
 - MtxManip, 21
- presolve_delete_from_major2
 - MtxManip, 22
- presolve_delete_from_row
 - MtxManip, 22
- presolve_delete_many_from_major
 - MtxManip, 21
- presolve_dupmajor
 - PresolveUtilities, 23
- presolve_expand_major
 - MtxManip, 18
- presolve_find_col
 - MtxManip, 19
- presolve_find_col1
 - MtxManip, 20
- presolve_find_minor

- MtxManip, 18
- presolve_find_minor1
 - MtxManip, 19
- presolve_find_minor2
 - MtxManip, 20
- presolve_find_minor3
 - MtxManip, 20
- presolve_find_row
 - MtxManip, 19
- presolve_find_row1
 - MtxManip, 19
- presolve_find_row2
 - MtxManip, 20
- presolve_find_row3
 - MtxManip, 21
- PRESOLVE_INSERT_LINK
 - presolvehlink, 521
- presolve_links_ok
 - PresolveDebugFunctions, 26
- PRESOLVE_MOVE_LINK
 - presolvehlink, 521
- presolve_no_dups
 - PresolveDebugFunctions, 26
- presolve_no_zeros
 - PresolveDebugFunctions, 26
- PRESOLVE_REMOVE_LINK
 - presolvehlink, 521
- PresolveDebugFunctions
 - presolve_check_duals, 27
 - presolve_check_free_list, 26
 - presolve_check_reduced_costs, 27
 - presolve_check_sol, 27
 - presolve_consistent, 26
 - presolve_links_ok, 26
 - presolve_no_dups, 26
 - presolve_no_zeros, 26
- presolvehlink, 520
 - PRESOLVE_INSERT_LINK, 521
 - PRESOLVE_MOVE_LINK, 521
 - PRESOLVE_REMOVE_LINK, 521
- presolveOptions_
 - CoinPresolveMatrix, 387
- PresolveUtilities
 - presolve_dupmajor, 23
- previous
 - CoinModel, 212
- previous_names_
 - CoinLpIO, 169
- print
 - CoinPartitionedVector, 344
- printGenericHelp
 - CoinParam, 339
 - CoinParamUtils, 33
- printHelp
 - CoinParam, 339
 - CoinParamUtils, 33
- printing
 - CoinMessageHandler, 186
- printIt
 - CoinParam, 338
 - CoinParamUtils, 32
- printLongHelp
 - CoinParam, 334
- printStatus_
 - CoinMessageHandler, 186
- puts
 - CoinFileOutput, 122
- quickAdd
 - CoinIndexedVector, 140
- quickAddNonZero
 - CoinIndexedVector, 140
- rcosts_
 - CoinPrePostsolveMatrix, 365
- read
 - CoinFileInput, 114
- read_monom_obj
 - CoinLpIO, 167
- read_monom_row
 - CoinLpIO, 168
- readBasis
 - CoinMpsIO, 250
- readConicMps
 - CoinMpsIO, 252
- readGms
 - CoinMpsIO, 251
- readLp
 - CoinLpIO, 164
- readMps
 - CoinMpsIO, 250
- readQuadraticMps

- CoinMpsIO, 252
- recomputeSums
 - CoinPresolveMatrix, 383
- releaseRedundantInformation
 - CoinMpsIO, 253
- remove_dual_action, 525
 - postsolve, 527
 - presolve, 527
- remove_fixed
 - remove_fixed_action, 531
- remove_fixed_action, 528
 - presolve, 531
 - remove_fixed, 531
- remove_fixed_action::action, 36
- reorder
 - CoinModel, 214
- replaceColumn
 - CoinDenseFactorization, 65
 - CoinFactorization, 100
 - CoinOslFactorization, 263
 - CoinOtherFactorization, 271
 - CoinSimpFactorization, 421
- replaceColumnPFI
 - CoinFactorization, 103
- replaceMessage
 - CoinMessages, 191
- replaceVector
 - CoinPackedMatrix, 292
- reserve
 - CoinIndexedVector, 142
 - CoinPackedMatrix, 286
 - CoinPackedVector, 316
- resize
 - CoinDenseVector, 69
 - CoinWarmStartBasis, 459
- restoreFactorization
 - CoinFactorization, 99
- returnVector
 - CoinIndexedVector, 139
- reverseOrderedCopyOf
 - CoinPackedMatrix, 295
- reverseOrdering
 - CoinPackedMatrix, 295
- rightAppendPackedMatrix
 - CoinPackedMatrix, 291
- rowChanged_
 - CoinPresolveMatrix, 387
- rowduals_
 - CoinPrePostsolveMatrix, 364
- rowIndex
 - CoinLpIO, 160
 - CoinMpsIO, 249
- rowName
 - CoinLpIO, 160
 - CoinMpsIO, 249
- rowProhibited2
 - CoinPresolveMatrix, 385
- rowstat_
 - CoinPrePostsolveMatrix, 365
- rowUsed
 - CoinPresolveMatrix, 385
- scan
 - CoinIndexedVector, 141
- setDecimals
 - CoinLpIO, 161
- setDefaultRowNames
 - CoinLpIO, 162
- setDetailMessages
 - CoinMessages, 191
- setDimensions
 - CoinPackedMatrix, 289
- setDisplay
 - CoinParam, 335
- setEpsilon
 - CoinLpIO, 160
- setExternalNumber
 - CoinOneMessage, 257
- setExtraGap
 - CoinPackedMatrix, 289
- setExtraMajor
 - CoinPackedMatrix, 289
- setInfinity
 - CoinLpIO, 160
- setInputSrc
 - CoinParam, 335
 - CoinParamUtils, 29
- setKwdVal
 - CoinParam, 334
- setLogLevel
 - CoinMessageHandler, 181, 182
- setLongHelp

- CoinParam, 334
- setLpDataRowAndColNames
 - CoinLpIO, 162
- setLpDataWithoutRowAndColNames
 - CoinLpIO, 161
- setMajorDim
 - CoinPackedMatrix, 297
- setMatrix
 - CoinPresolveMatrix, 382
- setMessageHandler
 - CoinPrePostsolveMatrix, 363
- setMinorDim
 - CoinPackedMatrix, 298
- setNumberAcross
 - CoinLpIO, 161
- setObjSense
 - CoinPrePostsolveMatrix, 363
- setSeed
 - CoinThreadRandom, 439
- setSize
 - CoinWarmStartBasis, 459
- setSolveMode
 - CoinOtherFactorization, 271
- setTestForDuplicateIndex
 - CoinPackedVectorBase, 322
- setTestForDuplicateIndexWhenTrue
 - CoinPackedVectorBase, 322
- setVariableType
 - CoinPresolveMatrix, 382, 383
- setVector
 - CoinDenseVector, 69
 - CoinIndexedVector, 140
 - CoinPackedVector, 316
- shortOrHelpMany
 - CoinParam, 338
 - CoinParamUtils, 32
- shortOrHelpOne
 - CoinParam, 338
 - CoinParamUtils, 32
- slack_doubleton_action, 531
 - name, 533
 - presolve, 533
- slack_singleton_action, 534
 - name, 536
- sol_
 - CoinPrePostsolveMatrix, 364
- solveMode
 - CoinOtherFactorization, 271
- solveMode_
 - CoinOtherFactorization, 272
- sort
 - CoinIndexedVector, 142
 - CoinPackedVector, 316
 - CoinPartitionedVector, 344
- sortOriginalOrder
 - CoinPackedVector, 316
- start_
 - CoinPackedMatrix, 307
- startHash
 - CoinLpIO, 165
- Status
 - CoinPrePostsolveMatrix, 362
 - CoinWarmStartBasis, 457
- status_
 - CoinPresolveMatrix, 386
- stepColsToDo
 - CoinPresolveMatrix, 384
- stepRowsToDo
 - CoinPresolveMatrix, 385
- stopHash
 - CoinLpIO, 165
- stringValue
 - CoinMessageHandler, 183
- structuralStatus_
 - CoinWarmStartBasis, 461
- submatrixOf
 - CoinPackedMatrix, 293
- submatrixOfWithDuplicates
 - CoinPackedMatrix, 294
- subst_constraint_action, 543
 - name, 545
- swap
 - CoinPackedMatrix, 296
- symrec, 545
- testForDuplicateIndex
 - CoinPackedVectorBase, 322
- testRedundant
 - useless_constraint_action, 553
- throwCoinError
 - CoinPresolveAction, 370
- times

- CoinPackedMatrix, 296
- timesMajor
 - CoinPackedMatrix, 301
- timesMinor
 - CoinPackedMatrix, 301
- transferCosts
 - make_fixed_action, 519
- transpose
 - CoinPackedMatrix, 296
- transposeTimes
 - CoinPackedMatrix, 296, 297
- tripleton_action, 546
 - name, 549
- tripleton_action::action, 38
- truncate
 - CoinPackedVector, 316
- twotwo_action, 549
 - name, 551
- updateColumnFT
 - CoinDenseFactorization, 65
 - CoinFactorization, 100
 - CoinOslFactorization, 264
 - CoinOtherFactorization, 271
 - CoinSimpFactorization, 421
- updateColumnRFT
 - CoinFactorization, 102
- updateColumnTransposeU
 - CoinFactorization, 102
- updateColumnTransposeUByColumn
 - CoinFactorization, 103
- updateColumnTransposeUDensish
 - CoinFactorization, 103
- updateColumnTransposeUSparse
 - CoinFactorization, 103
- updateColumnTransposeUSparsish
 - CoinFactorization, 102
- updateDeleted
 - CoinModelLinkedList, 229
- updateTwoColumnsFT
 - CoinFactorization, 101
- useless_constraint_action, 552
 - name, 553
 - testRedundant, 553
- verifyMtx
 - CoinPackedMatrix, 304
- write
 - CoinFileOutput, 121
- writeLp
 - CoinLpIO, 163, 164
- writeMps
 - CoinModel, 209
 - CoinMpsIO, 251
 - CoinStructuredModel, 437
- zero
 - CoinIndexedVector, 141
- ZTOLDP
 - CoinPresolveMatrix.hpp, 569
- ZTOLDP2
 - CoinPresolveMatrix.hpp, 569