

Cbc  
trunk

Generated by Doxygen 1.8.5

Mon Oct 21 2013 19:03:07

## Contents

<b>1</b>	<b>Todo List</b>	<b>1</b>
<b>2</b>	<b>Namespace Index</b>	<b>2</b>
2.1	Namespace List . . . . .	2
<b>3</b>	<b>Hierarchical Index</b>	<b>2</b>
3.1	Class Hierarchy . . . . .	2
<b>4</b>	<b>Class Index</b>	<b>10</b>
4.1	Class List . . . . .	10
<b>5</b>	<b>File Index</b>	<b>17</b>
5.1	File List . . . . .	17
<b>6</b>	<b>Namespace Documentation</b>	<b>20</b>
6.1	CbcCbcParamUtils Namespace Reference . . . . .	20
6.1.1	Function Documentation . . . . .	20
6.2	CbcGenParamUtils Namespace Reference . . . . .	21
6.2.1	Function Documentation . . . . .	21
6.3	CbcOsiParamUtils Namespace Reference . . . . .	22
6.3.1	Function Documentation . . . . .	22
<b>7</b>	<b>Class Documentation</b>	<b>22</b>
7.1	ampl_info Struct Reference . . . . .	22
7.1.1	Detailed Description . . . . .	23
7.1.2	Member Data Documentation . . . . .	23
7.2	CbcGenCtIBlk::babState_struct Struct Reference . . . . .	26
7.2.1	Detailed Description . . . . .	27
7.2.2	Member Data Documentation . . . . .	27
7.3	CbcBaseModel Class Reference . . . . .	27
7.3.1	Detailed Description . . . . .	27
7.3.2	Constructor & Destructor Documentation . . . . .	27
7.4	CbcBranchAllDifferent Class Reference . . . . .	28
7.4.1	Detailed Description . . . . .	29
7.4.2	Constructor & Destructor Documentation . . . . .	29
7.4.3	Member Function Documentation . . . . .	29
7.4.4	Member Data Documentation . . . . .	29
7.5	CbcBranchCut Class Reference . . . . .	30

7.5.1	Detailed Description	31
7.5.2	Constructor & Destructor Documentation	31
7.5.3	Member Function Documentation	31
7.6	CbcBranchDecision Class Reference	32
7.6.1	Detailed Description	33
7.6.2	Constructor & Destructor Documentation	34
7.6.3	Member Function Documentation	34
7.6.4	Member Data Documentation	35
7.7	CbcBranchDefaultDecision Class Reference	36
7.7.1	Detailed Description	37
7.7.2	Constructor & Destructor Documentation	37
7.7.3	Member Function Documentation	37
7.8	CbcBranchDynamicDecision Class Reference	38
7.8.1	Detailed Description	39
7.8.2	Constructor & Destructor Documentation	39
7.8.3	Member Function Documentation	39
7.9	CbcBranchingObject Class Reference	40
7.9.1	Detailed Description	43
7.9.2	Constructor & Destructor Documentation	43
7.9.3	Member Function Documentation	43
7.9.4	Member Data Documentation	46
7.10	CbcBranchToFixLots Class Reference	47
7.10.1	Detailed Description	48
7.10.2	Constructor & Destructor Documentation	48
7.10.3	Member Function Documentation	48
7.10.4	Member Data Documentation	49
7.11	CbcCbcParam Class Reference	50
7.11.1	Detailed Description	51
7.11.2	Member Enumeration Documentation	51
7.11.3	Constructor & Destructor Documentation	52
7.11.4	Member Function Documentation	53
7.12	CbcClique Class Reference	54
7.12.1	Detailed Description	55
7.12.2	Constructor & Destructor Documentation	56
7.12.3	Member Function Documentation	56
7.12.4	Member Data Documentation	57
7.13	CbcCliqueBranchingObject Class Reference	58

7.13.1 Detailed Description . . . . .	59
7.13.2 Constructor & Destructor Documentation . . . . .	59
7.13.3 Member Function Documentation . . . . .	59
7.14 CbcCompare Class Reference . . . . .	60
7.14.1 Detailed Description . . . . .	61
7.14.2 Constructor & Destructor Documentation . . . . .	61
7.14.3 Member Function Documentation . . . . .	61
7.14.4 Member Data Documentation . . . . .	61
7.15 CbcCompareBase Class Reference . . . . .	61
7.15.1 Detailed Description . . . . .	62
7.15.2 Constructor & Destructor Documentation . . . . .	62
7.15.3 Member Function Documentation . . . . .	63
7.15.4 Member Data Documentation . . . . .	64
7.16 CbcCompareDefault Class Reference . . . . .	64
7.16.1 Detailed Description . . . . .	66
7.16.2 Constructor & Destructor Documentation . . . . .	66
7.16.3 Member Function Documentation . . . . .	66
7.16.4 Member Data Documentation . . . . .	68
7.17 CbcCompareDepth Class Reference . . . . .	69
7.17.1 Detailed Description . . . . .	69
7.17.2 Constructor & Destructor Documentation . . . . .	70
7.17.3 Member Function Documentation . . . . .	70
7.18 CbcCompareEstimate Class Reference . . . . .	70
7.18.1 Detailed Description . . . . .	71
7.18.2 Constructor & Destructor Documentation . . . . .	71
7.18.3 Member Function Documentation . . . . .	71
7.19 CbcCompareObjective Class Reference . . . . .	71
7.19.1 Detailed Description . . . . .	72
7.19.2 Constructor & Destructor Documentation . . . . .	72
7.19.3 Member Function Documentation . . . . .	72
7.20 CbcConsequence Class Reference . . . . .	73
7.20.1 Detailed Description . . . . .	73
7.20.2 Constructor & Destructor Documentation . . . . .	73
7.20.3 Member Function Documentation . . . . .	73
7.21 CbcCountRowCut Class Reference . . . . .	74
7.21.1 Detailed Description . . . . .	75
7.21.2 Constructor & Destructor Documentation . . . . .	75

7.21.3	Member Function Documentation	75
7.22	CbcCutBranchingObject Class Reference	76
7.22.1	Detailed Description	77
7.22.2	Constructor & Destructor Documentation	77
7.22.3	Member Function Documentation	78
7.22.4	Member Data Documentation	79
7.23	CbcCutGenerator Class Reference	79
7.23.1	Detailed Description	82
7.23.2	Constructor & Destructor Documentation	82
7.23.3	Member Function Documentation	83
7.24	CbcCutModifier Class Reference	90
7.24.1	Detailed Description	90
7.24.2	Constructor & Destructor Documentation	90
7.24.3	Member Function Documentation	91
7.25	CbcCutSubsetModifier Class Reference	91
7.25.1	Detailed Description	92
7.25.2	Constructor & Destructor Documentation	92
7.25.3	Member Function Documentation	92
7.25.4	Member Data Documentation	93
7.26	CbcDummyBranchingObject Class Reference	93
7.26.1	Detailed Description	94
7.26.2	Constructor & Destructor Documentation	94
7.26.3	Member Function Documentation	94
7.27	CbcDynamicPseudoCostBranchingObject Class Reference	95
7.27.1	Detailed Description	97
7.27.2	Constructor & Destructor Documentation	97
7.27.3	Member Function Documentation	97
7.27.4	Member Data Documentation	99
7.28	CbcEventHandler Class Reference	99
7.28.1	Detailed Description	100
7.28.2	Member Typedef Documentation	100
7.28.3	Member Enumeration Documentation	101
7.28.4	Constructor & Destructor Documentation	101
7.28.5	Member Function Documentation	102
7.28.6	Member Data Documentation	102
7.29	CbcFathom Class Reference	103
7.29.1	Detailed Description	104

7.29.2	Constructor & Destructor Documentation	104
7.29.3	Member Function Documentation	104
7.29.4	Member Data Documentation	104
7.30	CbcFathomDynamicProgramming Class Reference	105
7.30.1	Detailed Description	107
7.30.2	Constructor & Destructor Documentation	107
7.30.3	Member Function Documentation	107
7.30.4	Member Data Documentation	108
7.31	CbcFeasibilityBase Class Reference	110
7.31.1	Detailed Description	110
7.31.2	Constructor & Destructor Documentation	110
7.31.3	Member Function Documentation	111
7.32	CbcFixingBranchingObject Class Reference	111
7.32.1	Detailed Description	112
7.32.2	Constructor & Destructor Documentation	112
7.32.3	Member Function Documentation	112
7.33	CbcFixVariable Class Reference	113
7.33.1	Detailed Description	114
7.33.2	Constructor & Destructor Documentation	114
7.33.3	Member Function Documentation	114
7.33.4	Member Data Documentation	115
7.34	CbcFollowOn Class Reference	115
7.34.1	Detailed Description	116
7.34.2	Constructor & Destructor Documentation	117
7.34.3	Member Function Documentation	117
7.34.4	Member Data Documentation	117
7.35	CbcFullNodeInfo Class Reference	118
7.35.1	Detailed Description	119
7.35.2	Constructor & Destructor Documentation	119
7.35.3	Member Function Documentation	120
7.35.4	Member Data Documentation	121
7.36	CbcGenCtlBlk Class Reference	121
7.36.1	Detailed Description	126
7.36.2	Member Enumeration Documentation	126
7.36.3	Constructor & Destructor Documentation	129
7.36.4	Member Function Documentation	130
7.36.5	Friends And Related Function Documentation	134

7.36.6	Member Data Documentation	134
7.37	CbcGeneral Class Reference	137
7.37.1	Detailed Description	138
7.37.2	Constructor & Destructor Documentation	138
7.37.3	Member Function Documentation	138
7.38	CbcGenParam Class Reference	139
7.38.1	Detailed Description	140
7.38.2	Member Enumeration Documentation	140
7.38.3	Constructor & Destructor Documentation	142
7.38.4	Member Function Documentation	143
7.39	CbcHeuristic Class Reference	143
7.39.1	Detailed Description	147
7.39.2	Constructor & Destructor Documentation	148
7.39.3	Member Function Documentation	148
7.39.4	Member Data Documentation	152
7.40	CbcHeuristicCrossover Class Reference	155
7.40.1	Detailed Description	156
7.40.2	Constructor & Destructor Documentation	156
7.40.3	Member Function Documentation	156
7.40.4	Member Data Documentation	157
7.41	CbcHeuristicDINS Class Reference	157
7.41.1	Detailed Description	159
7.41.2	Constructor & Destructor Documentation	159
7.41.3	Member Function Documentation	159
7.41.4	Member Data Documentation	160
7.42	CbcHeuristicDive Class Reference	161
7.42.1	Detailed Description	163
7.42.2	Constructor & Destructor Documentation	163
7.42.3	Member Function Documentation	163
7.42.4	Member Data Documentation	165
7.43	CbcHeuristicDiveCoefficient Class Reference	166
7.43.1	Detailed Description	167
7.43.2	Constructor & Destructor Documentation	167
7.43.3	Member Function Documentation	167
7.44	CbcHeuristicDiveFractional Class Reference	168
7.44.1	Detailed Description	169
7.44.2	Constructor & Destructor Documentation	169

7.44.3	Member Function Documentation	169
7.45	CbcHeuristicDiveGuided Class Reference	170
7.45.1	Detailed Description	170
7.45.2	Constructor & Destructor Documentation	170
7.45.3	Member Function Documentation	171
7.46	CbcHeuristicDiveLineSearch Class Reference	171
7.46.1	Detailed Description	172
7.46.2	Constructor & Destructor Documentation	172
7.46.3	Member Function Documentation	172
7.47	CbcHeuristicDivePseudoCost Class Reference	173
7.47.1	Detailed Description	174
7.47.2	Constructor & Destructor Documentation	174
7.47.3	Member Function Documentation	174
7.48	CbcHeuristicDiveVectorLength Class Reference	175
7.48.1	Detailed Description	175
7.48.2	Constructor & Destructor Documentation	175
7.48.3	Member Function Documentation	176
7.49	CbcHeuristicDW Class Reference	176
7.49.1	Detailed Description	181
7.49.2	Member Typedef Documentation	181
7.49.3	Constructor & Destructor Documentation	181
7.49.4	Member Function Documentation	181
7.49.5	Member Data Documentation	185
7.50	CbcHeuristicDynamic3 Class Reference	190
7.50.1	Detailed Description	191
7.50.2	Constructor & Destructor Documentation	191
7.50.3	Member Function Documentation	191
7.51	CbcHeuristicFPump Class Reference	192
7.51.1	Detailed Description	195
7.51.2	Constructor & Destructor Documentation	195
7.51.3	Member Function Documentation	195
7.51.4	Member Data Documentation	199
7.52	CbcHeuristicGreedyCover Class Reference	201
7.52.1	Detailed Description	202
7.52.2	Constructor & Destructor Documentation	202
7.52.3	Member Function Documentation	202
7.52.4	Member Data Documentation	203



7.53 CbcHeuristicGreedyEquality Class Reference . . . . .	203
7.53.1 Detailed Description . . . . .	205
7.53.2 Constructor & Destructor Documentation . . . . .	205
7.53.3 Member Function Documentation . . . . .	205
7.53.4 Member Data Documentation . . . . .	206
7.54 CbcHeuristicGreedySOS Class Reference . . . . .	207
7.54.1 Detailed Description . . . . .	208
7.54.2 Constructor & Destructor Documentation . . . . .	208
7.54.3 Member Function Documentation . . . . .	208
7.54.4 Member Data Documentation . . . . .	209
7.55 CbcHeuristicJustOne Class Reference . . . . .	210
7.55.1 Detailed Description . . . . .	211
7.55.2 Constructor & Destructor Documentation . . . . .	211
7.55.3 Member Function Documentation . . . . .	211
7.55.4 Member Data Documentation . . . . .	212
7.56 CbcHeuristicLocal Class Reference . . . . .	212
7.56.1 Detailed Description . . . . .	213
7.56.2 Constructor & Destructor Documentation . . . . .	213
7.56.3 Member Function Documentation . . . . .	213
7.56.4 Member Data Documentation . . . . .	214
7.57 CbcHeuristicNaive Class Reference . . . . .	215
7.57.1 Detailed Description . . . . .	216
7.57.2 Constructor & Destructor Documentation . . . . .	216
7.57.3 Member Function Documentation . . . . .	216
7.57.4 Member Data Documentation . . . . .	217
7.58 CbcHeuristicNode Class Reference . . . . .	217
7.58.1 Detailed Description . . . . .	217
7.58.2 Constructor & Destructor Documentation . . . . .	217
7.58.3 Member Function Documentation . . . . .	218
7.59 CbcHeuristicNodeList Class Reference . . . . .	218
7.59.1 Detailed Description . . . . .	218
7.59.2 Constructor & Destructor Documentation . . . . .	218
7.59.3 Member Function Documentation . . . . .	218
7.60 CbcHeuristicPartial Class Reference . . . . .	219
7.60.1 Detailed Description . . . . .	220
7.60.2 Constructor & Destructor Documentation . . . . .	220
7.60.3 Member Function Documentation . . . . .	220

7.60.4	Member Data Documentation	221
7.61	CbcHeuristicPivotAndFix Class Reference	221
7.61.1	Detailed Description	222
7.61.2	Constructor & Destructor Documentation	222
7.61.3	Member Function Documentation	222
7.62	CbcHeuristicProximity Class Reference	223
7.62.1	Detailed Description	224
7.62.2	Constructor & Destructor Documentation	224
7.62.3	Member Function Documentation	224
7.62.4	Member Data Documentation	225
7.63	CbcHeuristicRandRound Class Reference	225
7.63.1	Detailed Description	226
7.63.2	Constructor & Destructor Documentation	226
7.63.3	Member Function Documentation	226
7.64	CbcHeuristicRENS Class Reference	227
7.64.1	Detailed Description	228
7.64.2	Constructor & Destructor Documentation	228
7.64.3	Member Function Documentation	228
7.64.4	Member Data Documentation	229
7.65	CbcHeuristicRINS Class Reference	229
7.65.1	Detailed Description	230
7.65.2	Constructor & Destructor Documentation	231
7.65.3	Member Function Documentation	231
7.65.4	Member Data Documentation	232
7.66	CbcHeuristicVND Class Reference	233
7.66.1	Detailed Description	234
7.66.2	Constructor & Destructor Documentation	234
7.66.3	Member Function Documentation	234
7.66.4	Member Data Documentation	235
7.67	CbcIdiotBranch Class Reference	236
7.67.1	Detailed Description	237
7.67.2	Constructor & Destructor Documentation	237
7.67.3	Member Function Documentation	237
7.67.4	Member Data Documentation	238
7.68	CbcIntegerBranchingObject Class Reference	239
7.68.1	Detailed Description	240
7.68.2	Constructor & Destructor Documentation	240

7.68.3	Member Function Documentation	241
7.68.4	Member Data Documentation	242
7.69	CbcIntegerPseudoCostBranchingObject Class Reference	242
7.69.1	Detailed Description	244
7.69.2	Constructor & Destructor Documentation	244
7.69.3	Member Function Documentation	244
7.69.4	Member Data Documentation	245
7.70	CbcLongCliqueBranchingObject Class Reference	245
7.70.1	Detailed Description	246
7.70.2	Constructor & Destructor Documentation	246
7.70.3	Member Function Documentation	247
7.71	CbcLotsize Class Reference	247
7.71.1	Detailed Description	249
7.71.2	Constructor & Destructor Documentation	249
7.71.3	Member Function Documentation	249
7.72	CbcLotsizeBranchingObject Class Reference	251
7.72.1	Detailed Description	252
7.72.2	Constructor & Destructor Documentation	252
7.72.3	Member Function Documentation	253
7.72.4	Member Data Documentation	254
7.73	CbcMessage Class Reference	254
7.73.1	Detailed Description	254
7.73.2	Constructor & Destructor Documentation	255
7.74	CbcModel Class Reference	255
7.74.1	Detailed Description	271
7.74.2	Member Enumeration Documentation	272
7.74.3	Constructor & Destructor Documentation	273
7.74.4	Member Function Documentation	273
7.75	CbcNode Class Reference	310
7.75.1	Detailed Description	313
7.75.2	Constructor & Destructor Documentation	313
7.75.3	Member Function Documentation	313
7.76	CbcNodeInfo Class Reference	318
7.76.1	Detailed Description	321
7.76.2	Constructor & Destructor Documentation	321
7.76.3	Member Function Documentation	321
7.76.4	Member Data Documentation	325

7.77 CbcNWay Class Reference . . . . .	326
7.77.1 Detailed Description . . . . .	328
7.77.2 Constructor & Destructor Documentation . . . . .	328
7.77.3 Member Function Documentation . . . . .	328
7.77.4 Member Data Documentation . . . . .	329
7.78 CbcNWayBranchingObject Class Reference . . . . .	329
7.78.1 Detailed Description . . . . .	330
7.78.2 Constructor & Destructor Documentation . . . . .	331
7.78.3 Member Function Documentation . . . . .	331
7.79 CbcObject Class Reference . . . . .	332
7.79.1 Detailed Description . . . . .	334
7.79.2 Constructor & Destructor Documentation . . . . .	334
7.79.3 Member Function Documentation . . . . .	335
7.79.4 Member Data Documentation . . . . .	338
7.80 CbcObjectUpdateData Class Reference . . . . .	339
7.80.1 Detailed Description . . . . .	339
7.80.2 Constructor & Destructor Documentation . . . . .	340
7.80.3 Member Function Documentation . . . . .	340
7.80.4 Member Data Documentation . . . . .	340
7.81 CbcOsiParam Class Reference . . . . .	341
7.81.1 Detailed Description . . . . .	343
7.81.2 Member Enumeration Documentation . . . . .	343
7.81.3 Constructor & Destructor Documentation . . . . .	344
7.81.4 Member Function Documentation . . . . .	345
7.82 CbcOsiSolver Class Reference . . . . .	346
7.82.1 Detailed Description . . . . .	347
7.82.2 Constructor & Destructor Documentation . . . . .	347
7.82.3 Member Function Documentation . . . . .	347
7.82.4 Member Data Documentation . . . . .	347
7.83 CbcParam Class Reference . . . . .	348
7.83.1 Detailed Description . . . . .	350
7.83.2 Constructor & Destructor Documentation . . . . .	350
7.83.3 Member Function Documentation . . . . .	350
7.84 CbcGenCtlBlk::cbcParamsInfo_struct Struct Reference . . . . .	353
7.84.1 Detailed Description . . . . .	353
7.84.2 Member Data Documentation . . . . .	353
7.85 CbcPartialNodeInfo Class Reference . . . . .	354

7.85.1 Detailed Description . . . . .	355
7.85.2 Constructor & Destructor Documentation . . . . .	355
7.85.3 Member Function Documentation . . . . .	355
7.85.4 Member Data Documentation . . . . .	356
7.86 CbcRounding Class Reference . . . . .	356
7.86.1 Detailed Description . . . . .	357
7.86.2 Constructor & Destructor Documentation . . . . .	357
7.86.3 Member Function Documentation . . . . .	358
7.86.4 Member Data Documentation . . . . .	359
7.87 CbcRowCuts Class Reference . . . . .	359
7.87.1 Detailed Description . . . . .	359
7.87.2 Constructor & Destructor Documentation . . . . .	360
7.87.3 Member Function Documentation . . . . .	360
7.88 CbcSerendipity Class Reference . . . . .	360
7.88.1 Detailed Description . . . . .	361
7.88.2 Constructor & Destructor Documentation . . . . .	361
7.88.3 Member Function Documentation . . . . .	361
7.89 CbcSimpleInteger Class Reference . . . . .	362
7.89.1 Detailed Description . . . . .	364
7.89.2 Constructor & Destructor Documentation . . . . .	364
7.89.3 Member Function Documentation . . . . .	364
7.89.4 Member Data Documentation . . . . .	366
7.90 CbcSimpleIntegerDynamicPseudoCost Class Reference . . . . .	366
7.90.1 Detailed Description . . . . .	371
7.90.2 Constructor & Destructor Documentation . . . . .	371
7.90.3 Member Function Documentation . . . . .	371
7.90.4 Member Data Documentation . . . . .	378
7.91 CbcSimpleIntegerPseudoCost Class Reference . . . . .	381
7.91.1 Detailed Description . . . . .	382
7.91.2 Constructor & Destructor Documentation . . . . .	382
7.91.3 Member Function Documentation . . . . .	382
7.91.4 Member Data Documentation . . . . .	384
7.92 CbcSolver Class Reference . . . . .	384
7.92.1 Detailed Description . . . . .	386
7.92.2 Constructor & Destructor Documentation . . . . .	386
7.92.3 Member Function Documentation . . . . .	387
7.93 CbcSolverUsefulData Struct Reference . . . . .	389

7.93.1 Detailed Description . . . . .	389
7.93.2 Member Data Documentation . . . . .	390
7.94 CbcSOS Class Reference . . . . .	390
7.94.1 Detailed Description . . . . .	392
7.94.2 Constructor & Destructor Documentation . . . . .	392
7.94.3 Member Function Documentation . . . . .	392
7.95 CbcSOSBranchingObject Class Reference . . . . .	394
7.95.1 Detailed Description . . . . .	395
7.95.2 Constructor & Destructor Documentation . . . . .	396
7.95.3 Member Function Documentation . . . . .	396
7.96 CbcStatistics Class Reference . . . . .	397
7.96.1 Detailed Description . . . . .	398
7.96.2 Constructor & Destructor Documentation . . . . .	398
7.96.3 Member Function Documentation . . . . .	398
7.96.4 Member Data Documentation . . . . .	399
7.97 CbcStopNow Class Reference . . . . .	400
7.97.1 Detailed Description . . . . .	401
7.97.2 Constructor & Destructor Documentation . . . . .	401
7.97.3 Member Function Documentation . . . . .	401
7.98 CbcStrategy Class Reference . . . . .	402
7.98.1 Detailed Description . . . . .	403
7.98.2 Constructor & Destructor Documentation . . . . .	403
7.98.3 Member Function Documentation . . . . .	403
7.98.4 Member Data Documentation . . . . .	405
7.99 CbcStrategyDefault Class Reference . . . . .	405
7.99.1 Detailed Description . . . . .	406
7.99.2 Constructor & Destructor Documentation . . . . .	407
7.99.3 Member Function Documentation . . . . .	407
7.99.4 Member Data Documentation . . . . .	408
7.100 CbcStrategyDefaultSubTree Class Reference . . . . .	408
7.100.1 Detailed Description . . . . .	409
7.100.2 Constructor & Destructor Documentation . . . . .	409
7.100.3 Member Function Documentation . . . . .	409
7.100.4 Member Data Documentation . . . . .	410
7.101 CbcStrategyNull Class Reference . . . . .	410
7.101.1 Detailed Description . . . . .	411
7.101.2 Constructor & Destructor Documentation . . . . .	411

7.101.3 Member Function Documentation . . . . .	411
7.102CbcStrongInfo Struct Reference . . . . .	412
7.102.1 Detailed Description . . . . .	412
7.102.2 Member Data Documentation . . . . .	413
7.103CbcThread Class Reference . . . . .	414
7.103.1 Detailed Description . . . . .	414
7.103.2 Constructor & Destructor Documentation . . . . .	414
7.104CbcTree Class Reference . . . . .	414
7.104.1 Detailed Description . . . . .	417
7.104.2 Constructor & Destructor Documentation . . . . .	417
7.104.3 Member Function Documentation . . . . .	418
7.104.4 Member Data Documentation . . . . .	421
7.105CbcTreeLocal Class Reference . . . . .	422
7.105.1 Detailed Description . . . . .	423
7.105.2 Constructor & Destructor Documentation . . . . .	423
7.105.3 Member Function Documentation . . . . .	423
7.106CbcTreeVariable Class Reference . . . . .	425
7.106.1 Detailed Description . . . . .	426
7.106.2 Constructor & Destructor Documentation . . . . .	426
7.106.3 Member Function Documentation . . . . .	426
7.107CbcUser Class Reference . . . . .	428
7.107.1 Detailed Description . . . . .	429
7.107.2 Constructor & Destructor Documentation . . . . .	430
7.107.3 Member Function Documentation . . . . .	430
7.107.4 Member Data Documentation . . . . .	431
7.108CglTemporary Class Reference . . . . .	431
7.108.1 Detailed Description . . . . .	432
7.108.2 Constructor & Destructor Documentation . . . . .	432
7.108.3 Member Function Documentation . . . . .	432
7.109CbcGenCtlBlk::chooseStrongCtl_struct Struct Reference . . . . .	433
7.109.1 Detailed Description . . . . .	433
7.109.2 Member Data Documentation . . . . .	433
7.110ClpAmplObjective Class Reference . . . . .	434
7.110.1 Detailed Description . . . . .	435
7.110.2 Constructor & Destructor Documentation . . . . .	435
7.110.3 Member Function Documentation . . . . .	435
7.111ClpConstraintAmpl Class Reference . . . . .	436

7.111.1 Detailed Description . . . . .	437
7.111.2 Constructor & Destructor Documentation . . . . .	437
7.111.3 Member Function Documentation . . . . .	438
7.112CoinHashLink Struct Reference . . . . .	439
7.112.1 Detailed Description . . . . .	439
7.112.2 Member Data Documentation . . . . .	439
7.113CbcGenCtlBlk::debugSolInfo_struct Struct Reference . . . . .	439
7.113.1 Detailed Description . . . . .	440
7.113.2 Member Data Documentation . . . . .	440
7.114CbcGenCtlBlk::djFixCtl_struct Struct Reference . . . . .	440
7.114.1 Detailed Description . . . . .	440
7.114.2 Member Data Documentation . . . . .	440
7.115CbcGenCtlBlk::genParamsInfo_struct Struct Reference . . . . .	441
7.115.1 Detailed Description . . . . .	441
7.115.2 Member Data Documentation . . . . .	441
7.116OsiBiLinear Class Reference . . . . .	441
7.116.1 Detailed Description . . . . .	445
7.116.2 Constructor & Destructor Documentation . . . . .	445
7.116.3 Member Function Documentation . . . . .	445
7.116.4 Member Data Documentation . . . . .	449
7.117OsiBiLinearBranchingObject Class Reference . . . . .	452
7.117.1 Detailed Description . . . . .	452
7.117.2 Constructor & Destructor Documentation . . . . .	452
7.117.3 Member Function Documentation . . . . .	452
7.118OsiBiLinearEquality Class Reference . . . . .	453
7.118.1 Detailed Description . . . . .	454
7.118.2 Constructor & Destructor Documentation . . . . .	454
7.118.3 Member Function Documentation . . . . .	454
7.119OsiCbcSolverInterface Class Reference . . . . .	455
7.119.1 Detailed Description . . . . .	461
7.119.2 Constructor & Destructor Documentation . . . . .	461
7.119.3 Member Function Documentation . . . . .	462
7.119.4 Friends And Related Function Documentation . . . . .	473
7.119.5 Member Data Documentation . . . . .	473
7.120OsiChooseStrongSubset Class Reference . . . . .	473
7.120.1 Detailed Description . . . . .	474
7.120.2 Constructor & Destructor Documentation . . . . .	474



7.120.3 Member Function Documentation . . . . .	475
7.120.4 Member Data Documentation . . . . .	475
7.121OsiLink Class Reference . . . . .	476
7.121.1 Detailed Description . . . . .	476
7.121.2 Constructor & Destructor Documentation . . . . .	477
7.121.3 Member Function Documentation . . . . .	477
7.122OsiLinkBranchingObject Class Reference . . . . .	478
7.122.1 Detailed Description . . . . .	478
7.122.2 Constructor & Destructor Documentation . . . . .	478
7.122.3 Member Function Documentation . . . . .	478
7.123OsiLinkedBound Class Reference . . . . .	479
7.123.1 Detailed Description . . . . .	480
7.123.2 Constructor & Destructor Documentation . . . . .	480
7.123.3 Member Function Documentation . . . . .	480
7.124OsiOldLink Class Reference . . . . .	480
7.124.1 Detailed Description . . . . .	481
7.124.2 Constructor & Destructor Documentation . . . . .	481
7.124.3 Member Function Documentation . . . . .	482
7.125OsiOldLinkBranchingObject Class Reference . . . . .	483
7.125.1 Detailed Description . . . . .	483
7.125.2 Constructor & Destructor Documentation . . . . .	484
7.125.3 Member Function Documentation . . . . .	484
7.126OsiOneLink Class Reference . . . . .	484
7.126.1 Detailed Description . . . . .	485
7.126.2 Constructor & Destructor Documentation . . . . .	485
7.126.3 Member Function Documentation . . . . .	485
7.126.4 Member Data Documentation . . . . .	485
7.127CbcGenCtlBlk::osiParamsInfo_struct Struct Reference . . . . .	486
7.127.1 Detailed Description . . . . .	486
7.127.2 Member Data Documentation . . . . .	486
7.128OsiSimpleFixedInteger Class Reference . . . . .	486
7.128.1 Detailed Description . . . . .	487
7.128.2 Constructor & Destructor Documentation . . . . .	487
7.128.3 Member Function Documentation . . . . .	488
7.129OsiSolverLinearizedQuadratic Class Reference . . . . .	488
7.129.1 Detailed Description . . . . .	489
7.129.2 Constructor & Destructor Documentation . . . . .	489

7.129.3 Member Function Documentation . . . . .	490
7.129.4 Member Data Documentation . . . . .	490
7.130OsiSolverLink Class Reference . . . . .	491
7.130.1 Detailed Description . . . . .	494
7.130.2 Constructor & Destructor Documentation . . . . .	494
7.130.3 Member Function Documentation . . . . .	495
7.130.4 Member Data Documentation . . . . .	498
7.131OsiUsesBiLinear Class Reference . . . . .	500
7.131.1 Detailed Description . . . . .	502
7.131.2 Constructor & Destructor Documentation . . . . .	502
7.131.3 Member Function Documentation . . . . .	502
7.131.4 Member Data Documentation . . . . .	503
7.132PseudoReducedCost Struct Reference . . . . .	503
7.132.1 Detailed Description . . . . .	503
7.132.2 Member Data Documentation . . . . .	503
<b>8 File Documentation</b>	<b>504</b>
8.1 /home/ted/COIN/trunk/Cbc/src/Cbc_ampl.h File Reference . . . . .	504
8.1.1 Function Documentation . . . . .	504
8.2 /home/ted/COIN/trunk/Cbc/src/Cbc_C_Interface.h File Reference . . . . .	504
8.2.1 Function Documentation . . . . .	505
8.2.2 Variable Documentation . . . . .	506
8.3 /home/ted/COIN/trunk/Cbc/src/CbcBranchActual.hpp File Reference . . . . .	509
8.4 /home/ted/COIN/trunk/Cbc/src/CbcBranchAllDifferent.hpp File Reference . . . . .	509
8.5 /home/ted/COIN/trunk/Cbc/src/CbcBranchBase.hpp File Reference . . . . .	510
8.5.1 Enumeration Type Documentation . . . . .	510
8.5.2 Function Documentation . . . . .	510
8.6 /home/ted/COIN/trunk/Cbc/src/CbcBranchCut.hpp File Reference . . . . .	511
8.7 /home/ted/COIN/trunk/Cbc/src/CbcBranchDecision.hpp File Reference . . . . .	511
8.8 /home/ted/COIN/trunk/Cbc/src/CbcBranchDefaultDecision.hpp File Reference . . . . .	511
8.9 /home/ted/COIN/trunk/Cbc/src/CbcBranchDynamic.hpp File Reference . . . . .	511
8.10 /home/ted/COIN/trunk/Cbc/src/CbcBranchingObject.hpp File Reference . . . . .	512
8.10.1 Enumeration Type Documentation . . . . .	512
8.11 /home/ted/COIN/trunk/Cbc/src/CbcBranchLotsize.hpp File Reference . . . . .	512
8.12 /home/ted/COIN/trunk/Cbc/src/CbcBranchToFixLots.hpp File Reference . . . . .	513
8.13 /home/ted/COIN/trunk/Cbc/src/CbcCliques.hpp File Reference . . . . .	513
8.14 /home/ted/COIN/trunk/Cbc/src/CbcCompare.hpp File Reference . . . . .	513

8.15	/home/ted/COIN/trunk/Cbc/src/CbcCompareActual.hpp File Reference	513
8.16	/home/ted/COIN/trunk/Cbc/src/CbcCompareBase.hpp File Reference	514
8.17	/home/ted/COIN/trunk/Cbc/src/CbcCompareDefault.hpp File Reference	514
8.18	/home/ted/COIN/trunk/Cbc/src/CbcCompareDepth.hpp File Reference	514
8.19	/home/ted/COIN/trunk/Cbc/src/CbcCompareEstimate.hpp File Reference	514
8.20	/home/ted/COIN/trunk/Cbc/src/CbcCompareObjective.hpp File Reference	514
8.21	/home/ted/COIN/trunk/Cbc/src/CbcConfig.h File Reference	515
8.22	/home/ted/COIN/trunk/Cbc/src/CbcConsequence.hpp File Reference	515
8.23	/home/ted/COIN/trunk/Cbc/src/CbcCountRowCut.hpp File Reference	515
8.24	/home/ted/COIN/trunk/Cbc/src/CbcCutGenerator.hpp File Reference	515
8.24.1	Macro Definition Documentation	516
8.25	/home/ted/COIN/trunk/Cbc/src/CbcCutModifier.hpp File Reference	516
8.26	/home/ted/COIN/trunk/Cbc/src/CbcCutSubsetModifier.hpp File Reference	516
8.27	/home/ted/COIN/trunk/Cbc/src/CbcDummyBranchingObject.hpp File Reference	516
8.28	/home/ted/COIN/trunk/Cbc/src/CbcEventHandler.hpp File Reference	517
8.28.1	Detailed Description	517
8.29	/home/ted/COIN/trunk/Cbc/src/CbcFathom.hpp File Reference	517
8.30	/home/ted/COIN/trunk/Cbc/src/CbcFathomDynamicProgramming.hpp File Reference	518
8.31	/home/ted/COIN/trunk/Cbc/src/CbcFeasibilityBase.hpp File Reference	518
8.32	/home/ted/COIN/trunk/Cbc/src/CbcFixVariable.hpp File Reference	518
8.33	/home/ted/COIN/trunk/Cbc/src/CbcFollowOn.hpp File Reference	518
8.34	/home/ted/COIN/trunk/Cbc/src/CbcFullNodeInfo.hpp File Reference	519
8.35	/home/ted/COIN/trunk/Cbc/src/CbcGenCbcParam.hpp File Reference	519
8.36	/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp File Reference	519
8.36.1	Macro Definition Documentation	521
8.37	/home/ted/COIN/trunk/Cbc/src/CbcGeneral.hpp File Reference	521
8.38	/home/ted/COIN/trunk/Cbc/src/CbcGeneralDepth.hpp File Reference	521
8.39	/home/ted/COIN/trunk/Cbc/src/CbcGenMessages.hpp File Reference	521
8.39.1	Detailed Description	521
8.39.2	Enumeration Type Documentation	521
8.40	/home/ted/COIN/trunk/Cbc/src/CbcGenOsiParam.hpp File Reference	522
8.41	/home/ted/COIN/trunk/Cbc/src/CbcGenParam.hpp File Reference	522
8.42	/home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp File Reference	523
8.43	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDINS.hpp File Reference	523
8.44	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDive.hpp File Reference	524
8.45	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveCoefficient.hpp File Reference	524
8.46	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveFractional.hpp File Reference	524

8.47	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveGuided.hpp File Reference	524
8.48	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveLineSearch.hpp File Reference	525
8.49	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDivePseudoCost.hpp File Reference	525
8.50	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveVectorLength.hpp File Reference	525
8.51	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDW.hpp File Reference	525
8.52	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicFPump.hpp File Reference	525
8.53	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicGreedy.hpp File Reference	526
8.54	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicLocal.hpp File Reference	526
8.55	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicPivotAndFix.hpp File Reference	526
8.56	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRandRound.hpp File Reference	526
8.57	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRENS.hpp File Reference	527
8.58	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRINS.hpp File Reference	527
8.59	/home/ted/COIN/trunk/Cbc/src/CbcHeuristicVND.hpp File Reference	527
8.60	/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp File Reference	527
8.60.1	Function Documentation	528
8.61	/home/ted/COIN/trunk/Cbc/src/CbcMessage.hpp File Reference	529
8.61.1	Enumeration Type Documentation	529
8.62	/home/ted/COIN/trunk/Cbc/src/CbcMipStartIO.hpp File Reference	531
8.62.1	Function Documentation	531
8.63	/home/ted/COIN/trunk/Cbc/src/CbcModel.hpp File Reference	531
8.63.1	Function Documentation	532
8.64	/home/ted/COIN/trunk/Cbc/src/CbcNode.hpp File Reference	533
8.65	/home/ted/COIN/trunk/Cbc/src/CbcNodeInfo.hpp File Reference	533
8.66	/home/ted/COIN/trunk/Cbc/src/CbcNWay.hpp File Reference	533
8.67	/home/ted/COIN/trunk/Cbc/src/CbcObject.hpp File Reference	533
8.68	/home/ted/COIN/trunk/Cbc/src/CbcObjectUpdateData.hpp File Reference	534
8.69	/home/ted/COIN/trunk/Cbc/src/CbcParam.hpp File Reference	534
8.69.1	Enumeration Type Documentation	535
8.70	/home/ted/COIN/trunk/Cbc/src/CbcPartialNodeInfo.hpp File Reference	538
8.71	/home/ted/COIN/trunk/Cbc/src/CbcSimpleInteger.hpp File Reference	538
8.72	/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerDynamicPseudoCost.hpp File Reference	539
8.72.1	Macro Definition Documentation	539
8.73	/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerPseudoCost.hpp File Reference	540
8.74	/home/ted/COIN/trunk/Cbc/src/CbcSolver.hpp File Reference	540
8.74.1	Detailed Description	540
8.75	/home/ted/COIN/trunk/Cbc/src/CbcSolverAnalyze.hpp File Reference	541
8.75.1	Detailed Description	541

8.75.2	Function Documentation	541
8.76	/home/ted/COIN/trunk/Cbc/src/CbcSolverExpandKnapsack.hpp File Reference	541
8.76.1	Detailed Description	541
8.76.2	Function Documentation	541
8.77	/home/ted/COIN/trunk/Cbc/src/CbcSolverHeuristics.hpp File Reference	542
8.77.1	Detailed Description	542
8.77.2	Function Documentation	542
8.78	/home/ted/COIN/trunk/Cbc/src/CbcSOS.hpp File Reference	542
8.79	/home/ted/COIN/trunk/Cbc/src/CbcStatistics.hpp File Reference	542
8.80	/home/ted/COIN/trunk/Cbc/src/CbcStrategy.hpp File Reference	543
8.81	/home/ted/COIN/trunk/Cbc/src/CbcSubProblem.hpp File Reference	543
8.82	/home/ted/COIN/trunk/Cbc/src/CbcThread.hpp File Reference	543
8.83	/home/ted/COIN/trunk/Cbc/src/CbcTree.hpp File Reference	543
8.84	/home/ted/COIN/trunk/Cbc/src/CbcTreeLocal.hpp File Reference	544
8.85	/home/ted/COIN/trunk/Cbc/src/ClpAmplObjective.hpp File Reference	544
8.86	/home/ted/COIN/trunk/Cbc/src/ClpConstraintAmpl.hpp File Reference	544
8.87	/home/ted/COIN/trunk/Cbc/src/config_cbc_default.h File Reference	544
8.87.1	Macro Definition Documentation	545
8.88	/home/ted/COIN/trunk/Cbc/src/config_default.h File Reference	545
8.88.1	Macro Definition Documentation	545
8.89	/home/ted/COIN/trunk/Cbc/src/OsiCbc/OsiCbcSolverInterface.hpp File Reference	546
8.89.1	Function Documentation	546
8.89.2	Variable Documentation	546

## Index

548

## 1 Todo List

### Class **CbcCutGenerator**

Add a pointer to function member which will allow a client to install their own decision algorithm to decide whether or not to call the CGL `generateCuts` method. Create a default decision method that looks at the builtin criteria.

It strikes me as not good that `generateCuts` contains code specific to individual CGL algorithms. Another set of pointer to function members, so that the client can specify the cut generation method as well as pre- and post-generation methods? Taken a bit further, should this class contain a bunch of pointer to function members, one for each of the places where the cut generator might be referenced? Initialization, root node, search tree node, discovery of solution, and termination all come to mind. Initialization and termination would also be useful for instrumenting cbc.

### Class **CbcFullNodeInfo**

While there's no explicit statement, the code often makes the implicit assumption that an `CbcFullNodeInfo` structure will appear only at the root node of the search tree. Things will break if this assumption is violated.

**Member [CbcModel::addCuts1](#) ([CbcNode](#) \*node, [CoinWarmStartBasis](#) \*&lastws)**

[addCuts1\(\)](#) is called in contexts where it's known in advance that all that's desired is to determine a list of cuts and do the bookkeeping (adjust the reference counts). The work of installing bounds and building a basis goes to waste.

**Member [CbcModel::integerPresolve](#) (bool weak=false)**

It remains to work out the cleanest way of getting a solution to the original problem at the end. So this is very preliminary.

**Member [CbcNodeInfo::numberBranchesLeft\\_](#)**

There seems to be redundancy between this field and [CbcBranchingObject::numberBranchesLeft\\_](#). It'd be good to sort out if both are necessary.

**Class [CbcStrongInfo](#)**

The notion that all branches are binary (two arms) is wired into the implementation of [CbcObject](#), [CbcBranchingObject](#), and [CbcBranchDecision](#). Changing this will require a moderate amount of recoding.

## 2 Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">CbcCbcParamUtils</a>	20
<a href="#">CbcGenParamUtils</a>	21
<a href="#">CbcOsiParamUtils</a>	22

## 3 Hierarchical Index

### 3.1 Class Hierarchy

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

<a href="#">std::allocator&lt; T &gt;</a>	
<a href="#">ampl_info</a>	22
<a href="#">std::array&lt; T &gt;</a>	
<a href="#">std::auto_ptr&lt; T &gt;</a>	
<a href="#">CbcGenCtlBlk::babState_struct</a>	26
<a href="#">std::basic_string&lt; Char &gt;</a>	
<a href="#">std::string</a>	
<a href="#">std::wstring</a>	
<a href="#">std::basic_string&lt; char &gt;</a>	
<a href="#">std::basic_string&lt; wchar_t &gt;</a>	
<a href="#">std::bitset&lt; Bits &gt;</a>	
<a href="#">CbcBaseModel</a>	27
<a href="#">CbcBranchDecision</a>	32
<a href="#">CbcBranchDefaultDecision</a>	36

<b>CbcBranchDynamicDecision</b>	<b>38</b>
<b>CbcCompare</b>	<b>60</b>
<b>CbcCompareBase</b>	<b>61</b>
<b>CbcCompareDefault</b>	<b>64</b>
<b>CbcCompareDepth</b>	<b>69</b>
<b>CbcCompareEstimate</b>	<b>70</b>
<b>CbcCompareObjective</b>	<b>71</b>
<b>CbcConsequence</b>	<b>73</b>
<b>CbcFixVariable</b>	<b>113</b>
<b>CbcCutGenerator</b>	<b>79</b>
<b>CbcCutModifier</b>	<b>90</b>
<b>CbcCutSubsetModifier</b>	<b>91</b>
<b>CbcEventHandler</b>	<b>99</b>
<b>CbcFathom</b>	<b>103</b>
<b>CbcFathomDynamicProgramming</b>	<b>105</b>
<b>CbcFeasibilityBase</b>	<b>110</b>
<b>CbcGenCtlBlk</b>	<b>121</b>
<b>CbcHeuristic</b>	<b>143</b>
<b>CbcHeuristicCrossover</b>	<b>155</b>
<b>CbcHeuristicDINS</b>	<b>157</b>
<b>CbcHeuristicDive</b>	<b>161</b>
<b>CbcHeuristicDiveCoefficient</b>	<b>166</b>
<b>CbcHeuristicDiveFractional</b>	<b>168</b>
<b>CbcHeuristicDiveGuided</b>	<b>170</b>
<b>CbcHeuristicDiveLineSearch</b>	<b>171</b>
<b>CbcHeuristicDivePseudoCost</b>	<b>173</b>
<b>CbcHeuristicDiveVectorLength</b>	<b>175</b>
<b>CbcHeuristicDW</b>	<b>176</b>
<b>CbcHeuristicDynamic3</b>	<b>190</b>
<b>CbcHeuristicFPump</b>	<b>192</b>

<b>CbcHeuristicGreedyCover</b>	<b>201</b>
<b>CbcHeuristicGreedyEquality</b>	<b>203</b>
<b>CbcHeuristicGreedySOS</b>	<b>207</b>
<b>CbcHeuristicJustOne</b>	<b>210</b>
<b>CbcHeuristicLocal</b>	<b>212</b>
<b>CbcHeuristicNaive</b>	<b>215</b>
<b>CbcHeuristicPartial</b>	<b>219</b>
<b>CbcHeuristicPivotAndFix</b>	<b>221</b>
<b>CbcHeuristicProximity</b>	<b>223</b>
<b>CbcHeuristicRandRound</b>	<b>225</b>
<b>CbcHeuristicRENS</b>	<b>227</b>
<b>CbcHeuristicRINS</b>	<b>229</b>
<b>CbcHeuristicVND</b>	<b>233</b>
<b>CbcRounding</b>	<b>356</b>
<b>CbcSerendipity</b>	<b>360</b>
<b>CbcHeuristicNode</b>	<b>217</b>
<b>CbcHeuristicNodeList</b>	<b>218</b>
<b>CbcModel</b>	<b>255</b>
<b>CbcNodeInfo</b>	<b>318</b>
<b>CbcFullNodeInfo</b>	<b>118</b>
<b>CbcPartialNodeInfo</b>	<b>354</b>
<b>CbcObjectUpdateData</b>	<b>339</b>
<b>CbcParam</b>	<b>348</b>
<b>CbcGenCtIBlk::cbcParamsInfo_struct</b>	<b>353</b>
<b>CbcRowCuts</b>	<b>359</b>
<b>CbcSolver</b>	<b>384</b>
<b>CbcSolverUsefulData</b>	<b>389</b>
<b>CbcStatistics</b>	<b>397</b>
<b>CbcStopNow</b>	<b>400</b>
<b>CbcStrategy</b>	<b>402</b>



<b>CbcStrategyDefault</b>	<a href="#">405</a>
<b>CbcStrategyDefaultSubTree</b>	<a href="#">408</a>
<b>CbcStrategyNull</b>	<a href="#">410</a>
<b>CbcStrongInfo</b>	<a href="#">412</a>
<b>CbcThread</b>	<a href="#">414</a>
<b>CbcTree</b>	<a href="#">414</a>
<b>CbcTreeLocal</b>	<a href="#">422</a>
<b>CbcTreeVariable</b>	<a href="#">425</a>
<b>CbcUser</b>	<a href="#">428</a>
CglStored	
<b>CglTemporary</b>	<a href="#">431</a>
<b>CbcGenCtlBlk::chooseStrongCtl_struct</b>	<a href="#">433</a>
ClpConstraint	
<b>ClpConstraintAmpl</b>	<a href="#">436</a>
ClpObjective	
<b>ClpAmplObjective</b>	<a href="#">434</a>
<b>CoinHashLink</b>	<a href="#">439</a>
CoinMessages	
<b>CbcMessage</b>	<a href="#">254</a>
CoinParam	
<b>CbcCbcParam</b>	<a href="#">50</a>
<b>CbcGenParam</b>	<a href="#">139</a>
<b>CbcOsiParam</b>	<a href="#">341</a>
CoinTreeNode	
<b>CbcNode</b>	<a href="#">310</a>
std::complex	
std::list< T >::const_iterator	
std::map< K, T >::const_iterator	
std::forward_list< T >::const_iterator	
std::unordered_map< K, T >::const_iterator	
std::basic_string< Char >::const_iterator	
std::multimap< K, T >::const_iterator	
std::unordered_multimap< K, T >::const_iterator	
std::set< K >::const_iterator	
std::string::const_iterator	
std::unordered_set< K >::const_iterator	
std::multiset< K >::const_iterator	
std::wstring::const_iterator	
std::unordered_multiset< K >::const_iterator	

```

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

```

### **CbcGenCtlBlk::debugSolInfo\_struct**

439

```
std::deque< T >
```

### **CbcGenCtlBlk::djFixCtl\_struct**

440

```

std::error_category
std::error_code
std::error_condition
std::exception
    std::bad_alloc
    std::bad_cast
    std::bad_exception
    std::bad_typeid
    std::ios_base::failure
    std::logic_error
        std::domain_error
        std::invalid_argument
        std::length_error
        std::out_of_range
    std::runtime_error
        std::overflow_error
        std::range_error
        std::underflow_error
std::forward_list< T >

```

### **CbcGenCtlBlk::genParamsInfo\_struct**

441

```

std::ios_base
    basic_ios< char >
    basic_ios< wchar_t >
    std::basic_ios
        basic_istream< char >
        basic_istream< wchar_t >
        basic_ostream< char >
        basic_ostream< wchar_t >
    std::basic_istream
        basic_ifstream< char >
        basic_ifstream< wchar_t >
        basic_iostream< char >

```

```
basic_iostream< wchar_t >
basic_istream< char >
basic_istream< wchar_t >
std::basic_ifstream
    std::ifstream
    std::wifstream
std::basic_ostream
    basic_fstream< char >
    basic_fstream< wchar_t >
    basic_stringstream< char >
    basic_stringstream< wchar_t >
    std::basic_fstream
        std::fstream
        std::wfstream
    std::basic_stringstream
        std::stringstream
        std::wstringstream
std::basic_istream
    std::istream
    std::wistream
std::basic_ostream
    basic_iostream< char >
    basic_iostream< wchar_t >
    basic_ofstream< char >
    basic_ofstream< wchar_t >
    basic_ostringstream< char >
    basic_ostringstream< wchar_t >
    std::basic_iostream
    std::basic_ofstream
        std::ofstream
        std::wofstream
    std::basic_ostringstream
        std::ostringstream
        std::wostringstream
    std::ostream
    std::wostream
std::ios
std::wios
std::unordered_set< K >::iterator
std::set< K >::iterator
std::list< T >::iterator
std::map< K, T >::iterator
std::forward_list< T >::iterator
std::unordered_map< K, T >::iterator
std::multimap< K, T >::iterator
std::unordered_multimap< K, T >::iterator
std::string::iterator
std::wstring::iterator
std::multiset< K >::iterator
std::unordered_multiset< K >::iterator
std::basic_string< Char >::iterator
std::vector< T >::iterator
```

std::deque< T >::iterator	
std::list< T >	
std::map< K, T >	
std::multimap< K, T >	
std::multiset< K >	
OsiBranchingObject	
<b>CbcBranchingObject</b>	<b>40</b>
<b>CbcCliqueBranchingObject</b>	<b>58</b>
<b>CbcCutBranchingObject</b>	<b>76</b>
<b>CbcDummyBranchingObject</b>	<b>93</b>
<b>CbcFixingBranchingObject</b>	<b>111</b>
<b>CbcIntegerBranchingObject</b>	<b>239</b>
<b>CbcDynamicPseudoCostBranchingObject</b>	<b>95</b>
<b>CbcIntegerPseudoCostBranchingObject</b>	<b>242</b>
<b>CbcLongCliqueBranchingObject</b>	<b>245</b>
<b>CbcLotsizeBranchingObject</b>	<b>251</b>
<b>CbcNWayBranchingObject</b>	<b>329</b>
<b>CbcSOSBranchingObject</b>	<b>394</b>
OsiChooseStrong	
<b>OsiChooseStrongSubset</b>	<b>473</b>
OsiClpSolverInterface	
<b>CbcOsiSolver</b>	<b>346</b>
<b>OsiSolverLink</b>	<b>491</b>
<b>OsiSolverLinearizedQuadratic</b>	<b>488</b>
<b>OsiLinkedBound</b>	<b>479</b>
OsiObject	
<b>CbcObject</b>	<b>332</b>
<b>CbcBranchCut</b>	<b>30</b>
<b>CbcBranchAllDifferent</b>	<b>28</b>
<b>CbcBranchToFixLots</b>	<b>47</b>
<b>CbcClique</b>	<b>54</b>
<b>CbcFollowOn</b>	<b>115</b>
<b>CbcGeneral</b>	<b>137</b>

<b>CbcIdiotBranch</b>	<b>236</b>
<b>CbcLotsize</b>	<b>247</b>
<b>CbcNWay</b>	<b>326</b>
<b>CbcSimpleInteger</b>	<b>362</b>
<b>CbcSimpleIntegerDynamicPseudoCost</b>	<b>366</b>
<b>CbcSimpleIntegerPseudoCost</b>	<b>381</b>
<b>CbcSOS</b>	<b>390</b>
OsiObject2	
<b>OsiBiLinear</b>	<b>441</b>
<b>OsiBiLinearEquality</b>	<b>453</b>
<b>OsiOneLink</b>	<b>484</b>
<b>CbcGenCtIBlk::osiParamsInfo_struct</b>	<b>486</b>
OsiRowCut	
<b>CbcCountRowCut</b>	<b>74</b>
OsiSimpleInteger	
<b>OsiSimpleFixedInteger</b>	<b>486</b>
<b>OsiUsesBiLinear</b>	<b>500</b>
OsiSolverInterface	
<b>OsiCbcSolverInterface</b>	<b>455</b>
OsiSOS	
<b>OsiLink</b>	<b>476</b>
<b>OsiOldLink</b>	<b>480</b>
OsiSOSBranchingObject	
<b>OsiOldLinkBranchingObject</b>	<b>483</b>
OsiTwoWayBranchingObject	
<b>OsiBiLinearBranchingObject</b>	<b>452</b>
<b>OsiLinkBranchingObject</b>	<b>478</b>
std::priority_queue< T >	
<b>PseudoReducedCost</b>	<b>503</b>
std::queue< T >	
std::string::reverse_iterator	
std::forward_list< T >::reverse_iterator	
std::list< T >::reverse_iterator	
std::map< K, T >::reverse_iterator	
std::unordered_map< K, T >::reverse_iterator	
std::deque< T >::reverse_iterator	
std::basic_string< Char >::reverse_iterator	

```

std::vector< T >::reverse_iterator
std::set< K >::reverse_iterator
std::unordered_multiset< K >::reverse_iterator
std::wstring::reverse_iterator
std::multimap< K, T >::reverse_iterator
std::multiset< K >::reverse_iterator
std::unordered_multimap< K, T >::reverse_iterator
std::unordered_set< K >::reverse_iterator
std::set< K >
std::smart_ptr< T >
std::stack< T >
std::system_error
std::thread
std::unique_ptr< T >
std::unordered_map< K, T >
std::unordered_multimap< K, T >
std::unordered_multiset< K >
std::unordered_set< K >
std::valarray< T >
std::vector< T >
std::vector< bool >
std::vector< CbcHeuristicNode * >
std::vector< CbcNode * >
std::vector< double >
std::vector< int >
std::vector< std::pair< std::string, double > >
std::vector< std::string >
std::weak_ptr< T >
K
T

```

## 4 Class Index

### 4.1 Class List

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

<a href="#"><b>ampl_info</b></a>	<b>22</b>
<a href="#"><b>CbcGenCtIBlk::babState_struct</b></a> State of branch-and-cut	<b>26</b>
<a href="#"><b>CbcBaseModel</b></a> Base model	<b>27</b>
<a href="#"><b>CbcBranchAllDifferent</b></a> Define a branch class that branches so that it is only satisfied if all members have different values So cut is $x \leq y-1$ or $x \geq y+1$	<b>28</b>
<a href="#"><b>CbcBranchCut</b></a> Define a cut branching class	<b>30</b>
<a href="#"><b>CbcBranchDecision</b></a>	<b>32</b>

<a href="#">CbcBranchDefaultDecision</a>	
Branching decision default class	36
<a href="#">CbcBranchDynamicDecision</a>	
Branching decision dynamic class	38
<a href="#">CbcBranchingObject</a>	
Abstract branching object base class Now just difference with OsiBranchingObject	40
<a href="#">CbcBranchToFixLots</a>	
Define a branch class that branches so that one way variables are fixed while the other way cuts off that solution	47
<a href="#">CbcCbcParam</a>	
Class for control parameters that act on a <a href="#">CbcModel</a> object	50
<a href="#">CbcClique</a>	
Branching object for cliques	54
<a href="#">CbcCliqueBranchingObject</a>	
Branching object for unordered cliques	58
<a href="#">CbcCompare</a>	60
<a href="#">CbcCompareBase</a>	61
<a href="#">CbcCompareDefault</a>	64
<a href="#">CbcCompareDepth</a>	69
<a href="#">CbcCompareEstimate</a>	70
<a href="#">CbcCompareObjective</a>	71
<a href="#">CbcConsequence</a>	
Abstract base class for consequent bounds	73
<a href="#">CbcCountRowCut</a>	
OsiRowCut augmented with bookkeeping	74
<a href="#">CbcCutBranchingObject</a>	
Cut branching object	76
<a href="#">CbcCutGenerator</a>	
Interface between Cbc and Cut Generation Library	79
<a href="#">CbcCutModifier</a>	
Abstract cut modifier base class	90
<a href="#">CbcCutSubsetModifier</a>	
Simple cut modifier base class	91
<a href="#">CbcDummyBranchingObject</a>	
Dummy branching object	93
<a href="#">CbcDynamicPseudoCostBranchingObject</a>	
Simple branching object for an integer variable with pseudo costs	95

<a href="#"><b>CbcEventHandler</b></a>	
Base class for Cbc event handling	99
<a href="#"><b>CbcFathom</b></a>	
Fathom base class	103
<a href="#"><b>CbcFathomDynamicProgramming</b></a>	
FathomDynamicProgramming class	105
<a href="#"><b>CbcFeasibilityBase</b></a>	110
<a href="#"><b>CbcFixingBranchingObject</b></a>	
General Branching Object class	111
<a href="#"><b>CbcFixVariable</b></a>	
Class for consequent bounds	113
<a href="#"><b>CbcFollowOn</b></a>	
Define a follow on class	115
<a href="#"><b>CbcFullNodeInfo</b></a>	
Information required to recreate the subproblem at this node	118
<a href="#"><b>CbcGenCtlBlk</b></a>	121
<a href="#"><b>CbcGeneral</b></a>	
Define a catch all class	137
<a href="#"><b>CbcGenParam</b></a>	
Class for cbc-generic control parameters	139
<a href="#"><b>CbcHeuristic</b></a>	
Heuristic base class	143
<a href="#"><b>CbcHeuristicCrossover</b></a>	
Crossover Search class	155
<a href="#"><b>CbcHeuristicDINS</b></a>	157
<a href="#"><b>CbcHeuristicDive</b></a>	
Dive class	161
<a href="#"><b>CbcHeuristicDiveCoefficient</b></a>	
DiveCoefficient class	166
<a href="#"><b>CbcHeuristicDiveFractional</b></a>	
DiveFractional class	168
<a href="#"><b>CbcHeuristicDiveGuided</b></a>	
DiveGuided class	170
<a href="#"><b>CbcHeuristicDiveLineSearch</b></a>	
DiveLineSearch class	171
<a href="#"><b>CbcHeuristicDivePseudoCost</b></a>	
DivePseudoCost class	173



<a href="#"><b>CbcHeuristicDiveVectorLength</b></a>	
DiveVectorLength class	175
<a href="#"><b>CbcHeuristicDW</b></a>	
This is unlike the other heuristics in that it is very very compute intensive	176
<a href="#"><b>CbcHeuristicDynamic3</b></a>	
Heuristic - just picks up any good solution	190
<a href="#"><b>CbcHeuristicFPump</b></a>	
Feasibility Pump class	192
<a href="#"><b>CbcHeuristicGreedyCover</b></a>	
Greedy heuristic classes	201
<a href="#"><b>CbcHeuristicGreedyEquality</b></a>	203
<a href="#"><b>CbcHeuristicGreedySOS</b></a>	
Greedy heuristic for SOS and L rows (and positive elements)	207
<a href="#"><b>CbcHeuristicJustOne</b></a>	
Just One class - this chooses one at random	210
<a href="#"><b>CbcHeuristicLocal</b></a>	
LocalSearch class	212
<a href="#"><b>CbcHeuristicNaive</b></a>	
Naive class a) Fix all ints as close to zero as possible b) Fix all ints with nonzero costs and $<$ large to zero c) Put bounds round continuous and UIs and maximize	215
<a href="#"><b>CbcHeuristicNode</b></a>	
A class describing the branching decisions that were made to get to the node where a heuristic was invoked from	217
<a href="#"><b>CbcHeuristicNodeList</b></a>	218
<a href="#"><b>CbcHeuristicPartial</b></a>	
Partial solution class If user knows a partial solution this tries to get an integer solution it uses hotstart information	219
<a href="#"><b>CbcHeuristicPivotAndFix</b></a>	
LocalSearch class	221
<a href="#"><b>CbcHeuristicProximity</b></a>	223
<a href="#"><b>CbcHeuristicRandRound</b></a>	
LocalSearch class	225
<a href="#"><b>CbcHeuristicRENS</b></a>	
LocalSearch class	227
<a href="#"><b>CbcHeuristicRINS</b></a>	
LocalSearch class	229
<a href="#"><b>CbcHeuristicVND</b></a>	
LocalSearch class	233

<a href="#"><b>CbcIdiotBranch</b></a>	
Define an idiotic idea class	<b>236</b>
<a href="#"><b>CbcIntegerBranchingObject</b></a>	
Simple branching object for an integer variable	<b>239</b>
<a href="#"><b>CbcIntegerPseudoCostBranchingObject</b></a>	
Simple branching object for an integer variable with pseudo costs	<b>242</b>
<a href="#"><b>CbcLongCliqueBranchingObject</b></a>	
Unordered Clique Branching Object class	<b>245</b>
<a href="#"><b>CbcLotsize</b></a>	
Lotsize class	<b>247</b>
<a href="#"><b>CbcLotsizeBranchingObject</b></a>	
Lotsize branching object	<b>251</b>
<a href="#"><b>CbcMessage</b></a>	<b>254</b>
<a href="#"><b>CbcModel</b></a>	
Simple Branch and bound class	<b>255</b>
<a href="#"><b>CbcNode</b></a>	
Information required while the node is live	<b>310</b>
<a href="#"><b>CbcNodeInfo</b></a>	
Information required to recreate the subproblem at this node	<b>318</b>
<a href="#"><b>CbcNWay</b></a>	
Define an n-way class for variables	<b>326</b>
<a href="#"><b>CbcNWayBranchingObject</b></a>	
N way branching Object class	<b>329</b>
<a href="#"><b>CbcObject</b></a>	<b>332</b>
<a href="#"><b>CbcObjectUpdateData</b></a>	<b>339</b>
<a href="#"><b>CbcOsiParam</b></a>	
Class for control parameters that act on a OsiSolverInterface object	<b>341</b>
<a href="#"><b>CbcOsiSolver</b></a>	
This is for codes where solver needs to know about <a href="#"><b>CbcModel</b></a> Seems to provide only one value-added feature, a <a href="#"><b>CbcModel</b></a> object	<b>346</b>
<a href="#"><b>CbcParam</b></a>	
Very simple class for setting parameters	<b>348</b>
<a href="#"><b>CbcGenCtlBlk::cbcParamsInfo_struct</b></a>	
Start and end of <a href="#"><b>CbcModel</b></a> parameters in parameter vector	<b>353</b>
<a href="#"><b>CbcPartialNodeInfo</b></a>	
Holds information for recreating a subproblem by incremental change from the parent	<b>354</b>
<a href="#"><b>CbcRounding</b></a>	
Rounding class	<b>356</b>

<a href="#">CbcRowCuts</a>	359
<a href="#">CbcSerendipity</a> Heuristic - just picks up any good solution found by solver - see OsiBabSolver	360
<a href="#">CbcSimpleInteger</a> Define a single integer class	362
<a href="#">CbcSimpleIntegerDynamicPseudoCost</a> Define a single integer class but with dynamic pseudo costs	366
<a href="#">CbcSimpleIntegerPseudoCost</a> Define a single integer class but with pseudo costs	381
<a href="#">CbcSolver</a> This allows the use of the standalone solver in a flexible manner	384
<a href="#">CbcSolverUsefulData</a> Structure to hold useful arrays	389
<a href="#">CbcSOS</a> Branching object for Special Ordered Sets of type 1 and 2	390
<a href="#">CbcSOSBranchingObject</a> Branching object for Special ordered sets	394
<a href="#">CbcStatistics</a> For gathering statistics	397
<a href="#">CbcStopNow</a> Support the use of a call back class to decide whether to stop	400
<a href="#">CbcStrategy</a> Strategy base class	402
<a href="#">CbcStrategyDefault</a> Default class	405
<a href="#">CbcStrategyDefaultSubTree</a> Default class for sub trees	408
<a href="#">CbcStrategyNull</a> Null class	410
<a href="#">CbcStrongInfo</a> Abstract base class for 'objects'	412
<a href="#">CbcThread</a> A class to encapsulate thread stuff	414
<a href="#">CbcTree</a> Using MS heap implementation	414
<a href="#">CbcTreeLocal</a>	422
<a href="#">CbcTreeVariable</a>	425

<b>CbcUser</b>	
A class to allow the use of unknown user functionality	428
<b>CglTemporary</b>	
Stored Temporary Cut Generator Class - destroyed after first use	431
<b>CbcGenCtIBlk::chooseStrongCtl_struct</b>	
Control variables for a strong branching method	433
<b>ClpAmplObjective</b>	
Ampl Objective Class	434
<b>ClpConstraintAmpl</b>	
Ampl Constraint Class	436
<b>CoinHashLink</b>	
Really for Conflict cuts to - a) stop duplicates b) allow half baked cuts The whichRow_ field in OsiRowCut2 is used for a type 0 - normal 1 - processed cut 2 - unprocessed cut i.e	439
<b>CbcGenCtIBlk::debugSolInfo_struct</b>	
Array of primal variable values for debugging	439
<b>CbcGenCtIBlk::djFixCtl_struct</b>	
Control use of reduced cost fixing prior to B&C	440
<b>CbcGenCtIBlk::genParamsInfo_struct</b>	
Start and end of cbc-generic parameters in parameter vector	441
<b>OsiBiLinear</b>	
Define BiLinear objects	441
<b>OsiBiLinearBranchingObject</b>	
Branching object for BiLinear objects	452
<b>OsiBiLinearEquality</b>	
Define Continuous BiLinear objects for an == bound	453
<b>OsiCbcSolverInterface</b>	
Cbc Solver Interface	455
<b>OsiChooseStrongSubset</b>	
This class chooses a variable to branch on	473
<b>OsiLink</b>	
Define Special Linked Ordered Sets	476
<b>OsiLinkBranchingObject</b>	
Branching object for Linked ordered sets	478
<b>OsiLinkedBound</b>	
List of bounds which depend on other bounds	479
<b>OsiOldLink</b>	480
<b>OsiOldLinkBranchingObject</b>	
Branching object for Linked ordered sets	483

<a href="#">OsiOneLink</a>	
Define data for one link	484
<a href="#">CbcGenCtlBlk::osiParamsInfo_struct</a>	
Start and end of OsiSolverInterface parameters in parameter vector	486
<a href="#">OsiSimpleFixedInteger</a>	
Define a single integer class - but one where you keep branching until fixed even if satisfied	486
<a href="#">OsiSolverLinearizedQuadratic</a>	
This is to allow the user to replace initialSolve and resolve	488
<a href="#">OsiSolverLink</a>	
This is to allow the user to replace initialSolve and resolve This version changes coefficients	491
<a href="#">OsiUsesBiLinear</a>	
Define a single variable class which is involved with <a href="#">OsiBiLinear</a> objects	500
<a href="#">PseudoReducedCost</a>	503

## 5 File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">/home/ted/COIN/trunk/Cbc/src/Cbc_ampl.h</a>	504
<a href="#">/home/ted/COIN/trunk/Cbc/src/Cbc_C_Interface.h</a>	504
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchActual.hpp</a>	509
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchAllDifferent.hpp</a>	509
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchBase.hpp</a>	510
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchCut.hpp</a>	511
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchDecision.hpp</a>	511
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchDefaultDecision.hpp</a>	511
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchDynamic.hpp</a>	511
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchingObject.hpp</a>	512
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchLotsize.hpp</a>	512
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcBranchToFixLots.hpp</a>	513
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcClique.hpp</a>	513
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCompare.hpp</a>	513
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCompareActual.hpp</a>	513

<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCompareBase.hpp</a>	514
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCompareDefault.hpp</a>	514
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCompareDepth.hpp</a>	514
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCompareEstimate.hpp</a>	514
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCompareObjective.hpp</a>	514
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcConfig.h</a>	515
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcConsequence.hpp</a>	515
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCountRowCut.hpp</a>	515
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCutGenerator.hpp</a>	515
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCutModifier.hpp</a>	516
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcCutSubsetModifier.hpp</a>	516
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcDummyBranchingObject.hpp</a>	516
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcEventHandler.hpp</a>	
Event handling for cbc	517
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcFathom.hpp</a>	517
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcFathomDynamicProgramming.hpp</a>	518
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcFeasibilityBase.hpp</a>	518
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcFixVariable.hpp</a>	518
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcFollowOn.hpp</a>	518
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcFullNodeInfo.hpp</a>	519
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcGenCbcParam.hpp</a>	519
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp</a>	519
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcGeneral.hpp</a>	521
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcGeneralDepth.hpp</a>	521
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcGenMessages.hpp</a>	
This file contains the enum that defines symbolic names for for cbc-generic messages	521
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcGenOsiParam.hpp</a>	522
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcGenParam.hpp</a>	522
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp</a>	523
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDINS.hpp</a>	523
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDive.hpp</a>	524

/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveCoefficient.hpp	524
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveFractional.hpp	524
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveGuided.hpp	524
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveLineSearch.hpp	525
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDivePseudoCost.hpp	525
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveVectorLength.hpp	525
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDW.hpp	525
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicFPump.hpp	525
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicGreedy.hpp	526
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicLocal.hpp	526
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicPivotAndFix.hpp	526
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRandRound.hpp	526
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRENS.hpp	527
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRINS.hpp	527
/home/ted/COIN/trunk/Cbc/src/CbcHeuristicVND.hpp	527
/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp	527
/home/ted/COIN/trunk/Cbc/src/CbcMessage.hpp	529
/home/ted/COIN/trunk/Cbc/src/CbcMipStartIO.hpp	531
/home/ted/COIN/trunk/Cbc/src/CbcModel.hpp	531
/home/ted/COIN/trunk/Cbc/src/CbcNode.hpp	533
/home/ted/COIN/trunk/Cbc/src/CbcNodeInfo.hpp	533
/home/ted/COIN/trunk/Cbc/src/CbcNWay.hpp	533
/home/ted/COIN/trunk/Cbc/src/CbcObject.hpp	533
/home/ted/COIN/trunk/Cbc/src/CbcObjectUpdateData.hpp	534
/home/ted/COIN/trunk/Cbc/src/CbcParam.hpp	534
/home/ted/COIN/trunk/Cbc/src/CbcPartialNodeInfo.hpp	538
/home/ted/COIN/trunk/Cbc/src/CbcSimpleInteger.hpp	538
/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerDynamicPseudoCost.hpp	539
/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerPseudoCost.hpp	540

<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcSolver.hpp</a>	
Defines <a href="#">CbcSolver</a> , the proposed top-level class for the new-style cbc solver	540
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcSolverAnalyze.hpp</a>	
Look to see if a constraint is all-integer (variables & coeffs), or could be all integer	541
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcSolverExpandKnapsack.hpp</a>	
Expanding possibilities of $x*y$ , where $x*y$ are both integers, constructing a knapsack constraint	541
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcSolverHeuristics.hpp</a>	
Routines for doing heuristics	542
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcSOS.hpp</a>	542
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcStatistics.hpp</a>	542
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcStrategy.hpp</a>	543
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcSubProblem.hpp</a>	543
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcThread.hpp</a>	543
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcTree.hpp</a>	543
<a href="#">/home/ted/COIN/trunk/Cbc/src/CbcTreeLocal.hpp</a>	544
<a href="#">/home/ted/COIN/trunk/Cbc/src/ClpAmplObjective.hpp</a>	544
<a href="#">/home/ted/COIN/trunk/Cbc/src/ClpConstraintAmpl.hpp</a>	544
<a href="#">/home/ted/COIN/trunk/Cbc/src/config_cbc_default.h</a>	544
<a href="#">/home/ted/COIN/trunk/Cbc/src/config_default.h</a>	545
<a href="#">/home/ted/COIN/trunk/Cbc/src/OsiCbc/OsiCbcSolverInterface.hpp</a>	546

## 6 Namespace Documentation

### 6.1 CbcCbcParamUtils Namespace Reference

#### Functions

- void [addCbcCbcParams](#) (int &numParams, CoinParamVec &paramVec, [CbcModel](#) \*model)
- void [loadCbcParamObj](#) (const CoinParamVec paramVec, int first, int last, [CbcModel](#) \*model)
- void [setCbcModelDefaults](#) ([CbcModel](#) \*model)
- int [pushCbcCbcDbI](#) (CoinParam \*param)
- int [pushCbcCbcInt](#) (CoinParam \*param)

#### 6.1.1 Function Documentation

6.1.1.1 void CbcCbcParamUtils::addCbcCbcParams ( int & numParams, CoinParamVec & paramVec, CbcModel \* model )

6.1.1.2 void CbcCbcParamUtils::loadCbcParamObj ( const CoinParamVec paramVec, int first, int last, CbcModel \* model )



6.1.1.3 void CbcCbcParamUtils::setCbcModelDefaults ( CbcModel \* *model* )

6.1.1.4 int CbcCbcParamUtils::pushCbcCbcDbI ( CoinParam \* *param* )

6.1.1.5 int CbcCbcParamUtils::pushCbcCbcIInt ( CoinParam \* *param* )

## 6.2 CbcGenParamUtils Namespace Reference

### Functions

- void [addCbcGenParams](#) (int &numParams, CoinParamVec &paramVec, CbcGenCtlBlk \*ctlBlk)
- void [loadGenParamObj](#) (const CoinParamVec paramVec, int first, int last, CbcGenCtlBlk \*ctlBlk)
- void [saveSolution](#) (const OsiSolverInterface \*osi, std::string fileName)
- bool [readSolution](#) (std::string fileName, int &numRows, int &numCols, double &objVal, double \*\*rowActivity, double \*\*dualVars, double \*\*primalVars, double \*\*reducedCosts)
- int [doBaCParam](#) (CoinParam \*param)
- int [doDebugParam](#) (CoinParam \*param)
- int [doExitParam](#) (CoinParam \*param)
- int [doHelpParam](#) (CoinParam \*param)
- int [doIImportParam](#) (CoinParam \*param)
- int [doPrintMaskParam](#) (CoinParam \*param)
- int [doNothingParam](#) (CoinParam \*param)
- int [doSolutionParam](#) (CoinParam \*param)
- int [doUnimplementedParam](#) (CoinParam \*param)
- int [doVersionParam](#) (CoinParam \*param)
- int [pushCbcGenDbIParam](#) (CoinParam \*param)
- int [pushCbcGenIIntParam](#) (CoinParam \*param)
- int [pushCbcGenKwdParam](#) (CoinParam \*param)
- int [pushCbcGenStrParam](#) (CoinParam \*param)
- int [pushCbcGenCutParam](#) (CoinParam \*param)

### 6.2.1 Function Documentation

6.2.1.1 void CbcGenParamUtils::addCbcGenParams ( int & *numParams*, CoinParamVec & *paramVec*, CbcGenCtlBlk \* *ctlBlk* )

6.2.1.2 void CbcGenParamUtils::loadGenParamObj ( const CoinParamVec *paramVec*, int *first*, int *last*, CbcGenCtlBlk \* *ctlBlk* )

6.2.1.3 void CbcGenParamUtils::saveSolution ( const OsiSolverInterface \* *osi*, std::string *fileName* )

6.2.1.4 bool CbcGenParamUtils::readSolution ( std::string *fileName*, int & *numRows*, int & *numCols*, double & *objVal*, double \*\* *rowActivity*, double \*\* *dualVars*, double \*\* *primalVars*, double \*\* *reducedCosts* )

6.2.1.5 int CbcGenParamUtils::doBaCParam ( CoinParam \* *param* )

6.2.1.6 int CbcGenParamUtils::doDebugParam ( CoinParam \* *param* )

6.2.1.7 int CbcGenParamUtils::doExitParam ( CoinParam \* *param* )

6.2.1.8 int CbcGenParamUtils::doHelpParam ( CoinParam \* *param* )

6.2.1.9 int CbcGenParamUtils::doIImportParam ( CoinParam \* *param* )

6.2.1.10 int CbcGenParamUtils::doPrintMaskParam ( CoinParam \* *param* )

- 6.2.1.11 `int CbcGenParamUtils::doNothingParam ( CoinParam * param )`
- 6.2.1.12 `int CbcGenParamUtils::doSolutionParam ( CoinParam * param )`
- 6.2.1.13 `int CbcGenParamUtils::doUnimplementedParam ( CoinParam * param )`
- 6.2.1.14 `int CbcGenParamUtils::doVersionParam ( CoinParam * param )`
- 6.2.1.15 `int CbcGenParamUtils::pushCbcGenDbiParam ( CoinParam * param )`
- 6.2.1.16 `int CbcGenParamUtils::pushCbcGenIntParam ( CoinParam * param )`
- 6.2.1.17 `int CbcGenParamUtils::pushCbcGenKwdParam ( CoinParam * param )`
- 6.2.1.18 `int CbcGenParamUtils::pushCbcGenStrParam ( CoinParam * param )`
- 6.2.1.19 `int CbcGenParamUtils::pushCbcGenCutParam ( CoinParam * param )`

### 6.3 CbcOsiParamUtils Namespace Reference

#### Functions

- void [addCbcOsiParams](#) (int &numParams, CoinParamVec &paramVec, OsiSolverInterface \*osi)
- void [loadOsiParamObj](#) (const CoinParamVec paramVec, [CbcGenCtlBlk](#) \*ctlBlk)
- void [setOsiSolverInterfaceDefaults](#) (OsiSolverInterface \*osi)
- int [pushCbcOsiLogLevel](#) (CoinParam \*param)
- int [pushCbcOsiInt](#) (CoinParam \*param)
- int [pushCbcOsiDbI](#) (CoinParam \*param)
- int [pushCbcOsiKwd](#) (CoinParam \*param)
- int [pushCbcOsiHint](#) (CoinParam \*param)

#### 6.3.1 Function Documentation

- 6.3.1.1 `void CbcOsiParamUtils::addCbcOsiParams ( int & numParams, CoinParamVec & paramVec, OsiSolverInterface * osi )`
- 6.3.1.2 `void CbcOsiParamUtils::loadOsiParamObj ( const CoinParamVec paramVec, CbcGenCtlBlk * ctlBlk )`
- 6.3.1.3 `void CbcOsiParamUtils::setOsiSolverInterfaceDefaults ( OsiSolverInterface * osi )`
- 6.3.1.4 `int CbcOsiParamUtils::pushCbcOsiLogLevel ( CoinParam * param )`
- 6.3.1.5 `int CbcOsiParamUtils::pushCbcOsiInt ( CoinParam * param )`
- 6.3.1.6 `int CbcOsiParamUtils::pushCbcOsiDbI ( CoinParam * param )`
- 6.3.1.7 `int CbcOsiParamUtils::pushCbcOsiKwd ( CoinParam * param )`
- 6.3.1.8 `int CbcOsiParamUtils::pushCbcOsiHint ( CoinParam * param )`

## 7 Class Documentation

### 7.1 ampl\_info Struct Reference

```
#include <Cbc_ampl.h>
```

## Public Attributes

- int [numberRows](#)
- int [numberColumns](#)
- int [numberBinary](#)
- int [numberIntegers](#)
- int [numberSos](#)
- int [numberElements](#)
- int [numberArguments](#)
- int [problemStatus](#)
- double [direction](#)
- double [offset](#)
- double [objValue](#)
- double \* [objective](#)
- double \* [rowLower](#)
- double \* [rowUpper](#)
- double \* [columnLower](#)
- double \* [columnUpper](#)
- int \* [starts](#)
- int \* [rows](#)
- double \* [elements](#)
- double \* [primalSolution](#)
- double \* [dualSolution](#)
- int \* [columnStatus](#)
- int \* [rowStatus](#)
- int \* [priorities](#)
- int \* [branchDirection](#)
- double \* [pseudoDown](#)
- double \* [pseudoUp](#)
- char \* [sosType](#)
- int \* [sosPriority](#)
- int \* [sosStart](#)
- int \* [sosIndices](#)
- double \* [sosReference](#)
- int \* [cut](#)
- int \* [special](#)
- char \*\* [arguments](#)
- char [buffer](#) [300]
- int [logLevel](#)
- int [nonLinear](#)

### 7.1.1 Detailed Description

Definition at line 11 of file Cbc\_ampl.h.

### 7.1.2 Member Data Documentation

#### 7.1.2.1 int ampl\_info::numberRows

Definition at line 12 of file Cbc\_ampl.h.

#### 7.1.2.2 `int ampl_info::numberColumns`

Definition at line 13 of file `Cbc_ampl.h`.

#### 7.1.2.3 `int ampl_info::numberBinary`

Definition at line 14 of file `Cbc_ampl.h`.

#### 7.1.2.4 `int ampl_info::numberIntegers`

Definition at line 15 of file `Cbc_ampl.h`.

#### 7.1.2.5 `int ampl_info::numberSos`

Definition at line 16 of file `Cbc_ampl.h`.

#### 7.1.2.6 `int ampl_info::numberElements`

Definition at line 17 of file `Cbc_ampl.h`.

#### 7.1.2.7 `int ampl_info::numberArguments`

Definition at line 18 of file `Cbc_ampl.h`.

#### 7.1.2.8 `int ampl_info::problemStatus`

Definition at line 19 of file `Cbc_ampl.h`.

#### 7.1.2.9 `double ampl_info::direction`

Definition at line 20 of file `Cbc_ampl.h`.

#### 7.1.2.10 `double ampl_info::offset`

Definition at line 21 of file `Cbc_ampl.h`.

#### 7.1.2.11 `double ampl_info::objValue`

Definition at line 22 of file `Cbc_ampl.h`.

#### 7.1.2.12 `double* ampl_info::objective`

Definition at line 23 of file `Cbc_ampl.h`.

#### 7.1.2.13 `double* ampl_info::rowLower`

Definition at line 24 of file `Cbc_ampl.h`.

#### 7.1.2.14 `double* ampl_info::rowUpper`

Definition at line 25 of file `Cbc_ampl.h`.

#### 7.1.2.15 `double* ampl_info::columnLower`

Definition at line 26 of file `Cbc_ampl.h`.

**7.1.2.16 double\* ampl\_info::columnUpper**

Definition at line 27 of file Cbc\_ampl.h.

**7.1.2.17 int\* ampl\_info::starts**

Definition at line 28 of file Cbc\_ampl.h.

**7.1.2.18 int\* ampl\_info::rows**

Definition at line 29 of file Cbc\_ampl.h.

**7.1.2.19 double\* ampl\_info::elements**

Definition at line 30 of file Cbc\_ampl.h.

**7.1.2.20 double\* ampl\_info::primalSolution**

Definition at line 31 of file Cbc\_ampl.h.

**7.1.2.21 double\* ampl\_info::dualSolution**

Definition at line 32 of file Cbc\_ampl.h.

**7.1.2.22 int\* ampl\_info::columnStatus**

Definition at line 33 of file Cbc\_ampl.h.

**7.1.2.23 int\* ampl\_info::rowStatus**

Definition at line 34 of file Cbc\_ampl.h.

**7.1.2.24 int\* ampl\_info::priorities**

Definition at line 35 of file Cbc\_ampl.h.

**7.1.2.25 int\* ampl\_info::branchDirection**

Definition at line 36 of file Cbc\_ampl.h.

**7.1.2.26 double\* ampl\_info::pseudoDown**

Definition at line 37 of file Cbc\_ampl.h.

**7.1.2.27 double\* ampl\_info::pseudoUp**

Definition at line 38 of file Cbc\_ampl.h.

**7.1.2.28 char\* ampl\_info::sosType**

Definition at line 39 of file Cbc\_ampl.h.

**7.1.2.29 int\* ampl\_info::sosPriority**

Definition at line 40 of file Cbc\_ampl.h.

**7.1.2.30 int\* ampl\_info::sosStart**

Definition at line 41 of file Cbc\_ampl.h.

**7.1.2.31 int\* ampl\_info::sosIndices**

Definition at line 42 of file Cbc\_ampl.h.

**7.1.2.32 double\* ampl\_info::sosReference**

Definition at line 43 of file Cbc\_ampl.h.

**7.1.2.33 int\* ampl\_info::cut**

Definition at line 44 of file Cbc\_ampl.h.

**7.1.2.34 int\* ampl\_info::special**

Definition at line 45 of file Cbc\_ampl.h.

**7.1.2.35 char\*\* ampl\_info::arguments**

Definition at line 46 of file Cbc\_ampl.h.

**7.1.2.36 char ampl\_info::buffer[300]**

Definition at line 47 of file Cbc\_ampl.h.

**7.1.2.37 int ampl\_info::logLevel**

Definition at line 48 of file Cbc\_ampl.h.

**7.1.2.38 int ampl\_info::nonLinear**

Definition at line 49 of file Cbc\_ampl.h.

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

- [/home/ted/COIN/trunk/Cbc/src/Cbc\\_ampl.h](#)

**7.2 CbcGenCtlBlk::babState\_struct Struct Reference**

State of branch-and-cut.

```
#include <CbcGenCtlBlk.hpp>
```

**Public Attributes**

- [BACMajor majorStatus\\_](#)
- [BACMinor minorStatus\\_](#)
- [BACWhere where\\_](#)
- bool [haveAnswer\\_](#)
- OsiSolverInterface \* [answerSolver\\_](#)

### 7.2.1 Detailed Description

State of branch-and-cut.

Major and minor status codes, and a solver holding the answer, assuming we have a valid answer. See the documentation with the BACMajor, BACMinor, and BACWhere enums for the meaning of the codes.

Definition at line 718 of file CbcGenCtlBlk.hpp.

### 7.2.2 Member Data Documentation

#### 7.2.2.1 BACMajor CbcGenCtlBlk::babState\_struct::majorStatus\_

Definition at line 719 of file CbcGenCtlBlk.hpp.

#### 7.2.2.2 BACMinor CbcGenCtlBlk::babState\_struct::minorStatus\_

Definition at line 720 of file CbcGenCtlBlk.hpp.

#### 7.2.2.3 BACWhere CbcGenCtlBlk::babState\_struct::where\_

Definition at line 721 of file CbcGenCtlBlk.hpp.

#### 7.2.2.4 bool CbcGenCtlBlk::babState\_struct::haveAnswer\_

Definition at line 722 of file CbcGenCtlBlk.hpp.

#### 7.2.2.5 OsiSolverInterface\* CbcGenCtlBlk::babState\_struct::answerSolver\_

Definition at line 723 of file CbcGenCtlBlk.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp](#)

## 7.3 CbcBaseModel Class Reference

Base model.

```
#include <CbcThread.hpp>
```

### Public Member Functions

- [CbcBaseModel](#) ()
- virtual [~CbcBaseModel](#) ()

### 7.3.1 Detailed Description

Base model.

Definition at line 429 of file CbcThread.hpp.

### 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 CbcBaseModel::CbcBaseModel ( )

### 7.3.2.2 virtual CbcBaseModel::~~CbcBaseModel ( ) [inline],[virtual]

Definition at line 434 of file CbcThread.hpp.

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

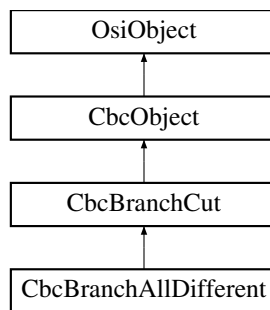
- [/home/ted/COIN/trunk/Cbc/src/CbcThread.hpp](#)

## 7.4 CbcBranchAllDifferent Class Reference

Define a branch class that branches so that it is only satisfied if all members have different values So cut is  $x \leq y-1$  or  $x \geq y+1$ .

```
#include <CbcBranchAllDifferent.hpp>
```

Inheritance diagram for CbcBranchAllDifferent:



### Public Member Functions

- [CbcBranchAllDifferent \( \)](#)
- [CbcBranchAllDifferent \(CbcModel \\*model, int number, const int \\*which\)](#)  
*Useful constructor - passed set of integer variables which must all be different.*
- [CbcBranchAllDifferent \(const CbcBranchAllDifferent &\)](#)
- virtual [CbcObject \\* clone \( \)](#) const  
*Clone.*
- [CbcBranchAllDifferent & operator= \(const CbcBranchAllDifferent &rhs\)](#)
- [~CbcBranchAllDifferent \( \)](#)
- virtual double [infeasibility](#) (const OsiBranchingInformation \*info, int &preferredWay) const  
*Infeasibility - large is 0.5.*
- virtual [CbcBranchingObject \\* createCbcBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way)  
*Creates a branching object.*

### Protected Attributes

- int [numberInSet\\_](#)  
*data*
- int \* [which\\_](#)  
*Which variables.*



### 7.4.1 Detailed Description

Define a branch class that branches so that it is only satisfied if all members have different values So cut is  $x \leq y-1$  or  $x \geq y+1$ .

Definition at line 22 of file CbcBranchAllDifferent.hpp.

### 7.4.2 Constructor & Destructor Documentation

7.4.2.1 CbcBranchAllDifferent::CbcBranchAllDifferent ( )

7.4.2.2 CbcBranchAllDifferent::CbcBranchAllDifferent ( CbcModel \* *model*, int *number*, const int \* *which* )

Useful constructor - passed set of integer variables which must all be different.

7.4.2.3 CbcBranchAllDifferent::CbcBranchAllDifferent ( const CbcBranchAllDifferent & )

7.4.2.4 CbcBranchAllDifferent::~~CbcBranchAllDifferent ( )

### 7.4.3 Member Function Documentation

7.4.3.1 virtual CbcObject\* CbcBranchAllDifferent::clone ( ) const [virtual]

Clone.

Reimplemented from [CbcBranchCut](#).

7.4.3.2 CbcBranchAllDifferent& CbcBranchAllDifferent::operator= ( const CbcBranchAllDifferent & *rhs* )

7.4.3.3 virtual double CbcBranchAllDifferent::infeasibility ( const OsiBranchingInformation \* *info*, int & *preferredWay* ) const [virtual]

Infeasibility - large is 0.5.

Reimplemented from [CbcBranchCut](#).

7.4.3.4 virtual CbcBranchingObject\* CbcBranchAllDifferent::createCbcBranch ( OsiSolverInterface \* *solver*, const OsiBranchingInformation \* *info*, int *way* ) [virtual]

Creates a branching object.

Reimplemented from [CbcBranchCut](#).

### 7.4.4 Member Data Documentation

7.4.4.1 int CbcBranchAllDifferent::numberInSet\_ [protected]

data

Number of entries

Definition at line 57 of file CbcBranchAllDifferent.hpp.

7.4.4.2 int\* CbcBranchAllDifferent::which\_ [protected]

Which variables.

Definition at line 59 of file CbcBranchAllDifferent.hpp.

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

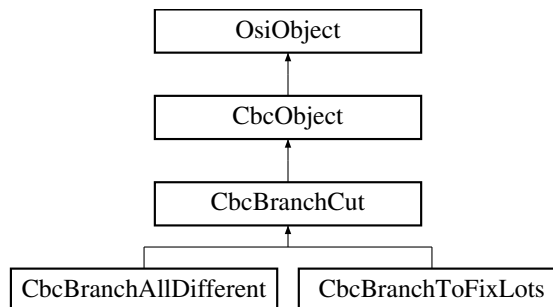
- [/home/ted/COIN/trunk/Cbc/src/CbcBranchAllDifferent.hpp](#)

## 7.5 CbcBranchCut Class Reference

Define a cut branching class.

```
#include <CbcBranchCut.hpp>
```

Inheritance diagram for CbcBranchCut:



### Public Member Functions

- [CbcBranchCut \(\)](#)
- [CbcBranchCut \(CbcModel \\*model\)](#)  
*In to maintain normal methods.*
- [CbcBranchCut \(const CbcBranchCut &\)](#)
- virtual [CbcObject \\* clone \(\)](#) const  
*Clone.*
- [CbcBranchCut & operator= \(const CbcBranchCut &rhs\)](#)
- [~CbcBranchCut \(\)](#)
- virtual double [infeasibility](#) (const OsiBranchingInformation \*info, int &[preferredWay](#)) const  
*Infeasibility.*
- virtual void [feasibleRegion](#) ()  
*Set bounds to contain the current solution.*
- virtual bool [boundBranch](#) () const  
*Return true if branch created by object should fix variables.*
- virtual [CbcBranchingObject \\* createCbcBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way)  
*Creates a branching object.*
- virtual [CbcBranchingObject \\* preferredNewFeasible](#) () const  
*Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in the good direction.*
- virtual [CbcBranchingObject \\* notPreferredNewFeasible](#) () const  
*Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a bad direction.*
- virtual void [resetBounds](#) ()  
*Reset original upper and lower bound values from the solver.*

## Additional Inherited Members

### 7.5.1 Detailed Description

Define a cut branching class.

At present empty - all stuff in descendants

Definition at line 17 of file CbcBranchCut.hpp.

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 CbcBranchCut::CbcBranchCut ( )

#### 7.5.2.2 CbcBranchCut::CbcBranchCut ( CbcModel \* *model* )

In to maintain normal methods.

#### 7.5.2.3 CbcBranchCut::CbcBranchCut ( const CbcBranchCut & )

#### 7.5.2.4 CbcBranchCut::~~CbcBranchCut ( )

### 7.5.3 Member Function Documentation

#### 7.5.3.1 virtual CbcObject\* CbcBranchCut::clone ( ) const [virtual]

Clone.

Implements [CbcObject](#).

Reimplemented in [CbcBranchToFixLots](#), and [CbcBranchAllDifferent](#).

#### 7.5.3.2 CbcBranchCut& CbcBranchCut::operator= ( const CbcBranchCut & *rhs* )

#### 7.5.3.3 virtual double CbcBranchCut::infeasibility ( const OsiBranchingInformation \* *info*, int & *preferredWay* ) const [virtual]

Infeasibility.

Reimplemented from [CbcObject](#).

Reimplemented in [CbcBranchToFixLots](#), and [CbcBranchAllDifferent](#).

#### 7.5.3.4 virtual void CbcBranchCut::feasibleRegion ( ) [virtual]

Set bounds to contain the current solution.

More precisely, for the variable associated with this object, take the value given in the current solution, force it within the current bounds if required, then set the bounds to fix the variable at the integer nearest the solution value.

At present this will do nothing

Implements [CbcObject](#).

#### 7.5.3.5 virtual bool CbcBranchCut::boundBranch ( ) const [virtual]

Return true if branch created by object should fix variables.

**7.5.3.6** `virtual CbcBranchingObject* CbcBranchCut::createCbcBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way ) [virtual]`

Creates a branching object.

Reimplemented from [CbcObject](#).

Reimplemented in [CbcBranchToFixLots](#), and [CbcBranchAllDifferent](#).

**7.5.3.7** `virtual CbcBranchingObject* CbcBranchCut::preferredNewFeasible ( ) const [virtual]`

Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in the good direction.

The preferred branching object will force the variable to be +/-1 from its current value, depending on the reduced cost and objective sense. If movement in the direction which improves the objective is impossible due to bounds on the variable, the branching object will move in the other direction. If no movement is possible, the method returns NULL.

Only the bounds on this variable are considered when determining if the new point is feasible.

At present this does nothing

Reimplemented from [CbcObject](#).

**7.5.3.8** `virtual CbcBranchingObject* CbcBranchCut::notPreferredNewFeasible ( ) const [virtual]`

Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a bad direction.

As for [preferredNewFeasible\(\)](#), but the preferred branching object will force movement in a direction that degrades the objective.

At present this does nothing

Reimplemented from [CbcObject](#).

**7.5.3.9** `virtual void CbcBranchCut::resetBounds ( ) [virtual]`

Reset original upper and lower bound values from the solver.

Handy for updating bounds held in this object after bounds held in the solver have been tightened.

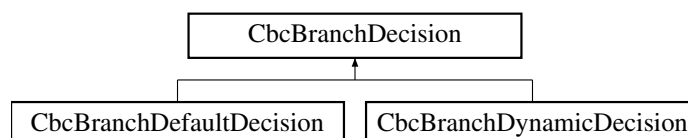
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcBranchCut.hpp](#)

## 7.6 CbcBranchDecision Class Reference

```
#include <CbcBranchDecision.hpp>
```

Inheritance diagram for CbcBranchDecision:



## Public Member Functions

- [CbcBranchDecision](#) ()  
*Default Constructor.*
- [CbcBranchDecision](#) (const [CbcBranchDecision](#) &)
- virtual [~CbcBranchDecision](#) ()  
*Destructor.*
- virtual [CbcBranchDecision](#) \* [clone](#) () const =0  
*Clone.*
- virtual void [initialize](#) ([CbcModel](#) \*model)=0  
*Initialize e.g. before starting to choose a branch at a node.*
- virtual int [betterBranch](#) ([CbcBranchingObject](#) \*thisOne, [CbcBranchingObject](#) \*bestSoFar, double changeUp, int numberInfeasibilitiesUp, double changeDown, int numberInfeasibilitiesDown)=0  
*Compare two branching objects.*
- virtual int [bestBranch](#) ([CbcBranchingObject](#) \*\*objects, int numberObjects, int numberUnsatisfied, double \*changeUp, int \*numberInfeasibilitiesUp, double \*changeDown, int \*numberInfeasibilitiesDown, double objectiveValue)  
*Compare N branching objects.*
- virtual int [whichMethod](#) ()  
*Says whether this method can handle both methods - 1 better, 2 best, 3 both.*
- virtual void [saveBranchingObject](#) ([OsiBranchingObject](#) \*)  
*Saves a clone of current branching object.*
- virtual void [updateInformation](#) ([OsiSolverInterface](#) \*, const [CbcNode](#) \*)  
*Pass in information on branch just done.*
- virtual void [setBestCriterion](#) (double)  
*Sets or gets best criterion so far.*
- virtual double [getBestCriterion](#) () const
- virtual void [generateCpp](#) (FILE \*)  
*Create C++ lines to get to current state.*
- [CbcModel](#) \* [cbcModel](#) () const  
*Model.*
- [OsiChooseVariable](#) \* [chooseMethod](#) () const
- void [setChooseMethod](#) (const [OsiChooseVariable](#) &method)  
*Set (clone) chooseMethod.*

## Protected Attributes

- [CbcBranchingObject](#) \* [object\\_](#)
- [CbcModel](#) \* [model\\_](#)  
*Pointer to model.*
- [OsiChooseVariable](#) \* [chooseMethod\\_](#)

## 7.6.1 Detailed Description

Definition at line 28 of file [CbcBranchDecision.hpp](#).

## 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 `CbcBranchDecision::CbcBranchDecision ( )`

Default Constructor.

### 7.6.2.2 `CbcBranchDecision::CbcBranchDecision ( const CbcBranchDecision & )`

### 7.6.2.3 `virtual CbcBranchDecision::~~CbcBranchDecision ( ) [virtual]`

Destructor.

## 7.6.3 Member Function Documentation

### 7.6.3.1 `virtual CbcBranchDecision* CbcBranchDecision::clone ( ) const [pure virtual]`

Clone.

Implemented in [CbcBranchDynamicDecision](#), and [CbcBranchDefaultDecision](#).

### 7.6.3.2 `virtual void CbcBranchDecision::initialize ( CbcModel * model ) [pure virtual]`

Initialize e.g. before starting to choose a branch at a node.

Implemented in [CbcBranchDynamicDecision](#), and [CbcBranchDefaultDecision](#).

### 7.6.3.3 `virtual int CbcBranchDecision::betterBranch ( CbcBranchingObject * thisOne, CbcBranchingObject * bestSoFar, double changeUp, int numberInfeasibilitiesUp, double changeDown, int numberInfeasibilitiesDown ) [pure virtual]`

Compare two branching objects.

Return nonzero if branching using `thisOne` is better than branching using `bestSoFar`.

If `bestSoFar` is NULL, the routine should return a nonzero value. This routine is used only after strong branching. Either this or `bestBranch` is used depending which user wants.

Implemented in [CbcBranchDynamicDecision](#), and [CbcBranchDefaultDecision](#).

### 7.6.3.4 `virtual int CbcBranchDecision::bestBranch ( CbcBranchingObject ** objects, int numberObjects, int numberUnsatisfied, double * changeUp, int * numberInfeasibilitiesUp, double * changeDown, int * numberInfeasibilitiesDown, double objectiveValue ) [virtual]`

Compare N branching objects.

Return index of best and sets way of branching in chosen object.

Either this or `betterBranch` is used depending which user wants.

Reimplemented in [CbcBranchDefaultDecision](#).

### 7.6.3.5 `virtual int CbcBranchDecision::whichMethod ( ) [inline],[virtual]`

Says whether this method can handle both methods - 1 better, 2 best, 3 both.

Reimplemented in [CbcBranchDynamicDecision](#).

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

**7.6.3.6** `virtual void CbcBranchDecision::saveBranchingObject ( OsiBranchingObject * ) [inline],[virtual]`

Saves a clone of current branching object.

Can be used to update information on object causing branch - after branch

Reimplemented in [CbcBranchDynamicDecision](#).

Definition at line 80 of file CbcBranchDecision.hpp.

**7.6.3.7** `virtual void CbcBranchDecision::updateInformation ( OsiSolverInterface *, const CbcNode * ) [inline],[virtual]`

Pass in information on branch just done.

assumes object can get information from solver

Reimplemented in [CbcBranchDynamicDecision](#).

Definition at line 83 of file CbcBranchDecision.hpp.

**7.6.3.8** `virtual void CbcBranchDecision::setBestCriterion ( double ) [inline],[virtual]`

Sets or gets best criterion so far.

Reimplemented in [CbcBranchDynamicDecision](#), and [CbcBranchDefaultDecision](#).

Definition at line 86 of file CbcBranchDecision.hpp.

**7.6.3.9** `virtual double CbcBranchDecision::getBestCriterion ( ) const [inline],[virtual]`

Reimplemented in [CbcBranchDynamicDecision](#), and [CbcBranchDefaultDecision](#).

Definition at line 87 of file CbcBranchDecision.hpp.

**7.6.3.10** `virtual void CbcBranchDecision::generateCpp ( FILE * ) [inline],[virtual]`

Create C++ lines to get to current state.

Definition at line 91 of file CbcBranchDecision.hpp.

**7.6.3.11** `CbcModel* CbcBranchDecision::cbcModel ( ) const [inline]`

Model.

Definition at line 93 of file CbcBranchDecision.hpp.

**7.6.3.12** `OsiChooseVariable* CbcBranchDecision::chooseMethod ( ) const [inline]`

Definition at line 107 of file CbcBranchDecision.hpp.

**7.6.3.13** `void CbcBranchDecision::setChooseMethod ( const OsiChooseVariable & method )`

Set (clone) chooseMethod.

## 7.6.4 Member Data Documentation

**7.6.4.1** `CbcBranchingObject* CbcBranchDecision::object_ [protected]`

Definition at line 116 of file CbcBranchDecision.hpp.

#### 7.6.4.2 `CbcModel*` `CbcBranchDecision::model_` [protected]

Pointer to model.

Definition at line 118 of file `CbcBranchDecision.hpp`.

#### 7.6.4.3 `OsiChooseVariable*` `CbcBranchDecision::chooseMethod_` [protected]

Definition at line 122 of file `CbcBranchDecision.hpp`.

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

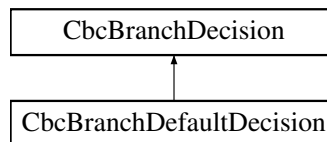
- [/home/ted/COIN/trunk/Cbc/src/CbcBranchDecision.hpp](#)

## 7.7 `CbcBranchDefaultDecision` Class Reference

Branching decision default class.

```
#include <CbcBranchDefaultDecision.hpp>
```

Inheritance diagram for `CbcBranchDefaultDecision`:



### Public Member Functions

- [CbcBranchDefaultDecision](#) ()
- [CbcBranchDefaultDecision](#) (const [CbcBranchDefaultDecision](#) &)
- virtual [~CbcBranchDefaultDecision](#) ()
- virtual [CbcBranchDecision](#) \* [clone](#) () const  
*Clone.*
- virtual void [initialize](#) ([CbcModel](#) \*model)  
*Initialize, e.g. before the start of branch selection at a node.*
- virtual int [betterBranch](#) ([CbcBranchingObject](#) \*thisOne, [CbcBranchingObject](#) \*bestSoFar, double changeUp, int numInfUp, double changeDn, int numInfDn)  
*Compare two branching objects.*
- virtual void [setBestCriterion](#) (double value)  
*Sets or gets best criterion so far.*
- virtual double [getBestCriterion](#) () const
- virtual int [bestBranch](#) ([CbcBranchingObject](#) \*\*objects, int numberObjects, int numberUnsatisfied, double \*changeUp, int \*numberInfeasibilitiesUp, double \*changeDown, int \*numberInfeasibilitiesDown, double objectiveValue)  
*Compare N branching objects.*

### Additional Inherited Members



## 7.7.1 Detailed Description

Branching decision default class.

This class implements a simple default algorithm ([betterBranch\(\)](#)) for choosing a branching variable.

Definition at line 18 of file CbcBranchDefaultDecision.hpp.

## 7.7.2 Constructor &amp; Destructor Documentation

7.7.2.1 `CbcBranchDefaultDecision::CbcBranchDefaultDecision ( )`

7.7.2.2 `CbcBranchDefaultDecision::CbcBranchDefaultDecision ( const CbcBranchDefaultDecision & )`

7.7.2.3 `virtual CbcBranchDefaultDecision::~~CbcBranchDefaultDecision ( )` [virtual]

## 7.7.3 Member Function Documentation

7.7.3.1 `virtual CbcBranchDecision* CbcBranchDefaultDecision::clone ( ) const` [virtual]

Clone.

Implements [CbcBranchDecision](#).

7.7.3.2 `virtual void CbcBranchDefaultDecision::initialize ( CbcModel * model )` [virtual]

Initialize, e.g. before the start of branch selection at a node.

Implements [CbcBranchDecision](#).

7.7.3.3 `virtual int CbcBranchDefaultDecision::betterBranch ( CbcBranchingObject * thisOne, CbcBranchingObject * bestSoFar, double changeUp, int numInfUp, double changeDn, int numInfDn )` [virtual]

Compare two branching objects.

Return nonzero if `thisOne` is better than `bestSoFar`.

The routine compares branches using the values supplied in `numInfUp` and `numInfDn` until a solution is found by search, after which it uses the values supplied in `changeUp` and `changeDn`. The best branching object seen so far and the associated parameter values are remembered in the [CbcBranchDefaultDecision](#) object. The nonzero return value is +1 if the up branch is preferred, -1 if the down branch is preferred.

As the names imply, the assumption is that the values supplied for `numInfUp` and `numInfDn` will be the number of infeasibilities reported by the branching object, and `changeUp` and `changeDn` will be the estimated change in objective. Other measures can be used if desired.

Because an [CbcBranchDefaultDecision](#) object remembers the current best branching candidate (`#bestObject_`) as well as the values used in the comparison, the parameter `bestSoFar` is redundant, hence unused.

Implements [CbcBranchDecision](#).

7.7.3.4 `virtual void CbcBranchDefaultDecision::setBestCriterion ( double value )` [virtual]

Sets or gets best criterion so far.

Reimplemented from [CbcBranchDecision](#).

7.7.3.5 `virtual double CbcBranchDefaultDecision::getBestCriterion ( ) const` [virtual]

Reimplemented from [CbcBranchDecision](#).

7.7.3.6 `virtual int CbcBranchDefaultDecision::bestBranch ( CbcBranchingObject ** objects, int numberObjects, int numberUnsatisfied, double * changeUp, int * numberInfeasibilitiesUp, double * changeDown, int * numberInfeasibilitiesDown, double objectiveValue ) [virtual]`

Compare N branching objects.

Return index of best and sets way of branching in chosen object.

This routine is used only after strong branching.

Reimplemented from [CbcBranchDecision](#).

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

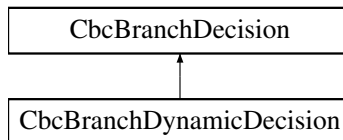
- [/home/ted/COIN/trunk/Cbc/src/CbcBranchDefaultDecision.hpp](#)

## 7.8 CbcBranchDynamicDecision Class Reference

Branching decision dynamic class.

```
#include <CbcBranchDynamic.hpp>
```

Inheritance diagram for CbcBranchDynamicDecision:



### Public Member Functions

- [CbcBranchDynamicDecision](#) ()
- [CbcBranchDynamicDecision](#) (const [CbcBranchDynamicDecision](#) &)
- `virtual ~CbcBranchDynamicDecision ()`
- `virtual CbcBranchDecision * clone () const`  
*Clone.*
- `virtual void initialize (CbcModel *model)`  
*Initialize, e.g. before the start of branch selection at a node.*
- `virtual int betterBranch (CbcBranchingObject *thisOne, CbcBranchingObject *bestSoFar, double changeUp, int numInfUp, double changeDn, int numInfDn)`  
*Compare two branching objects.*
- `virtual void setBestCriterion (double value)`  
*Sets or gets best criterion so far.*
- `virtual double getBestCriterion () const`
- `virtual int whichMethod ()`  
*Says whether this method can handle both methods - 1 better, 2 best, 3 both.*
- `virtual void saveBranchingObject (OsiBranchingObject *object)`  
*Saves a clone of current branching object.*
- `virtual void updateInformation (OsiSolverInterface *solver, const CbcNode *node)`  
*Pass in information on branch just done.*

## Additional Inherited Members

## 7.8.1 Detailed Description

Branching decision dynamic class.

This class implements a simple algorithm ([betterBranch\(\)](#)) for choosing a branching variable when dynamic pseudo costs.

Definition at line 19 of file CbcBranchDynamic.hpp.

## 7.8.2 Constructor &amp; Destructor Documentation

7.8.2.1 CbcBranchDynamicDecision::CbcBranchDynamicDecision ( )

7.8.2.2 CbcBranchDynamicDecision::CbcBranchDynamicDecision ( const CbcBranchDynamicDecision & )

7.8.2.3 virtual CbcBranchDynamicDecision::~CbcBranchDynamicDecision ( ) [virtual]

## 7.8.3 Member Function Documentation

7.8.3.1 virtual CbcBranchDecision\* CbcBranchDynamicDecision::clone ( ) const [virtual]

Clone.

Implements [CbcBranchDecision](#).

7.8.3.2 virtual void CbcBranchDynamicDecision::initialize ( CbcModel \* model ) [virtual]

Initialize, e.g. before the start of branch selection at a node.

Implements [CbcBranchDecision](#).

7.8.3.3 virtual int CbcBranchDynamicDecision::betterBranch ( CbcBranchingObject \* thisOne, CbcBranchingObject \* bestSoFar, double changeUp, int numInfUp, double changeDn, int numInfDn ) [virtual]

Compare two branching objects.

Return nonzero if *thisOne* is better than *bestSoFar*.

The routine compares branches using the values supplied in *numInfUp* and *numInfDn* until a solution is found by search, after which it uses the values supplied in *changeUp* and *changeDn*. The best branching object seen so far and the associated parameter values are remembered in the [CbcBranchDynamicDecision](#) object. The nonzero return value is +1 if the up branch is preferred, -1 if the down branch is preferred.

As the names imply, the assumption is that the values supplied for *numInfUp* and *numInfDn* will be the number of infeasibilities reported by the branching object, and *changeUp* and *changeDn* will be the estimated change in objective. Other measures can be used if desired.

Because an [CbcBranchDynamicDecision](#) object remembers the current best branching candidate (*#bestObject\_*) as well as the values used in the comparison, the parameter *bestSoFar* is redundant, hence unused.

Implements [CbcBranchDecision](#).

7.8.3.4 virtual void CbcBranchDynamicDecision::setBestCriterion ( double value ) [virtual]

Sets or gets best criterion so far.

Reimplemented from [CbcBranchDecision](#).

7.8.3.5 `virtual double CbcBranchDynamicDecision::getBestCriterion ( ) const` [virtual]

Reimplemented from [CbcBranchDecision](#).

7.8.3.6 `virtual int CbcBranchDynamicDecision::whichMethod ( )` [inline],[virtual]

Says whether this method can handle both methods - 1 better, 2 best, 3 both.

Reimplemented from [CbcBranchDecision](#).

Definition at line 63 of file CbcBranchDynamic.hpp.

7.8.3.7 `virtual void CbcBranchDynamicDecision::saveBranchingObject ( OsiBranchingObject * object )` [virtual]

Saves a clone of current branching object.

Can be used to update information on object causing branch - after branch

Reimplemented from [CbcBranchDecision](#).

7.8.3.8 `virtual void CbcBranchDynamicDecision::updateInformation ( OsiSolverInterface * solver, const CbcNode * node )`  
[virtual]

Pass in information on branch just done.

assumes object can get information from solver

Reimplemented from [CbcBranchDecision](#).

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

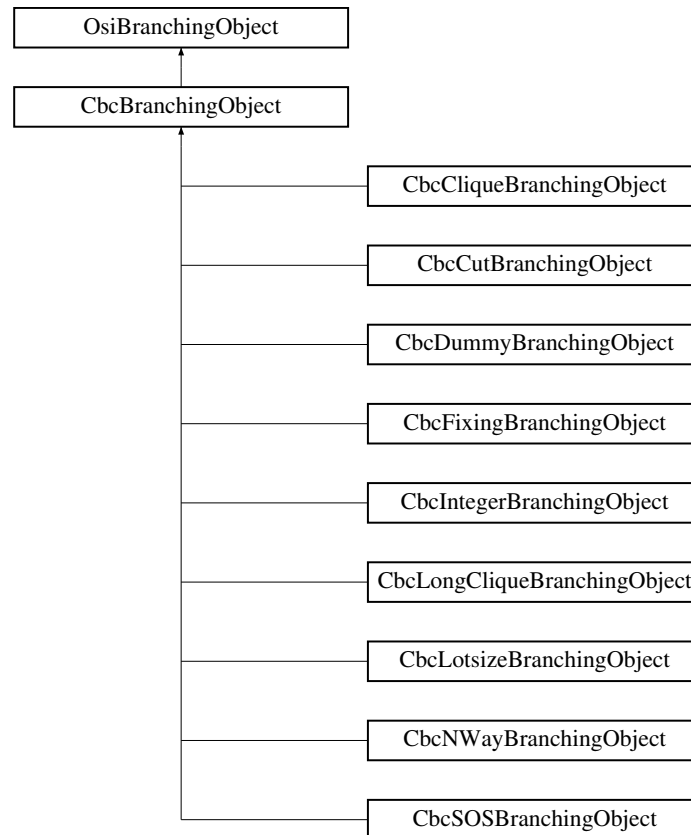
- [/home/ted/COIN/trunk/Cbc/src/CbcBranchDynamic.hpp](#)

## 7.9 CbcBranchingObject Class Reference

Abstract branching object base class Now just difference with OsiBranchingObject.

```
#include <CbcBranchingObject.hpp>
```

Inheritance diagram for CbcBranchingObject:



### Public Member Functions

- [CbcBranchingObject](#) ()  
*Default Constructor.*
- [CbcBranchingObject](#) ([CbcModel](#) \**model*, int *variable*, int *way*, double *value*)  
*Constructor.*
- [CbcBranchingObject](#) (const [CbcBranchingObject](#) &)  
*Copy constructor.*
- [CbcBranchingObject](#) & *operator=* (const [CbcBranchingObject](#) &*rhs*)  
*Assignment operator.*
- virtual [CbcBranchingObject](#) \* *clone* () const =0  
*Clone.*
- virtual ~[CbcBranchingObject](#) ()  
*Destructor.*
- virtual int *fillStrongInfo* ([CbcStrongInfo](#) &)  
*Some branchingObjects may claim to be able to skip strong branching.*
- void *resetNumberBranchesLeft* ()  
*Reset number of branches left to original.*
- void *setNumberBranches* (int *value*)  
*Set number of branches to do.*
- virtual double *branch* ()=0  
*Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.*

- virtual double **branch** (OsiSolverInterface \*)  
*Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.*
- virtual void **fix** (OsiSolverInterface \*, double \*, double \*, int) const  
*Update bounds in solver as in 'branch' and update given bounds.*
- virtual bool **tighten** (OsiSolverInterface \*)  
*Change (tighten) bounds in object to reflect bounds in solver.*
- virtual void **previousBranch** ()  
*Reset every information so that the branching object appears to point to the previous child.*
- virtual void **print** () const  
*Print something about branch - only if log level high.*
- int **variable** () const  
*Index identifying the associated CbcObject within its class.*
- int **way** () const  
*Get the state of the branching object.*
- void **way** (int way)  
*Set the state of the branching object.*
- void **setModel** (CbcModel \*model)  
*update model*
- CbcModel \* **model** () const  
*Return model.*
- CbcObject \* **object** () const  
*Return pointer back to object which created.*
- void **setOriginalObject** (CbcObject \*object)  
*Set pointer back to object which created.*
- virtual CbcBranchObjType **type** () const =0  
*Return the type (an integer identifier) of this.*
- virtual int **compareOriginalObject** (const CbcBranchingObject \*brObj) const  
*Compare the original object of this with the original object of brObj.*
- virtual CbcRangeCompare **compareBranchingObject** (const CbcBranchingObject \*brObj, const bool replaceIfOverlap=false)=0  
*Compare the this with brObj.*

#### Protected Attributes

- CbcModel \* **model\_**  
*The model that owns this branching object.*
- CbcObject \* **originalCbcObject\_**  
*Pointer back to object which created.*
- int **variable\_**  
*Branching variable (0 is first integer)*
- int **way\_**  
*The state of the branching object.*

### 7.9.1 Detailed Description

Abstract branching object base class Now just difference with OsiBranchingObject.

In the abstract, an [CbcBranchingObject](#) contains instructions for how to branch. We want an abstract class so that we can describe how to branch on simple objects (e.g., integers) and more exotic objects (e.g., cliques or hyperplanes).

The [branch\(\)](#) method is the crucial routine: it is expected to be able to step through a set of branch arms, executing the actions required to create each subproblem in turn. The base class is primarily virtual to allow for a wide range of problem modifications.

See [CbcObject](#) for an overview of the three classes ([CbcObject](#), [CbcBranchingObject](#), and [CbcBranchDecision](#)) which make up cbc's branching model.

Definition at line 53 of file CbcBranchingObject.hpp.

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 CbcBranchingObject::CbcBranchingObject ( )

Default Constructor.

#### 7.9.2.2 CbcBranchingObject::CbcBranchingObject ( CbcModel \* model, int variable, int way, double value )

Constructor.

#### 7.9.2.3 CbcBranchingObject::CbcBranchingObject ( const CbcBranchingObject & )

Copy constructor.

#### 7.9.2.4 virtual CbcBranchingObject::~~CbcBranchingObject ( ) [virtual]

Destructor.

### 7.9.3 Member Function Documentation

#### 7.9.3.1 CbcBranchingObject& CbcBranchingObject::operator= ( const CbcBranchingObject & rhs )

Assignment operator.

#### 7.9.3.2 virtual CbcBranchingObject\* CbcBranchingObject::clone ( ) const [pure virtual]

Clone.

Implemented in [CbcIntegerPseudoCostBranchingObject](#), [CbcLongCliqueBranchingObject](#), [CbcSOSBranchingObject](#), [CbcLotsizeBranchingObject](#), [CbcCliqueBranchingObject](#), [CbcDynamicPseudoCostBranchingObject](#), [CbcCutBranchingObject](#), [CbcNWayBranchingObject](#), [CbcFixingBranchingObject](#), [CbcIntegerBranchingObject](#), and [CbcDummyBranchingObject](#).

#### 7.9.3.3 virtual int CbcBranchingObject::fillStrongInfo ( CbcStrongInfo & ) [inline],[virtual]

Some branchingObjects may claim to be able to skip strong branching.

If so they have to fill in [CbcStrongInfo](#). The object mention in incoming [CbcStrongInfo](#) must match. Returns nonzero if skip is wanted

Reimplemented in [CbcDynamicPseudoCostBranchingObject](#).

Definition at line 79 of file CbcBranchingObject.hpp.

**7.9.3.4** `void CbcBranchingObject::resetNumberBranchesLeft ( ) [inline]`

Reset number of branches left to original.

Definition at line 83 of file CbcBranchingObject.hpp.

**7.9.3.5** `void CbcBranchingObject::setNumberBranches ( int value ) [inline]`

Set number of branches to do.

Definition at line 87 of file CbcBranchingObject.hpp.

**7.9.3.6** `virtual double CbcBranchingObject::branch ( ) [pure virtual]`

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

Mainly for diagnostics, whether it is true branch or strong branching is also passed. Returns change in guessed objective on next branch

Implemented in [CbcIntegerPseudoCostBranchingObject](#), [CbcLongCliqueBranchingObject](#), [CbcSOSBranchingObject](#), [CbcLotsizeBranchingObject](#), [CbcCliqueBranchingObject](#), [CbcDynamicPseudoCostBranchingObject](#), [CbcCutBranchingObject](#), [CbcNWayBranchingObject](#), [CbcFixingBranchingObject](#), [CbcIntegerBranchingObject](#), and [CbcDummyBranchingObject](#).

**7.9.3.7** `virtual double CbcBranchingObject::branch ( OsiSolverInterface * ) [inline],[virtual]`

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

Mainly for diagnostics, whether it is true branch or strong branching is also passed. Returns change in guessed objective on next branch

Definition at line 105 of file CbcBranchingObject.hpp.

**7.9.3.8** `virtual void CbcBranchingObject::fix ( OsiSolverInterface *, double *, double *, int ) const [inline],[virtual]`

Update bounds in solver as in 'branch' and update given bounds.

branchState is -1 for 'down' +1 for 'up'

Reimplemented in [CbcSOSBranchingObject](#), and [CbcIntegerBranchingObject](#).

Definition at line 110 of file CbcBranchingObject.hpp.

**7.9.3.9** `virtual bool CbcBranchingObject::tighten ( OsiSolverInterface * ) [inline],[virtual]`

Change (tighten) bounds in object to reflect bounds in solver.

Return true if now fixed

Reimplemented in [CbcIntegerBranchingObject](#).

Definition at line 116 of file CbcBranchingObject.hpp.

**7.9.3.10** `virtual void CbcBranchingObject::previousBranch ( ) [inline],[virtual]`

Reset every information so that the branching object appears to point to the previous child.

This method does not need to modify anything in any solver.



Reimplemented in [CbcSOSBranchingObject](#).

Definition at line 121 of file CbcBranchingObject.hpp.

**7.9.3.11** `virtual void CbcBranchingObject::print ( ) const` `[inline],[virtual]`

Print something about branch - only if log level high.

Definition at line 130 of file CbcBranchingObject.hpp.

**7.9.3.12** `int CbcBranchingObject::variable ( ) const` `[inline]`

Index identifying the associated [CbcObject](#) within its class.

The name is misleading, and typically the index will *not* refer directly to a variable. Rather, it identifies an [CbcObject](#) within the class of similar CbcObjects

*E.g.*, for an [CbcSimpleInteger](#), `variable()` is the index of the integer variable in the set of integer variables (*not* the index of the variable in the set of all variables).

Definition at line 143 of file CbcBranchingObject.hpp.

**7.9.3.13** `int CbcBranchingObject::way ( ) const` `[inline]`

Get the state of the branching object.

Returns a code indicating the active arm of the branching object. The precise meaning is defined in the derived class.

See Also

[way\\_](#)

Definition at line 154 of file CbcBranchingObject.hpp.

**7.9.3.14** `void CbcBranchingObject::way ( int way )` `[inline]`

Set the state of the branching object.

See [way\(\)](#)

Definition at line 162 of file CbcBranchingObject.hpp.

**7.9.3.15** `void CbcBranchingObject::setModel ( CbcModel * model )` `[inline]`

update model

Definition at line 167 of file CbcBranchingObject.hpp.

**7.9.3.16** `CbcModel* CbcBranchingObject::model ( ) const` `[inline]`

Return model.

Definition at line 171 of file CbcBranchingObject.hpp.

**7.9.3.17** `CbcObject* CbcBranchingObject::object ( ) const` `[inline]`

Return pointer back to object which created.

Definition at line 176 of file CbcBranchingObject.hpp.

**7.9.3.18** `void CbcBranchingObject::setOriginalObject ( CbcObject * object )` `[inline]`

Set pointer back to object which created.

Definition at line 180 of file CbcBranchingObject.hpp.

**7.9.3.19** `virtual CbcBranchObjType CbcBranchingObject::type ( ) const [pure virtual]`

Return the type (an integer identifier) of `this`.

See definition of CbcBranchObjType above for possibilities

Implemented in [CbcIntegerPseudoCostBranchingObject](#), [CbcLongCliqueBranchingObject](#), [CbcSOSBranchingObject](#), [CbcLotsizeBranchingObject](#), [CbcCliqueBranchingObject](#), [CbcDynamicPseudoCostBranchingObject](#), [CbcCutBranchingObject](#), [CbcNWayBranchingObject](#), [CbcIntegerBranchingObject](#), [CbcFixingBranchingObject](#), and [CbcDummyBranchingObject](#).

**7.9.3.20** `virtual int CbcBranchingObject::compareOriginalObject ( const CbcBranchingObject * brObj ) const [inline], [virtual]`

Compare the original object of `this` with the original object of `brObj`.

Assumes that there is an ordering of the original objects. This method should be invoked only if `this` and `brObj` are of the same type. Return negative/0/positive depending on whether `this` is smaller/same/larger than the argument.

Reimplemented in [CbcLongCliqueBranchingObject](#), [CbcSOSBranchingObject](#), [CbcCliqueBranchingObject](#), [CbcCutBranchingObject](#), [CbcNWayBranchingObject](#), [CbcFixingBranchingObject](#), and [CbcDummyBranchingObject](#).

Definition at line 199 of file CbcBranchingObject.hpp.

**7.9.3.21** `virtual CbcRangeCompare CbcBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false ) [pure virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate CbcRangeCompare value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implemented in [CbcIntegerPseudoCostBranchingObject](#), [CbcLongCliqueBranchingObject](#), [CbcSOSBranchingObject](#), [CbcLotsizeBranchingObject](#), [CbcCliqueBranchingObject](#), [CbcCutBranchingObject](#), [CbcNWayBranchingObject](#), [CbcIntegerBranchingObject](#), [CbcFixingBranchingObject](#), and [CbcDummyBranchingObject](#).

## 7.9.4 Member Data Documentation

**7.9.4.1** `CbcModel* CbcBranchingObject::model_ [protected]`

The model that owns this branching object.

Definition at line 218 of file CbcBranchingObject.hpp.

**7.9.4.2** `CbcObject* CbcBranchingObject::originalCbcObject_ [protected]`

Pointer back to object which created.

Definition at line 220 of file CbcBranchingObject.hpp.

**7.9.4.3** `int CbcBranchingObject::variable_ [protected]`

Branching variable (0 is first integer)

Definition at line 223 of file CbcBranchingObject.hpp.

#### 7.9.4.4 int CbcBranchingObject::way\_ [protected]

The state of the branching object.

Specifies the active arm of the branching object. Coded as -1 to take the 'down' arm, +1 for the 'up' arm. 'Down' and 'up' are defined based on the natural meaning (floor and ceiling, respectively) for a simple integer. The precise meaning is defined in the derived class.

Definition at line 232 of file CbcBranchingObject.hpp.

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

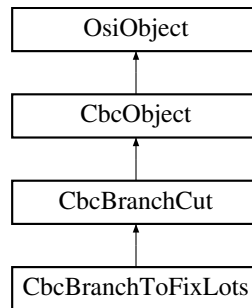
- /home/ted/COIN/trunk/Cbc/src/CbcBranchingObject.hpp

## 7.10 CbcBranchToFixLots Class Reference

Define a branch class that branches so that one way variables are fixed while the other way cuts off that solution.

```
#include <CbcBranchToFixLots.hpp>
```

Inheritance diagram for CbcBranchToFixLots:



### Public Member Functions

- [CbcBranchToFixLots](#) ()
- [CbcBranchToFixLots](#) ([CbcModel](#) \**model*, double *djTolerance*, double *fractionFixed*, int *depth*, int *numberClean*=0, const char \**mark*=NULL, bool *alwaysCreate*=false)  
*Useful constructor - passed reduced cost tolerance and fraction we would like fixed.*
- [CbcBranchToFixLots](#) (const [CbcBranchToFixLots](#) &)
- virtual [CbcObject](#) \* *clone* () const  
*Clone.*
- [CbcBranchToFixLots](#) & *operator=* (const [CbcBranchToFixLots](#) &*rhs*)
- [~CbcBranchToFixLots](#) ()
- int *shallWe* () const  
*Does a lot of the work, Returns 0 if no good, 1 if dj, 2 if clean, 3 if both FIXME: should use enum or equivalent to make these numbers clearer.*
- virtual double *infeasibility* (const [OsiBranchingInformation](#) \**info*, int &*preferredWay*) const  
*Infeasibility for an integer variable - large is 0.5, but also can be infinity when known infeasible.*
- virtual bool *canDoHeuristics* () const  
*Return true if object can take part in normal heuristics.*
- virtual [CbcBranchingObject](#) \* *createCbcBranch* ([OsiSolverInterface](#) \**solver*, const [OsiBranchingInformation](#) \**info*, int *way*)

*Creates a branching object.*

- virtual void [redoSequenceEtc](#) ([CbcModel](#) \*[model](#), int numberColumns, const int \*originalColumns)

*Redoes data when sequence numbers change.*

#### Protected Attributes

- double [djTolerance\\_](#)  
*data*
- double [fractionFixed\\_](#)  
*We only need to make sure this fraction fixed.*
- char \* [mark\\_](#)  
*Never fix ones marked here.*
- CoinPackedMatrix [matrixByRow\\_](#)  
*Matrix by row.*
- int [depth\\_](#)  
*Do if depth multiple of this.*
- int [numberClean\\_](#)  
*number of ==1 rows which need to be clean*
- bool [alwaysCreate\\_](#)  
*If true then always create branch.*

#### 7.10.1 Detailed Description

Define a branch class that branches so that one way variables are fixed while the other way cuts off that solution.

a) On reduced cost b) When enough ==1 or <=1 rows have been satisfied (not fixed - satisfied)

Definition at line 23 of file CbcBranchToFixLots.hpp.

#### 7.10.2 Constructor & Destructor Documentation

##### 7.10.2.1 CbcBranchToFixLots::CbcBranchToFixLots ( )

##### 7.10.2.2 CbcBranchToFixLots::CbcBranchToFixLots ( [CbcModel](#) \* *model*, double *djTolerance*, double *fractionFixed*, int *depth*, int *numberClean* = 0, const char \* *mark* = NULL, bool *alwaysCreate* = false )

Useful constructor - passed reduced cost tolerance and fraction we would like fixed.

Also depth level to do at. Also passed number of 1 rows which when clean triggers fix Always does if all 1 rows cleaned up and number>0 or if fraction columns reached Also whether to create branch if can't reach fraction.

##### 7.10.2.3 CbcBranchToFixLots::CbcBranchToFixLots ( const [CbcBranchToFixLots](#) & )

##### 7.10.2.4 CbcBranchToFixLots::~CbcBranchToFixLots ( )

#### 7.10.3 Member Function Documentation

##### 7.10.3.1 virtual [CbcObject](#)\* [CbcBranchToFixLots::clone](#) ( ) const [virtual]

Clone.

Reimplemented from [CbcBranchCut](#).

7.10.3.2 **CbcBranchToFixLots& CbcBranchToFixLots::operator= ( const CbcBranchToFixLots & rhs )**

7.10.3.3 **int CbcBranchToFixLots::shallWe ( ) const**

Does a lot of the work, Returns 0 if no good, 1 if dj, 2 if clean, 3 if both FIXME: should use enum or equivalent to make these numbers clearer.

7.10.3.4 **virtual double CbcBranchToFixLots::infeasibility ( const OsiBranchingInformation \* info, int & preferredWay ) const**  
[virtual]

Infeasibility for an integer variable - large is 0.5, but also can be infinity when known infeasible.

Reimplemented from [CbcBranchCut](#).

7.10.3.5 **virtual bool CbcBranchToFixLots::canDoHeuristics ( ) const** [inline],[virtual]

Return true if object can take part in normal heuristics.

Definition at line 65 of file CbcBranchToFixLots.hpp.

7.10.3.6 **virtual CbcBranchingObject\* CbcBranchToFixLots::createCbcBranch ( OsiSolverInterface \* solver, const OsiBranchingInformation \* info, int way )** [virtual]

Creates a branching object.

Reimplemented from [CbcBranchCut](#).

7.10.3.7 **virtual void CbcBranchToFixLots::redoSequenceEtc ( CbcModel \* model, int numberColumns, const int \* originalColumns )** [virtual]

Redoes data when sequence numbers change.

Reimplemented from [CbcObject](#).

#### 7.10.4 Member Data Documentation

7.10.4.1 **double CbcBranchToFixLots::djTolerance\_** [protected]

data

Reduced cost tolerance i.e. dj has to be >= this before fixed

Definition at line 79 of file CbcBranchToFixLots.hpp.

7.10.4.2 **double CbcBranchToFixLots::fractionFixed\_** [protected]

We only need to make sure this fraction fixed.

Definition at line 81 of file CbcBranchToFixLots.hpp.

7.10.4.3 **char\* CbcBranchToFixLots::mark\_** [protected]

Never fix ones marked here.

Definition at line 83 of file CbcBranchToFixLots.hpp.

7.10.4.4 **CoinPackedMatrix CbcBranchToFixLots::matrixByRow\_** [protected]

Matrix by row.

Definition at line 85 of file CbcBranchToFixLots.hpp.

7.10.4.5 `int CbcBranchToFixLots::depth_ [protected]`

Do if depth multiple of this.

Definition at line 87 of file CbcBranchToFixLots.hpp.

7.10.4.6 `int CbcBranchToFixLots::numberClean_ [protected]`

number of ==1 rows which need to be clean

Definition at line 89 of file CbcBranchToFixLots.hpp.

7.10.4.7 `bool CbcBranchToFixLots::alwaysCreate_ [protected]`

If true then always create branch.

Definition at line 91 of file CbcBranchToFixLots.hpp.

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

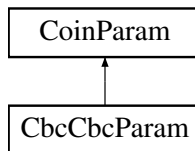
- [/home/ted/COIN/trunk/Cbc/src/CbcBranchToFixLots.hpp](#)

## 7.11 CbcCbcParam Class Reference

Class for control parameters that act on a [CbcModel](#) object.

```
#include <CbcGenCbcParam.hpp>
```

Inheritance diagram for CbcCbcParam:



### Public Types

#### Subtypes

- enum [CbcCbcParamCode](#) {  
[CBCCBC\\_FIRSTPARAM](#) = CbcGenParam::CBCGEN\_LASTPARAM + 1, [ALLOWABLEGAP](#), [COSTSTRATEGY](#), [CUTDEPTH](#),  
[CUTOFF](#), [CUTPASS](#), [DIRECTION](#), [GAPRATIO](#),  
[INCREMENT](#), [INFEASIBILITYWEIGHT](#), [INTEGERTOLERANCE](#), [LOGLEVEL](#),  
[MAXIMIZE](#), [MAXNODES](#), [MINIMIZE](#), [MIOPTIONS](#),  
[MOREMIOPTIONS](#), [NUMBERANALYZE](#), [NUMBERBEFORE](#), [NUMBERMINI](#),  
[STRONGBRANCHING](#), [TIMELIMIT\\_BAB](#), [CBCCBC\\_LASTPARAM](#) }  
*Enumeration for parameters that control a [CbcModel](#) object.*

### Public Member Functions

#### Constructors and Destructors

*Be careful how you specify parameters for the constructors! There's great potential for confusion.*

- [CbcCbcParam](#) ()  
*Default constructor.*
- [CbcCbcParam](#) ([CbcCbcParamCode](#) code, 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.*
- [CbcCbcParam](#) ([CbcCbcParamCode](#) code, std::string name, std::string help, int lower, int upper, int dflt=0, bool display=true)  
*Constructor for a parameter with an integer value.*
- [CbcCbcParam](#) ([CbcCbcParamCode](#) code, std::string name, std::string help, std::string firstValue, int dflt, bool display=true)  
*Constructor for a parameter with keyword values.*
- [CbcCbcParam](#) ([CbcCbcParamCode](#) code, std::string name, std::string help, std::string dflt, bool display=true)  
*Constructor for a string parameter.*
- [CbcCbcParam](#) ([CbcCbcParamCode](#) code, std::string name, std::string help, bool display=true)  
*Constructor for an action parameter.*
- [CbcCbcParam](#) (const [CbcCbcParam](#) &orig)  
*Copy constructor.*
- [CbcCbcParam](#) \* [clone](#) ()  
*Clone.*
- [CbcCbcParam](#) & [operator=](#) (const [CbcCbcParam](#) &rhs)  
*Assignment.*
- [~CbcCbcParam](#) ()  
*Destructor.*

#### Methods to query and manipulate a parameter object

- [CbcCbcParamCode](#) [paramCode](#) () const  
*Get the parameter code.*
- void [setParamCode](#) ([CbcCbcParamCode](#) code)  
*Set the parameter code.*
- [CbcModel](#) \* [obj](#) () const  
*Get the underlying [CbcModel](#) object.*
- void [setObj](#) ([CbcModel](#) \*obj)  
*Set the underlying [CbcModel](#) object.*

##### 7.11.1 Detailed Description

Class for control parameters that act on a [CbcModel](#) object.

Adds parameter type codes and push/pull functions to the generic parameter object.

Definition at line 31 of file CbcGenCbcParam.hpp.

##### 7.11.2 Member Enumeration Documentation

###### 7.11.2.1 enum CbcCbcParam::CbcCbcParamCode

Enumeration for parameters that control a [CbcModel](#) object.

These are parameters that control the operation of a [CbcModel](#) object. CBCCBC\_FIRSTPARAM and CBCCBC\_LASTPARAM are markers to allow convenient separation of parameter groups.

Enumerator

**CBCCBC\_FIRSTPARAM**

**ALLOWABLEGAP**  
**COSTSTRATEGY**  
**CUTDEPTH**  
**CUTOFF**  
**CUTPASS**  
**DIRECTION**  
**GAPRATIO**  
**INCREMENT**  
**INFEASIBILITYWEIGHT**  
**INTEGERTOLERANCE**  
**LOGLEVEL**  
**MAXIMIZE**  
**MAXNODES**  
**MINIMIZE**  
**MIOPTIONS**  
**MOREMIOPTIONS**  
**NUMBERANALYZE**  
**NUMBERBEFORE**  
**NUMBERMINI**  
**STRONGBRANCHING**  
**TIMELIMIT\_BAB**  
**CBCCBC\_LASTPARAM**

Definition at line 45 of file CbcGenCbcParam.hpp.

### 7.11.3 Constructor & Destructor Documentation

#### 7.11.3.1 CbcCbcParam::CbcCbcParam ( )

Default constructor.

#### 7.11.3.2 CbcCbcParam::CbcCbcParam ( CbcCbcParamCode *code*, 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.

#### 7.11.3.3 CbcCbcParam::CbcCbcParam ( CbcCbcParamCode *code*, 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.



7.11.3.4 **CbcCbcParam::CbcCbcParam** ( **CbcCbcParamCode** *code*, **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 keyword. Additional keywords can be added using `appendKwd()`. Keywords are numbered from zero. It's necessary to specify both the first keyword (*firstValue*) and the default keyword index (*dflt*) in order to distinguish this constructor from the string and action parameter constructors.

7.11.3.5 **CbcCbcParam::CbcCbcParam** ( **CbcCbcParamCode** *code*, **std::string** *name*, **std::string** *help*, **std::string** *dflt*, **bool** *display* = **true** )

Constructor for a string parameter.

The default string value must be specified explicitly to distinguish a string constructor from an action parameter constructor.

7.11.3.6 **CbcCbcParam::CbcCbcParam** ( **CbcCbcParamCode** *code*, **std::string** *name*, **std::string** *help*, **bool** *display* = **true** )

Constructor for an action parameter.

7.11.3.7 **CbcCbcParam::CbcCbcParam** ( **const CbcCbcParam** & *orig* )

Copy constructor.

7.11.3.8 **CbcCbcParam::~~CbcCbcParam** ( )

Destructor.

#### 7.11.4 Member Function Documentation

7.11.4.1 **CbcCbcParam\*** **CbcCbcParam::clone** ( )

Clone.

7.11.4.2 **CbcCbcParam&** **CbcCbcParam::operator=** ( **const CbcCbcParam** & *rhs* )

Assignment.

7.11.4.3 **CbcCbcParamCode** **CbcCbcParam::paramCode** ( ) **const** [inline]

Get the parameter code.

Definition at line 139 of file `CbcGenCbcParam.hpp`.

7.11.4.4 **void** **CbcCbcParam::setParamCode** ( **CbcCbcParamCode** *code* ) [inline]

Set the parameter code.

Definition at line 145 of file `CbcGenCbcParam.hpp`.

7.11.4.5 **CbcModel\*** **CbcCbcParam::obj** ( ) **const** [inline]

Get the underlying [CbcModel](#) object.

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

#### 7.11.4.6 void CbcCbcParam::setObj ( CbcModel \* obj ) [inline]

Set the underlying [CbcModel](#) object.

Definition at line 157 of file CbcGenCbcParam.hpp.

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

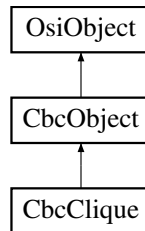
- </home/ted/COIN/trunk/Cbc/src/CbcGenCbcParam.hpp>

## 7.12 CbcClique Class Reference

Branching object for cliques.

```
#include <CbcClique.hpp>
```

Inheritance diagram for CbcClique:



### Public Member Functions

- [CbcClique](#) ()  
*Default Constructor.*
- [CbcClique](#) ([CbcModel](#) \*model, int cliqueType, int numberMembers, const int \*which, const char \*type, int identifier, int slack=-1)  
*Useful constructor (which are integer indices) slack can denote a slack in set.*
- [CbcClique](#) (const [CbcClique](#) &)  
*Copy constructor.*
- virtual [CbcObject](#) \* clone () const  
*Clone.*
- [CbcClique](#) & operator= (const [CbcClique](#) &rhs)  
*Assignment operator.*
- virtual ~[CbcClique](#) ()  
*Destructor.*
- virtual double infeasibility (const OsiBranchingInformation \*info, int &preferredWay) const  
*Infeasibility - large is 0.5.*
- virtual void feasibleRegion ()  
*This looks at solution and sets bounds to contain solution.*
- virtual [CbcBranchingObject](#) \* createCbcBranch (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way)  
*Creates a branching object.*
- int numberMembers () const  
*Number of members.*

- int [numberNonSOSMembers](#) () const  
*Number of variables with -1 coefficient.*
- const int \* [members](#) () const  
*Members (indices in range 0 ... numberIntegers\_-1)*
- char [type](#) (int index) const  
*Type of each member, i.e., which way is strong.*
- int [cliqueType](#) () const  
*Clique type: 0 is <=, 1 is ==.*
- virtual void [redoSequenceEtc](#) (CbcModel \*model, int numberColumns, const int \*originalColumns)  
*Redoes data when sequence numbers change.*

#### Protected Attributes

- int [numberMembers\\_](#)  
*data Number of members*
- int [numberNonSOSMembers\\_](#)  
*Number of Non SOS members i.e. fixing to zero is strong.*
- int \* [members\\_](#)  
*Members (indices in range 0 ... numberIntegers\_-1)*
- char \* [type\\_](#)  
*Strong value for each member.*
- int [cliqueType\\_](#)  
*Clique type.*
- int [slack\\_](#)  
*Slack variable for the clique.*

#### 7.12.1 Detailed Description

Branching object for cliques.

A clique is defined to be a set of binary variables where fixing any one variable to its 'strong' value fixes all other variables. An example is the most common SOS1 construction: a set of binary variables  $x_j$  s.t.  $\sum\{x_j\} = 1$ . Setting any one variable to 1 forces all other variables to 0. (See comments for [CbcSOS](#) below.)

Other configurations are possible, however: Consider  $x_1 - x_2 + x_3 \leq 0$ . Setting  $x_1$  ( $x_3$ ) to 1 forces  $x_2$  to 1 and  $x_3$  ( $x_1$ ) to 0. Setting  $x_2$  to 0 forces  $x_1$  and  $x_3$  to 0.

The proper point of view to take when interpreting [CbcClique](#) is 'generalisation of SOS1 on binary variables.' To get into the proper frame of mind, here's an example.

Consider the following sequence, where  $x_j = (1 - y_j)$ :

$x_1 + x_2 + x_3 \leq 1$	all strong at 1
$x_1 - y_2 + x_3 \leq 0$	$y_2$ strong at 0; $x_1, x_3$ strong at 1
$-y_1 - y_2 + x_3 \leq -1$	$y_1, y_2$ strong at 0, $x_3$ strong at 1
$-y_1 - y_2 - y_3 \leq -2$	all strong at 0

The first line is a standard SOS1 on binary variables.

Variables with +1 coefficients are 'SOS-style' and variables with -1 coefficients are 'non-SOS-style'. So [numberNonSOSMembers\\_](#) simply tells you how many variables have -1 coefficients. The implicit rhs for a clique is  $1 - \text{numberNonSOSMembers\_}$ .

Definition at line 41 of file CbcClique.hpp.

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 `CbcClique::CbcClique ( )`

Default Constructor.

#### 7.12.2.2 `CbcClique::CbcClique ( CbcModel * model, int cliqueType, int numberMembers, const int * which, const char * type, int identifier, int slack = -1 )`

Useful constructor (which are integer indices) slack can denote a slack in set.

If type == NULL then as if 1

#### 7.12.2.3 `CbcClique::CbcClique ( const CbcClique & )`

Copy constructor.

#### 7.12.2.4 `virtual CbcClique::~~CbcClique ( ) [virtual]`

Destructor.

### 7.12.3 Member Function Documentation

#### 7.12.3.1 `virtual CbcObject* CbcClique::clone ( ) const [virtual]`

Clone.

Implements [CbcObject](#).

#### 7.12.3.2 `CbcClique& CbcClique::operator= ( const CbcClique & rhs )`

Assignment operator.

#### 7.12.3.3 `virtual double CbcClique::infeasibility ( const OsiBranchingInformation * info, int & preferredWay ) const [virtual]`

Infeasibility - large is 0.5.

Reimplemented from [CbcObject](#).

#### 7.12.3.4 `virtual void CbcClique::feasibleRegion ( ) [virtual]`

This looks at solution and sets bounds to contain solution.

Implements [CbcObject](#).

#### 7.12.3.5 `virtual CbcBranchingObject* CbcClique::createCbcBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way ) [virtual]`

Creates a branching object.

Reimplemented from [CbcObject](#).

#### 7.12.3.6 `int CbcClique::numberMembers ( ) const [inline]`

Number of members.

Definition at line 78 of file CbcClique.hpp.

7.12.3.7 `int CbcCliques::numberNonSOSMembers ( ) const [inline]`

Number of variables with -1 coefficient.

Number of non-SOS members, i.e., fixing to zero is strong. See comments at head of class, and comments for [type\\_](#).

Definition at line 86 of file CbcCliques.hpp.

7.12.3.8 `const int* CbcCliques::members ( ) const [inline]`

Members (indices in range 0 ... numberIntegers\_-1)

Definition at line 91 of file CbcCliques.hpp.

7.12.3.9 `char CbcCliques::type ( int index ) const [inline]`

Type of each member, i.e., which way is strong.

This also specifies whether a variable has a +1 or -1 coefficient.

- 0 => -1 coefficient, 0 is strong value
- 1 => +1 coefficient, 1 is strong value If unspecified, all coefficients are assumed to be positive.

Indexed as 0 .. numberMembers\_-1

Definition at line 104 of file CbcCliques.hpp.

7.12.3.10 `int CbcCliques::cliqueType ( ) const [inline]`

Clique type: 0 is <=, 1 is ==.

Definition at line 110 of file CbcCliques.hpp.

7.12.3.11 `virtual void CbcCliques::redoSequenceEtc ( CbcModel * model, int numberColumns, const int * originalColumns ) [virtual]`

Redoes data when sequence numbers change.

Reimplemented from [CbcObject](#).

## 7.12.4 Member Data Documentation

7.12.4.1 `int CbcCliques::numberMembers_ [protected]`

data Number of members

Definition at line 119 of file CbcCliques.hpp.

7.12.4.2 `int CbcCliques::numberNonSOSMembers_ [protected]`

Number of Non SOS members i.e. fixing to zero is strong.

Definition at line 122 of file CbcCliques.hpp.

7.12.4.3 `int* CbcCliques::members_ [protected]`

Members (indices in range 0 ... numberIntegers\_-1)

Definition at line 125 of file CbcCliques.hpp.

#### 7.12.4.4 `char* CbcClique::type_` [protected]

Strong value for each member.

This also specifies whether a variable has a +1 or -1 coefficient.

- 0 => -1 coefficient, 0 is strong value
- 1 => +1 coefficient, 1 is strong value If unspecified, all coefficients are assumed to be positive.

Indexed as 0 .. numberMembers\_-1

Definition at line 136 of file CbcClique.hpp.

#### 7.12.4.5 `int CbcClique::cliqueType_` [protected]

Clique type.

0 defines a  $\leq$  relation, 1 an equality. The assumed value of the rhs is numberNonSOSMembers\_+1. (See comments for the class.)

Definition at line 143 of file CbcClique.hpp.

#### 7.12.4.6 `int CbcClique::slack_` [protected]

Slack variable for the clique.

Identifies the slack variable for the clique (typically added to convert a  $\leq$  relation to an equality). Value is sequence number within clique members.

Definition at line 151 of file CbcClique.hpp.

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

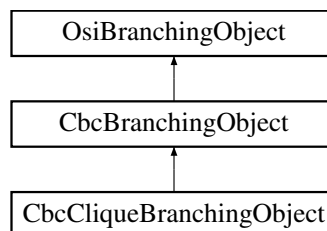
- </home/ted/COIN/trunk/Cbc/src/CbcClique.hpp>

### 7.13 CbcCliqueBranchingObject Class Reference

Branching object for unordered cliques.

```
#include <CbcClique.hpp>
```

Inheritance diagram for CbcCliqueBranchingObject:



#### Public Member Functions

- [CbcCliqueBranchingObject](#) ()
- [CbcCliqueBranchingObject](#) ([CbcModel](#) \*model, const [CbcClique](#) \*clique, int way, int numberOnDownSide, const int \*down, int numberOnUpSide, const int \*up)

- [CbcCliqueBranchingObject](#) (const [CbcCliqueBranchingObject](#) &)
- [CbcCliqueBranchingObject](#) & operator= (const [CbcCliqueBranchingObject](#) &rhs)
- virtual [CbcBranchingObject](#) \* clone () const  
*Clone.*
- virtual ~[CbcCliqueBranchingObject](#) ()
- virtual double [branch](#) ()  
*Does next branch and updates state.*
- virtual void [print](#) ()  
*Print something about branch - only if log level high.*
- virtual [CbcBranchObjType](#) type () const  
*Return the type (an integer identifier) of this.*
- virtual int [compareOriginalObject](#) (const [CbcBranchingObject](#) \*brObj) const  
*Compare the original object of this with the original object of brObj.*
- virtual [CbcRangeCompare](#) [compareBranchingObject](#) (const [CbcBranchingObject](#) \*brObj, const bool replacelf-Overlap=false)  
*Compare the this with brObj.*

#### Additional Inherited Members

##### 7.13.1 Detailed Description

Branching object for unordered cliques.

Intended for cliques which are long enough to make it worthwhile but  $\leq 64$  members. There will also be ones for long cliques.

Variable\_ is the clique id number (redundant, as the object also holds a pointer to the clique).

Definition at line 162 of file CbcClique.hpp.

##### 7.13.2 Constructor & Destructor Documentation

7.13.2.1 [CbcCliqueBranchingObject::CbcCliqueBranchingObject](#) ( )

7.13.2.2 [CbcCliqueBranchingObject::CbcCliqueBranchingObject](#) ( [CbcModel](#) \* model, const [CbcClique](#) \* clique, int way, int numberOnDownSide, const int \* down, int numberOnUpSide, const int \* up )

7.13.2.3 [CbcCliqueBranchingObject::CbcCliqueBranchingObject](#) ( const [CbcCliqueBranchingObject](#) & )

7.13.2.4 virtual [CbcCliqueBranchingObject::~CbcCliqueBranchingObject](#) ( ) [virtual]

##### 7.13.3 Member Function Documentation

7.13.3.1 [CbcCliqueBranchingObject& CbcCliqueBranchingObject::operator=](#) ( const [CbcCliqueBranchingObject](#) & rhs )

7.13.3.2 virtual [CbcBranchingObject\\*](#) [CbcCliqueBranchingObject::clone](#) ( ) const [virtual]

Clone.

Implements [CbcBranchingObject](#).

### 7.13.3.3 `virtual double CbcCliquesBranchingObject::branch ( ) [virtual]`

Does next branch and updates state.

Implements [CbcBranchingObject](#).

### 7.13.3.4 `virtual void CbcCliquesBranchingObject::print ( ) [virtual]`

Print something about branch - only if log level high.

### 7.13.3.5 `virtual CbcBranchObjType CbcCliquesBranchingObject::type ( ) const [inline],[virtual]`

Return the type (an integer identifier) of `this`.

Implements [CbcBranchingObject](#).

Definition at line 197 of file `CbcCliques.hpp`.

### 7.13.3.6 `virtual int CbcCliquesBranchingObject::compareOriginalObject ( const CbcBranchingObject * brObj ) const [virtual]`

Compare the original object of `this` with the original object of `brObj`.

Assumes that there is an ordering of the original objects. This method should be invoked only if `this` and `brObj` are of the same type. Return negative/0/positive depending on whether `this` is smaller/same/larger than the argument.

Reimplemented from [CbcBranchingObject](#).

### 7.13.3.7 `virtual CbcRangeCompare CbcCliquesBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false ) [virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

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

- `/home/ted/COIN/trunk/Cbc/src/CbcCliques.hpp`

## 7.14 CbcCompare Class Reference

```
#include <CbcCompare.hpp>
```

### Public Member Functions

- [CbcCompare](#) ( )
- `virtual ~CbcCompare` ( )
- `bool operator()` ( [CbcNode](#) \*x, [CbcNode](#) \*y )
- `bool compareNodes` ( [CbcNode](#) \*x, [CbcNode](#) \*y )
- `bool alternateTest` ( [CbcNode](#) \*x, [CbcNode](#) \*y )  
*This is alternate test function.*
- [CbcCompareBase](#) \* `comparisonObject` ( ) const  
*return comparison object*



## Public Attributes

- [CbcCompareBase](#) \* [test\\_](#)

## 7.14.1 Detailed Description

Definition at line 11 of file CbcCompare.hpp.

## 7.14.2 Constructor &amp; Destructor Documentation

## 7.14.2.1 CbcCompare::CbcCompare ( ) [inline]

Definition at line 15 of file CbcCompare.hpp.

## 7.14.2.2 virtual CbcCompare::~~CbcCompare ( ) [inline], [virtual]

Definition at line 19 of file CbcCompare.hpp.

## 7.14.3 Member Function Documentation

## 7.14.3.1 bool CbcCompare::operator() ( CbcNode \* x, CbcNode \* y ) [inline]

Definition at line 21 of file CbcCompare.hpp.

## 7.14.3.2 bool CbcCompare::compareNodes ( CbcNode \* x, CbcNode \* y ) [inline]

Definition at line 24 of file CbcCompare.hpp.

## 7.14.3.3 bool CbcCompare::alternateTest ( CbcNode \* x, CbcNode \* y ) [inline]

This is alternate test function.

Definition at line 28 of file CbcCompare.hpp.

## 7.14.3.4 CbcCompareBase\* CbcCompare::comparisonObject ( ) const [inline]

return comparison object

Definition at line 33 of file CbcCompare.hpp.

## 7.14.4 Member Data Documentation

## 7.14.4.1 CbcCompareBase\* CbcCompare::test\_

Definition at line 13 of file CbcCompare.hpp.

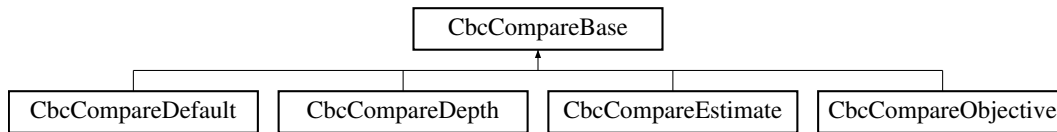
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Cbc/src/[CbcCompare.hpp](#)

## 7.15 CbcCompareBase Class Reference

```
#include <CbcCompareBase.hpp>
```

Inheritance diagram for CbcCompareBase:



### Public Member Functions

- [CbcCompareBase](#) ()
- virtual bool [newSolution](#) ([CbcModel](#) \*)  
*Reconsider behaviour after discovering a new solution.*
- virtual bool [newSolution](#) ([CbcModel](#) \*, double, int)  
*Reconsider behaviour after discovering a new solution.*
- virtual bool [every1000Nodes](#) ([CbcModel](#) \*, int)
- virtual bool [fullScan](#) () const  
*Returns true if wants code to do scan with alternate criterion NOTE - this is temporarily disabled.*
- virtual [~CbcCompareBase](#) ()
- virtual void [generateCpp](#) (FILE \*)  
*Create C++ lines to get to current state.*
- [CbcCompareBase](#) (const [CbcCompareBase](#) &rhs)
- [CbcCompareBase](#) & [operator=](#) (const [CbcCompareBase](#) &rhs)
- virtual [CbcCompareBase](#) \* [clone](#) () const  
*Clone.*
- virtual bool [test](#) ([CbcNode](#) \*, [CbcNode](#) \*)  
*This is test function.*
- virtual bool [alternateTest](#) ([CbcNode](#) \*x, [CbcNode](#) \*y)  
*This is alternate test function.*
- bool [operator\(\)](#) ([CbcNode](#) \*x, [CbcNode](#) \*y)
- bool [equalityTest](#) ([CbcNode](#) \*x, [CbcNode](#) \*y) const  
*Further test if everything else equal.*
- void [sayThreaded](#) ()  
*Say threaded.*

### Protected Attributes

- [CbcCompareBase](#) \* [test\\_](#)
- bool [threaded\\_](#)

#### 7.15.1 Detailed Description

Definition at line 27 of file CbcCompareBase.hpp.

#### 7.15.2 Constructor & Destructor Documentation

##### 7.15.2.1 CbcCompareBase::CbcCompareBase ( ) [inline]

Definition at line 30 of file CbcCompareBase.hpp.

7.15.2.2 `virtual CbcCompareBase::~CbcCompareBase ( ) [inline],[virtual]`

Definition at line 75 of file CbcCompareBase.hpp.

7.15.2.3 `CbcCompareBase::CbcCompareBase ( const CbcCompareBase & rhs ) [inline]`

Definition at line 80 of file CbcCompareBase.hpp.

### 7.15.3 Member Function Documentation

7.15.3.1 `virtual bool CbcCompareBase::newSolution ( CbcModel * ) [inline],[virtual]`

Reconsider behaviour after discovering a new solution.

This allows any method to change its behaviour. It is called after each solution.

The method should return true if changes are made which will alter the evaluation criteria applied to a node. (So that in cases where the search tree is sorted, it can be properly rebuilt.)

Definition at line 45 of file CbcCompareBase.hpp.

7.15.3.2 `virtual bool CbcCompareBase::newSolution ( CbcModel *, double, int ) [inline],[virtual]`

Reconsider behaviour after discovering a new solution.

This allows any method to change its behaviour. It is called after each solution.

The method should return true if changes are made which will alter the evaluation criteria applied to a node. (So that in cases where the search tree is sorted, it can be properly rebuilt.)

Reimplemented in [CbcCompareDefault](#).

Definition at line 57 of file CbcCompareBase.hpp.

7.15.3.3 `virtual bool CbcCompareBase::every1000Nodes ( CbcModel *, int ) [inline],[virtual]`

Reimplemented in [CbcCompareDefault](#).

Definition at line 64 of file CbcCompareBase.hpp.

7.15.3.4 `virtual bool CbcCompareBase::fullScan ( ) const [inline],[virtual]`

Returns true if wants code to do scan with alternate criterion NOTE - this is temporarily disabled.

Definition at line 71 of file CbcCompareBase.hpp.

7.15.3.5 `virtual void CbcCompareBase::generateCpp ( FILE * ) [inline],[virtual]`

Create C++ lines to get to current state.

Reimplemented in [CbcCompareDefault](#), [CbcCompareEstimate](#), [CbcCompareObjective](#), and [CbcCompareDepth](#).

Definition at line 77 of file CbcCompareBase.hpp.

7.15.3.6 `CbcCompareBase& CbcCompareBase::operator= ( const CbcCompareBase & rhs ) [inline]`

Definition at line 86 of file CbcCompareBase.hpp.

7.15.3.7 `virtual CbcCompareBase* CbcCompareBase::clone ( ) const [inline],[virtual]`

Clone.

Reimplemented in [CbcCompareDefault](#), [CbcCompareEstimate](#), [CbcCompareObjective](#), and [CbcCompareDepth](#).

Definition at line 95 of file [CbcCompareBase.hpp](#).

**7.15.3.8** `virtual bool CbcCompareBase::test ( CbcNode * , CbcNode * ) [inline],[virtual]`

This is test function.

Reimplemented in [CbcCompareDefault](#), [CbcCompareObjective](#), [CbcCompareDepth](#), and [CbcCompareEstimate](#).

Definition at line 101 of file [CbcCompareBase.hpp](#).

**7.15.3.9** `virtual bool CbcCompareBase::alternateTest ( CbcNode * x, CbcNode * y ) [inline],[virtual]`

This is alternate test function.

Definition at line 106 of file [CbcCompareBase.hpp](#).

**7.15.3.10** `bool CbcCompareBase::operator() ( CbcNode * x, CbcNode * y ) [inline]`

Definition at line 110 of file [CbcCompareBase.hpp](#).

**7.15.3.11** `bool CbcCompareBase::equalityTest ( CbcNode * x, CbcNode * y ) const [inline]`

Further test if everything else equal.

Definition at line 114 of file [CbcCompareBase.hpp](#).

**7.15.3.12** `void CbcCompareBase::sayThreaded ( ) [inline]`

Say threaded.

Definition at line 132 of file [CbcCompareBase.hpp](#).

## 7.15.4 Member Data Documentation

**7.15.4.1** `CbcCompareBase* CbcCompareBase::test_ [protected]`

Definition at line 136 of file [CbcCompareBase.hpp](#).

**7.15.4.2** `bool CbcCompareBase::threaded_ [protected]`

Definition at line 138 of file [CbcCompareBase.hpp](#).

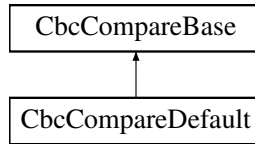
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcCompareBase.hpp](#)

## 7.16 CbcCompareDefault Class Reference

```
#include <CbcCompareDefault.hpp>
```

Inheritance diagram for [CbcCompareDefault](#):



### Public Member Functions

- [CbcCompareDefault](#) ()  
*Default Constructor.*
- [CbcCompareDefault](#) (double weight)  
*Constructor with weight.*
- [CbcCompareDefault](#) (const [CbcCompareDefault](#) &rhs)  
*Copy constructor.*
- [CbcCompareDefault](#) & [operator=](#) (const [CbcCompareDefault](#) &rhs)  
*Assignment operator.*
- virtual [CbcCompareBase](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- [~CbcCompareDefault](#) ()
- virtual bool [test](#) ([CbcNode](#) \*x, [CbcNode](#) \*y)  
*This is test function.*
- virtual bool [newSolution](#) ([CbcModel](#) \*model, double objectiveAtContinuous, int numberInfeasibilitiesAtContinuous)  
*This allows method to change behavior as it is called after each solution.*
- virtual bool [every1000Nodes](#) ([CbcModel](#) \*model, int numberNodes)  
*This allows method to change behavior Return true if want tree re-sorted.*
- double [getWeight](#) () const
- void [setWeight](#) (double weight)
- double [getCutoff](#) () const  
*Cutoff.*
- void [setCutoff](#) (double cutoff)
- double [getBestPossible](#) () const  
*Best possible solution.*
- void [setBestPossible](#) (double bestPossible)
- void [setBreadthDepth](#) (int value)  
*Depth above which want to explore first.*
- void [startDive](#) ([CbcModel](#) \*model)  
*Start dive.*
- void [cleanDive](#) ()  
*Clean up diving (i.e. switch off or prepare)*

## Protected Attributes

- double `weight_`  
*Weight for each infeasibility.*
- double `saveWeight_`  
*Weight for each infeasibility - computed from solution.*
- double `cutoff_`  
*Cutoff.*
- double `bestPossible_`  
*Best possible solution.*
- int `numberSolutions_`  
*Number of solutions.*
- int `treeSize_`  
*Tree size (at last check)*
- int `breadthDepth_`  
*Depth above which want to explore first.*
- int `startNodeNumber_`  
*Chosen node from estimated (-1 is off)*
- int `afterNodeNumber_`  
*Node number when dive started.*
- bool `setupForDiving_`  
*Indicates doing setup for diving.*

### 7.16.1 Detailed Description

Definition at line 31 of file CbcCompareDefault.hpp.

### 7.16.2 Constructor & Destructor Documentation

#### 7.16.2.1 CbcCompareDefault::CbcCompareDefault ( )

Default Constructor.

#### 7.16.2.2 CbcCompareDefault::CbcCompareDefault ( double *weight* )

Constructor with weight.

#### 7.16.2.3 CbcCompareDefault::CbcCompareDefault ( const CbcCompareDefault & *rhs* )

Copy constructor.

#### 7.16.2.4 CbcCompareDefault::~~CbcCompareDefault ( )

### 7.16.3 Member Function Documentation

#### 7.16.3.1 CbcCompareDefault& CbcCompareDefault::operator= ( const CbcCompareDefault & *rhs* )

Assignment operator.

7.16.3.2 `virtual CbcCompareBase* CbcCompareDefault::clone ( ) const` [virtual]

Clone.

Reimplemented from [CbcCompareBase](#).

7.16.3.3 `virtual void CbcCompareDefault::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcCompareBase](#).

7.16.3.4 `virtual bool CbcCompareDefault::test ( CbcNode * , CbcNode * )` [virtual]

This is test function.

Reimplemented from [CbcCompareBase](#).

7.16.3.5 `virtual bool CbcCompareDefault::newSolution ( CbcModel * model, double objectiveAtContinuous, int numberInfeasibilitiesAtContinuous )` [virtual]

This allows method to change behavior as it is called after each solution.

Reimplemented from [CbcCompareBase](#).

7.16.3.6 `virtual bool CbcCompareDefault::every1000Nodes ( CbcModel * model, int numberNodes )` [virtual]

This allows method to change behavior Return true if want tree re-sorted.

Reimplemented from [CbcCompareBase](#).

7.16.3.7 `double CbcCompareDefault::getWeight ( ) const` [inline]

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

7.16.3.8 `void CbcCompareDefault::setWeight ( double weight )` [inline]

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

7.16.3.9 `double CbcCompareDefault::getCutoff ( ) const` [inline]

Cutoff.

Definition at line 75 of file `CbcCompareDefault.hpp`.

7.16.3.10 `void CbcCompareDefault::setCutoff ( double cutoff )` [inline]

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

7.16.3.11 `double CbcCompareDefault::getBestPossible ( ) const` [inline]

Best possible solution.

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

7.16.3.12 `void CbcCompareDefault::setBestPossible ( double bestPossible )` [inline]

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

7.16.3.13 `void CbcCompareDefault::setBreadthDepth ( int value ) [inline]`

Depth above which want to explore first.

Definition at line 89 of file CbcCompareDefault.hpp.

7.16.3.14 `void CbcCompareDefault::startDive ( CbcModel * model )`

Start dive.

7.16.3.15 `void CbcCompareDefault::cleanDive ( )`

Clean up diving (i.e. switch off or prepare)

#### 7.16.4 Member Data Documentation

7.16.4.1 `double CbcCompareDefault::weight_ [protected]`

Weight for each infeasibility.

Definition at line 98 of file CbcCompareDefault.hpp.

7.16.4.2 `double CbcCompareDefault::saveWeight_ [protected]`

Weight for each infeasibility - computed from solution.

Definition at line 100 of file CbcCompareDefault.hpp.

7.16.4.3 `double CbcCompareDefault::cutoff_ [protected]`

Cutoff.

Definition at line 102 of file CbcCompareDefault.hpp.

7.16.4.4 `double CbcCompareDefault::bestPossible_ [protected]`

Best possible solution.

Definition at line 104 of file CbcCompareDefault.hpp.

7.16.4.5 `int CbcCompareDefault::numberSolutions_ [protected]`

Number of solutions.

Definition at line 106 of file CbcCompareDefault.hpp.

7.16.4.6 `int CbcCompareDefault::treeSize_ [protected]`

Tree size (at last check)

Definition at line 108 of file CbcCompareDefault.hpp.

7.16.4.7 `int CbcCompareDefault::breadthDepth_ [protected]`

Depth above which want to explore first.

Definition at line 110 of file CbcCompareDefault.hpp.



## 7.16.4.8 int CbcCompareDefault::startNodeNumber\_ [protected]

Chosen node from estimated (-1 is off)

Definition at line 112 of file CbcCompareDefault.hpp.

## 7.16.4.9 int CbcCompareDefault::afterNodeNumber\_ [protected]

Node number when dive started.

Definition at line 114 of file CbcCompareDefault.hpp.

## 7.16.4.10 bool CbcCompareDefault::setupForDiving\_ [protected]

Indicates doing setup for diving.

Definition at line 116 of file CbcCompareDefault.hpp.

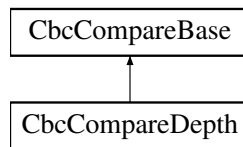
The documentation for this class was generated from the following file:

- /home/ted/COIN/trunk/Cbc/src/CbcCompareDefault.hpp

## 7.17 CbcCompareDepth Class Reference

```
#include <CbcCompareDepth.hpp>
```

Inheritance diagram for CbcCompareDepth:



## Public Member Functions

- [CbcCompareDepth](#) ()
- [~CbcCompareDepth](#) ()
- [CbcCompareDepth](#) (const [CbcCompareDepth](#) &rhs)
- [CbcCompareDepth](#) & [operator=](#) (const [CbcCompareDepth](#) &rhs)
- virtual [CbcCompareBase](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual bool [test](#) ([CbcNode](#) \*x, [CbcNode](#) \*y)  
*This is test function.*

## Additional Inherited Members

## 7.17.1 Detailed Description

Definition at line 25 of file CbcCompareDepth.hpp.

### 7.17.2 Constructor & Destructor Documentation

7.17.2.1 `CbcCompareDepth::CbcCompareDepth ( )`

7.17.2.2 `CbcCompareDepth::~~CbcCompareDepth ( )`

7.17.2.3 `CbcCompareDepth::CbcCompareDepth ( const CbcCompareDepth & rhs )`

### 7.17.3 Member Function Documentation

7.17.3.1 `CbcCompareDepth& CbcCompareDepth::operator= ( const CbcCompareDepth & rhs )`

7.17.3.2 `virtual CbcCompareBase* CbcCompareDepth::clone ( ) const` [virtual]

Clone.

Reimplemented from [CbcCompareBase](#).

7.17.3.3 `virtual void CbcCompareDepth::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcCompareBase](#).

7.17.3.4 `virtual bool CbcCompareDepth::test ( CbcNode *, CbcNode * )` [virtual]

This is test function.

Reimplemented from [CbcCompareBase](#).

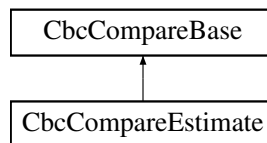
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcCompareDepth.hpp](#)

## 7.18 CbcCompareEstimate Class Reference

```
#include <CbcCompareEstimate.hpp>
```

Inheritance diagram for CbcCompareEstimate:



### Public Member Functions

- [CbcCompareEstimate \( \)](#)
- [~CbcCompareEstimate \( \)](#)
- [CbcCompareEstimate \(const CbcCompareEstimate &rhs\)](#)
- [CbcCompareEstimate & operator= \(const CbcCompareEstimate &rhs\)](#)
- `virtual CbcCompareBase * clone ( ) const`  
Clone.
- `virtual void generateCpp (FILE *fp)`

Create C++ lines to get to current state.

- virtual bool `test (CbcNode *x, CbcNode *y)`

This is test function.

## Additional Inherited Members

### 7.18.1 Detailed Description

Definition at line 27 of file CbcCompareEstimate.hpp.

### 7.18.2 Constructor & Destructor Documentation

7.18.2.1 `CbcCompareEstimate::CbcCompareEstimate ( )`

7.18.2.2 `CbcCompareEstimate::~~CbcCompareEstimate ( )`

7.18.2.3 `CbcCompareEstimate::CbcCompareEstimate ( const CbcCompareEstimate & rhs )`

### 7.18.3 Member Function Documentation

7.18.3.1 `CbcCompareEstimate& CbcCompareEstimate::operator= ( const CbcCompareEstimate & rhs )`

7.18.3.2 `virtual CbcCompareBase* CbcCompareEstimate::clone ( ) const` [virtual]

Clone.

Reimplemented from [CbcCompareBase](#).

7.18.3.3 `virtual void CbcCompareEstimate::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcCompareBase](#).

7.18.3.4 `virtual bool CbcCompareEstimate::test ( CbcNode *, CbcNode * )` [virtual]

This is test function.

Reimplemented from [CbcCompareBase](#).

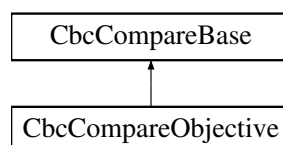
The documentation for this class was generated from the following file:

- `/home/ted/COIN/trunk/Cbc/src/CbcCompareEstimate.hpp`

## 7.19 CbcCompareObjective Class Reference

```
#include <CbcCompareObjective.hpp>
```

Inheritance diagram for CbcCompareObjective:



## Public Member Functions

- [CbcCompareObjective](#) ()
- virtual [~CbcCompareObjective](#) ()
- [CbcCompareObjective](#) (const [CbcCompareObjective](#) &rhs)
- [CbcCompareObjective](#) & operator= (const [CbcCompareObjective](#) &rhs)
- virtual [CbcCompareBase](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual bool [test](#) ([CbcNode](#) \*x, [CbcNode](#) \*y)  
*This is test function.*

## Additional Inherited Members

### 7.19.1 Detailed Description

Definition at line 26 of file [CbcCompareObjective.hpp](#).

### 7.19.2 Constructor & Destructor Documentation

#### 7.19.2.1 [CbcCompareObjective::CbcCompareObjective](#) ( )

#### 7.19.2.2 virtual [CbcCompareObjective::~CbcCompareObjective](#) ( ) [virtual]

#### 7.19.2.3 [CbcCompareObjective::CbcCompareObjective](#) ( const [CbcCompareObjective](#) & rhs )

### 7.19.3 Member Function Documentation

#### 7.19.3.1 [CbcCompareObjective& CbcCompareObjective::operator=](#) ( const [CbcCompareObjective](#) & rhs )

#### 7.19.3.2 virtual [CbcCompareBase](#)\* [CbcCompareObjective::clone](#) ( ) const [virtual]

Clone.

Reimplemented from [CbcCompareBase](#).

#### 7.19.3.3 virtual void [CbcCompareObjective::generateCpp](#) ( FILE \* fp ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcCompareBase](#).

#### 7.19.3.4 virtual bool [CbcCompareObjective::test](#) ( [CbcNode](#) \*, [CbcNode](#) \* ) [virtual]

This is test function.

Reimplemented from [CbcCompareBase](#).

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

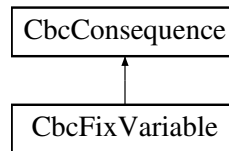
- [/home/ted/COIN/trunk/Cbc/src/CbcCompareObjective.hpp](#)

## 7.20 CbcConsequence Class Reference

Abstract base class for consequent bounds.

```
#include <CbcConsequence.hpp>
```

Inheritance diagram for CbcConsequence:



### Public Member Functions

- [CbcConsequence](#) ()
- [CbcConsequence](#) (const [CbcConsequence](#) &rhs)
- [CbcConsequence](#) & [operator=](#) (const [CbcConsequence](#) &rhs)
- virtual [CbcConsequence](#) \* [clone](#) () const =0  
*Clone.*
- virtual [~CbcConsequence](#) ()  
*Destructor.*
- virtual void [applyToSolver](#) (OsiSolverInterface \*solver, int state) const =0  
*Apply to an LP solver.*

### 7.20.1 Detailed Description

Abstract base class for consequent bounds.

When a variable is branched on it normally interacts with other variables by means of equations. There are cases where we want to step outside LP and do something more directly e.g. fix bounds. This class is for that.

At present it need not be virtual as only instance is [CbcFixVariable](#), but ...

Definition at line 22 of file CbcConsequence.hpp.

### 7.20.2 Constructor & Destructor Documentation

7.20.2.1 [CbcConsequence::CbcConsequence](#) ( )

7.20.2.2 [CbcConsequence::CbcConsequence](#) ( const [CbcConsequence](#) & rhs )

7.20.2.3 [virtual CbcConsequence::~~CbcConsequence](#) ( ) [virtual]

Destructor.

### 7.20.3 Member Function Documentation

7.20.3.1 [CbcConsequence& CbcConsequence::operator=](#) ( const [CbcConsequence](#) & rhs )

7.20.3.2 `virtual CbcConsequence* CbcConsequence::clone ( ) const` [pure virtual]

Clone.

Implemented in [CbcFixVariable](#).

7.20.3.3 `virtual void CbcConsequence::applyToSolver ( OsiSolverInterface * solver, int state ) const` [pure virtual]

Apply to an LP solver.

Action depends on state

Implemented in [CbcFixVariable](#).

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

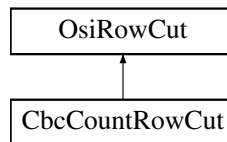
- [/home/ted/COIN/trunk/Cbc/src/CbcConsequence.hpp](#)

## 7.21 CbcCountRowCut Class Reference

OsiRowCut augmented with bookkeeping.

`#include <CbcCountRowCut.hpp>`

Inheritance diagram for CbcCountRowCut:



### Public Member Functions

- void [increment](#) (int change=1)  
*Increment the number of references.*
- int [decrement](#) (int change=1)  
*Decrement the number of references and return the number left.*
- void [setInfo](#) (CbcNodeInfo \*, int whichOne)  
*Set the information associating this cut with a node.*
- int [numberPointingToThis](#) ()  
*Number of other CbcNodeInfo objects pointing to this row cut.*
- int [whichCutGenerator](#) () const  
*Which generator for cuts - as user order.*
- bool [canDropCut](#) (const OsiSolverInterface \*solver, int row) const  
*Returns true if can drop cut if slack basic.*

### Constructors & destructors

- [CbcCountRowCut](#) ()  
*Default Constructor.*
- [CbcCountRowCut](#) (const OsiRowCut &)  
*'Copy' constructor using an OsiRowCut*

- [CbcCountRowCut](#) (const [OsiRowCut](#) &, [CbcNodeInfo](#) \*, int whichOne, int whichGenerator=-1, int [numberPointingToThis](#)=0)  
*'Copy' constructor using an OsiRowCut and an CbcNodeInfo*
- virtual [~CbcCountRowCut](#) ()  
*Destructor.*

### 7.21.1 Detailed Description

OsiRowCut augmented with bookkeeping.

[CbcCountRowCut](#) is an [OsiRowCut](#) object augmented with bookkeeping information: a reference count and information that specifies the the generator that created the cut and the node to which it's associated.

The general principles for handling the reference count are as follows:

- Once it's determined how the node will branch, increment the reference count under the assumption that all children will use all cuts currently tight at the node and will survive to be placed in the search tree.
- As this assumption is proven incorrect (a cut becomes loose, or a child is fathomed), decrement the reference count accordingly.

When all possible uses of a cut have been demonstrated to be unnecessary, the reference count ([#numberPointingToThis](#)) will fall to zero. The [CbcCountRowCut](#) object (and its included [OsiRowCut](#) object) are then deleted.

Definition at line 35 of file [CbcCountRowCut.hpp](#).

### 7.21.2 Constructor & Destructor Documentation

#### 7.21.2.1 CbcCountRowCut::CbcCountRowCut ( )

Default Constructor.

#### 7.21.2.2 CbcCountRowCut::CbcCountRowCut ( const [OsiRowCut](#) & )

'Copy' constructor using an [OsiRowCut](#)

#### 7.21.2.3 CbcCountRowCut::CbcCountRowCut ( const [OsiRowCut](#) &, [CbcNodeInfo](#) \*, int whichOne, int whichGenerator = -1, int [numberPointingToThis](#) = 0 )

'Copy' constructor using an [OsiRowCut](#) and an [CbcNodeInfo](#)

#### 7.21.2.4 virtual CbcCountRowCut::~CbcCountRowCut ( ) [virtual]

Destructor.

#### Note

The destructor will reach out (via [#owner](#)) and NULL the reference to the cut in the owner's [cuts](#) list.

### 7.21.3 Member Function Documentation

#### 7.21.3.1 void CbcCountRowCut::increment ( int *change* = 1 )

Increment the number of references.

### 7.21.3.2 `int CbcCountRowCut::decrement ( int change = 1 )`

Decrement the number of references and return the number left.

### 7.21.3.3 `void CbcCountRowCut::setInfo ( CbcNodeInfo *, int whichOne )`

Set the information associating this cut with a node.

An [CbcNodeInfo](#) object and an index in the cut set of the node. For locally valid cuts, the node will be the search tree node where the cut was generated. For globally valid cuts, it's the node where the cut was activated.

### 7.21.3.4 `int CbcCountRowCut::numberPointingToThis ( ) [inline]`

Number of other [CbcNodeInfo](#) objects pointing to this row cut.

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

### 7.21.3.5 `int CbcCountRowCut::whichCutGenerator ( ) const [inline]`

Which generator for cuts - as user order.

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

### 7.21.3.6 `bool CbcCountRowCut::canDropCut ( const OsiSolverInterface * solver, int row ) const`

Returns true if can drop cut if slack basic.

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

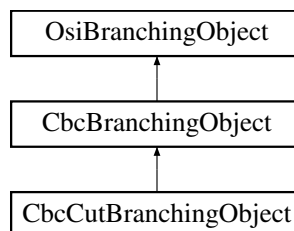
- [/home/ted/COIN/trunk/Cbc/src/CbcCountRowCut.hpp](#)

## 7.22 CbcCutBranchingObject Class Reference

Cut branching object.

```
#include <CbcBranchCut.hpp>
```

Inheritance diagram for `CbcCutBranchingObject`:



### Public Member Functions

- [CbcCutBranchingObject \( \)](#)  
*Default constructor.*
- [CbcCutBranchingObject \(CbcModel \\*\*model\*, OsiRowCut &\*down\*, OsiRowCut &\*up\*, bool \*canFix\*\)](#)  
*Create a cut branching object.*
- [CbcCutBranchingObject \(const CbcCutBranchingObject &\)](#)  
*Copy constructor.*



- `CbcCutBranchingObject & operator= (const CbcCutBranchingObject &rhs)`  
*Assignment operator.*
- `virtual CbcBranchingObject * clone () const`  
*Clone.*
- `virtual ~CbcCutBranchingObject ()`  
*Destructor.*
- `virtual double branch ()`  
*Sets the bounds for variables or adds a cut depending on the current arm of the branch and advances the object state to the next arm.*
- `virtual void print ()`  
*Print something about branch - only if log level high.*
- `virtual bool boundBranch () const`  
*Return true if branch should fix variables.*
- `virtual CbcBranchObjType type () const`  
*Return the type (an integer identifier) of this.*
- `virtual int compareOriginalObject (const CbcBranchingObject *brObj) const`  
*Compare the original object of this with the original object of brObj.*
- `virtual CbcRangeCompare compareBranchingObject (const CbcBranchingObject *brObj, const bool replaceIfOverlap=false)`  
*Compare the this with brObj.*

#### Protected Attributes

- `OsiRowCut down_`  
*Cut for the down arm (way\_ = -1)*
- `OsiRowCut up_`  
*Cut for the up arm (way\_ = 1)*
- `bool canFix_`  
*True if one way can fix variables.*

#### 7.22.1 Detailed Description

Cut branching object.

This object can specify a two-way branch in terms of two cuts

Definition at line 108 of file CbcBranchCut.hpp.

#### 7.22.2 Constructor & Destructor Documentation

##### 7.22.2.1 CbcCutBranchingObject::CbcCutBranchingObject ( )

Default constructor.

##### 7.22.2.2 CbcCutBranchingObject::CbcCutBranchingObject ( CbcModel \* model, OsiRowCut & down, OsiRowCut & up, bool canFix )

Create a cut branching object.

Cut down will applied on way=-1, up on way==1 Assumed down will be first so way\_ set to -1

### 7.22.2.3 `CbcCutBranchingObject::CbcCutBranchingObject ( const CbcCutBranchingObject & )`

Copy constructor.

### 7.22.2.4 `virtual CbcCutBranchingObject::~CbcCutBranchingObject ( ) [virtual]`

Destructor.

## 7.22.3 Member Function Documentation

### 7.22.3.1 `CbcCutBranchingObject& CbcCutBranchingObject::operator= ( const CbcCutBranchingObject & rhs )`

Assignment operator.

### 7.22.3.2 `virtual CbcBranchingObject* CbcCutBranchingObject::clone ( ) const [virtual]`

Clone.

Implements [CbcBranchingObject](#).

### 7.22.3.3 `virtual double CbcCutBranchingObject::branch ( ) [virtual]`

Sets the bounds for variables or adds a cut depending on the current arm of the branch and advances the object state to the next arm.

Returns change in guessed objective on next branch

Implements [CbcBranchingObject](#).

### 7.22.3.4 `virtual void CbcCutBranchingObject::print ( ) [virtual]`

Print something about branch - only if log level high.

### 7.22.3.5 `virtual bool CbcCutBranchingObject::boundBranch ( ) const [virtual]`

Return true if branch should fix variables.

### 7.22.3.6 `virtual CbcBranchObjType CbcCutBranchingObject::type ( ) const [inline],[virtual]`

Return the type (an integer identifier) of `this`.

Implements [CbcBranchingObject](#).

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

### 7.22.3.7 `virtual int CbcCutBranchingObject::compareOriginalObject ( const CbcBranchingObject * brObj ) const [virtual]`

Compare the original object of `this` with the original object of `brObj`.

Assumes that there is an ordering of the original objects. This method should be invoked only if `this` and `brObj` are of the same type. Return negative/0/positive depending on whether `this` is smaller/same/larger than the argument.

Reimplemented from [CbcBranchingObject](#).

### 7.22.3.8 `virtual CbcRangeCompare CbcCutBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false ) [virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

#### 7.22.4 Member Data Documentation

##### 7.22.4.1 `OsiRowCut CbcCutBranchingObject::down_` [protected]

Cut for the down arm (`way_ = -1`)

Definition at line 177 of file `CbcBranchCut.hpp`.

##### 7.22.4.2 `OsiRowCut CbcCutBranchingObject::up_` [protected]

Cut for the up arm (`way_ = 1`)

Definition at line 179 of file `CbcBranchCut.hpp`.

##### 7.22.4.3 `bool CbcCutBranchingObject::canFix_` [protected]

True if one way can fix variables.

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

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

- `/home/ted/COIN/trunk/Cbc/src/CbcBranchCut.hpp`

## 7.23 CbcCutGenerator Class Reference

Interface between Cbc and Cut Generation Library.

```
#include <CbcCutGenerator.hpp>
```

### Public Member Functions

#### Generate Cuts

- `bool generateCuts` (`OsiCuts &cs`, `int fullScan`, `OsiSolverInterface *solver`, [CbcNode](#) \*node)  
*Generate cuts for the client model.*

#### Constructors and destructors

- [CbcCutGenerator](#) ()  
*Default constructor.*
- [CbcCutGenerator](#) ([CbcModel](#) \*model, `CglCutGenerator *generator`, `int howOften=1`, `const char *name=NULL`, `bool normal=true`, `bool atSolution=false`, `bool infeasible=false`, `int howOftenInsub=-100`, `int whatDepth=-1`, `int whatDepthInSub=-1`, `int switchOffIfLessThan=0`)  
*Normal constructor.*
- [CbcCutGenerator](#) (`const CbcCutGenerator &`)  
*Copy constructor.*
- [CbcCutGenerator](#) & `operator=` (`const CbcCutGenerator &rhs`)  
*Assignment operator.*

- `~CbcCutGenerator ()`  
*Destructor.*

### Gets and sets

- void `refreshModel (CbcModel *model)`  
*Set the client model.*
- const char \* `cutGeneratorName ()` const  
*return name of generator*
- void `generateTuning (FILE *fp)`  
*Create C++ lines to show how to tune.*
- void `setHowOften (int value)`  
*Set the cut generation interval.*
- int `howOften ()` const  
*Get the cut generation interval.*
- int `howOftenInSub ()` const  
*Get the cut generation interval.in sub tree.*
- int `inaccuracy ()` const  
*Get level of cut inaccuracy (0 means exact e.g. cliques)*
- void `setInaccuracy (int level)`  
*Set level of cut inaccuracy (0 means exact e.g. cliques)*
- void `setWhatDepth (int value)`  
*Set the cut generation depth.*
- void `setWhatDepthInSub (int value)`  
*Set the cut generation depth in sub tree.*
- int `whatDepth ()` const  
*Get the cut generation depth criterion.*
- int `whatDepthInSub ()` const  
*Get the cut generation depth criterion.in sub tree.*
- void `setMaximumTries (int value)`  
*Set maximum number of times to enter.*
- int `maximumTries ()` const  
*Get maximum number of times to enter.*
- int `switches ()` const  
*Get switches (for debug)*
- bool `normal ()` const  
*Get whether the cut generator should be called in the normal place.*
- void `setNormal (bool value)`  
*Set whether the cut generator should be called in the normal place.*
- bool `atSolution ()` const  
*Get whether the cut generator should be called when a solution is found.*
- void `setAtSolution (bool value)`  
*Set whether the cut generator should be called when a solution is found.*
- bool `whenInfeasible ()` const  
*Get whether the cut generator should be called when the subproblem is found to be infeasible.*
- void `setWhenInfeasible (bool value)`  
*Set whether the cut generator should be called when the subproblem is found to be infeasible.*
- bool `timing ()` const  
*Get whether the cut generator is being timed.*
- void `setTiming (bool value)`  
*Set whether the cut generator is being timed.*
- double `timeInCutGenerator ()` const  
*Return time taken in cut generator.*
- void `incrementTimeInCutGenerator (double value)`
- CglCutGenerator \* `generator ()` const

- *Get the `CglCutGenerator` corresponding to this `CbcCutGenerator`.*
- `int numberTimesEntered () const`  
*Number times cut generator entered.*
- `void setNumberTimesEntered (int value)`
- `void incrementNumberTimesEntered (int value=1)`
- `int numberCutsInTotal () const`  
*Total number of cuts added.*
- `void setNumberCutsInTotal (int value)`
- `void incrementNumberCutsInTotal (int value=1)`
- `int numberElementsInTotal () const`  
*Total number of elements added.*
- `void setNumberElementsInTotal (int value)`
- `void incrementNumberElementsInTotal (int value=1)`
- `int numberColumnCuts () const`  
*Total number of column cuts.*
- `void setNumberColumnCuts (int value)`
- `void incrementNumberColumnCuts (int value=1)`
- `int numberCutsActive () const`  
*Total number of cuts active after (at end of  $n$  cut passes at each node)*
- `void setNumberCutsActive (int value)`
- `void incrementNumberCutsActive (int value=1)`
- `void setSwitchOffIfLessThan (int value)`
- `int switchOffIfLessThan () const`
- `bool needsOptimalBasis () const`  
*Say if optimal basis needed.*
- `void setNeedsOptimalBasis (bool yesNo)`  
*Set if optimal basis needed.*
- `bool mustCallAgain () const`  
*Whether generator MUST be called again if any cuts (i.e. ignore break from loop)*
- `void setMustCallAgain (bool yesNo)`  
*Set whether generator MUST be called again if any cuts (i.e. ignore break from loop)*
- `bool switchedOff () const`  
*Whether generator switched off for moment.*
- `void setSwitchedOff (bool yesNo)`  
*Set whether generator switched off for moment.*
- `bool ineffectualCuts () const`  
*Whether last round of cuts did little.*
- `void setIneffectualCuts (bool yesNo)`  
*Set whether last round of cuts did little.*
- `bool whetherToUse () const`  
*Whether to use if any cuts generated.*
- `void setWhetherToUse (bool yesNo)`  
*Set whether to use if any cuts generated.*
- `bool whetherInMustCallAgainMode () const`  
*Whether in must call again mode (or after others)*
- `void setWhetherInMustCallAgainMode (bool yesNo)`  
*Set whether in must call again mode (or after others)*
- `bool whetherCallAtEnd () const`  
*Whether to call at end.*
- `void setWhetherCallAtEnd (bool yesNo)`  
*Set whether to call at end.*
- `int numberCutsAtRoot () const`  
*Number of cuts generated at root.*
- `void setNumberCutsAtRoot (int value)`
- `int numberActiveCutsAtRoot () const`  
*Number of cuts active at root.*

- void `setNumberActiveCutsAtRoot` (int value)
- int `numberShortCutsAtRoot` () const  
*Number of short cuts at root.*
- void `setNumberShortCutsAtRoot` (int value)
- void `setModel` (`CbcModel` \*model)  
*Set model.*
- bool `globalCutsAtRoot` () const  
*Whether global cuts at root.*
- void `setGlobalCutsAtRoot` (bool yesNo)  
*Set whether global cuts at root.*
- bool `globalCuts` () const  
*Whether global cuts.*
- void `setGlobalCuts` (bool yesNo)  
*Set whether global cuts.*
- void `addStatistics` (const `CbcCutGenerator` \*other)  
*Add in statistics from other.*
- void `scaleBackStatistics` (int factor)  
*Scale back statistics by factor.*

### 7.23.1 Detailed Description

Interface between Cbc and Cut Generation Library.

`CbcCutGenerator` is intended to provide an intelligent interface between Cbc and the cutting plane algorithms in the CGL. A `CbcCutGenerator` is bound to a `CglCutGenerator` and to an `CbcModel`. It contains parameters which control when and how the `generateCuts` method of the `CglCutGenerator` will be called.

The builtin decision criteria available to use when deciding whether to generate cuts are limited: every  $X$  nodes, when a solution is found, and when a subproblem is found to be infeasible. The idea is that the class will grow more intelligent with time.

**Todo** Add a pointer to function member which will allow a client to install their own decision algorithm to decide whether or not to call the CGL `generateCuts` method. Create a default decision method that looks at the builtin criteria.

**Todo** It strikes me as not good that `generateCuts` contains code specific to individual CGL algorithms. Another set of pointer to function members, so that the client can specify the cut generation method as well as pre- and post-generation methods? Taken a bit further, should this class contain a bunch of pointer to function members, one for each of the places where the cut generator might be referenced? Initialization, root node, search tree node, discovery of solution, and termination all come to mind. Initialization and termination would also be useful for instrumenting cbc.

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

### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 `CbcCutGenerator::CbcCutGenerator ( )`

Default constructor.

7.23.2.2 `CbcCutGenerator::CbcCutGenerator ( CbcModel * model, CglCutGenerator * generator, int howOften = 1, const char * name = NULL, bool normal = true, bool atSolution = false, bool infeasible = false, int howOftenInsub = -100, int whatDepth = -1, int whatDepthInSub = -1, int switchOffIfLessThan = 0 )`

Normal constructor.

## 7.23.2.3 CbcCutGenerator::CbcCutGenerator ( const CbcCutGenerator &amp; )

Copy constructor.

## 7.23.2.4 CbcCutGenerator::~CbcCutGenerator ( )

Destructor.

## 7.23.3 Member Function Documentation

## 7.23.3.1 bool CbcCutGenerator::generateCuts ( OsiCuts &amp; cs, int fullScan, OsiSolverInterface \* solver, CbcNode \* node )

Generate cuts for the client model.

Evaluate the state of the client model and decide whether to generate cuts. The generated cuts are inserted into and returned in the collection of cuts `cs`.

If `fullScan` is !=0, the generator is obliged to call the CGL `generateCuts` routine. Otherwise, it is free to make a local decision. Negative `fullScan` says things like at integer solution The current implementation uses `whenCutGenerator_` to decide.

The routine returns true if reoptimisation is needed (because the state of the solver interface has been modified).

If `node` then can find out depth

## 7.23.3.2 CbcCutGenerator&amp; CbcCutGenerator::operator= ( const CbcCutGenerator &amp; rhs )

Assignment operator.

## 7.23.3.3 void CbcCutGenerator::refreshModel ( CbcModel \* model )

Set the client model.

In addition to setting the client model, `refreshModel` also calls the `refreshSolver` method of the `CglCutGenerator` object.

## 7.23.3.4 const char\* CbcCutGenerator::cutGeneratorName ( ) const [inline]

return name of generator

Definition at line 108 of file `CbcCutGenerator.hpp`.

## 7.23.3.5 void CbcCutGenerator::generateTuning ( FILE \* fp )

Create C++ lines to show how to tune.

## 7.23.3.6 void CbcCutGenerator::setHowOften ( int value )

Set the cut generation interval.

Set the number of nodes evaluated between calls to the Cgl object's `generateCuts` routine.

If `value` is positive, cuts will always be generated at the specified interval. If `value` is negative, cuts will initially be generated at the specified interval, but Cbc may adjust the value depending on the success of cuts produced by this generator.

A value of -100 disables the generator, while a value of -99 means just at root.

**7.23.3.7** `int CbcCutGenerator::howOften ( ) const [inline]`

Get the cut generation interval.

Definition at line 131 of file CbcCutGenerator.hpp.

**7.23.3.8** `int CbcCutGenerator::howOftenInSub ( ) const [inline]`

Get the cut generation interval.in sub tree.

Definition at line 135 of file CbcCutGenerator.hpp.

**7.23.3.9** `int CbcCutGenerator::inaccuracy ( ) const [inline]`

Get level of cut inaccuracy (0 means exact e.g. cliques)

Definition at line 139 of file CbcCutGenerator.hpp.

**7.23.3.10** `void CbcCutGenerator::setInaccuracy ( int level ) [inline]`

Set level of cut inaccuracy (0 means exact e.g. cliques)

Definition at line 143 of file CbcCutGenerator.hpp.

**7.23.3.11** `void CbcCutGenerator::setWhatDepth ( int value )`

Set the cut generation depth.

Set the depth criterion for calls to the Cgl object's `generateCuts` routine. Only active if  $> 0$ .

If `whenCutGenerator` is positive and this is positive then this overrides. If `whenCutGenerator` is -1 then this is used as criterion if any cuts were generated at root node. If `whenCutGenerator` is anything else this is ignored.

**7.23.3.12** `void CbcCutGenerator::setWhatDepthInSub ( int value )`

Set the cut generation depth in sub tree.

**7.23.3.13** `int CbcCutGenerator::whatDepth ( ) const [inline]`

Get the cut generation depth criterion.

Definition at line 161 of file CbcCutGenerator.hpp.

**7.23.3.14** `int CbcCutGenerator::whatDepthInSub ( ) const [inline]`

Get the cut generation depth criterion.in sub tree.

Definition at line 165 of file CbcCutGenerator.hpp.

**7.23.3.15** `void CbcCutGenerator::setMaximumTries ( int value ) [inline]`

Set maximum number of times to enter.

Definition at line 169 of file CbcCutGenerator.hpp.

**7.23.3.16** `int CbcCutGenerator::maximumTries ( ) const [inline]`

Get maximum number of times to enter.

Definition at line 172 of file CbcCutGenerator.hpp.



**7.23.3.17** `int CbcCutGenerator::switches ( ) const [inline]`

Get switches (for debug)

Definition at line 176 of file CbcCutGenerator.hpp.

**7.23.3.18** `bool CbcCutGenerator::normal ( ) const [inline]`

Get whether the cut generator should be called in the normal place.

Definition at line 180 of file CbcCutGenerator.hpp.

**7.23.3.19** `void CbcCutGenerator::setNormal ( bool value ) [inline]`

Set whether the cut generator should be called in the normal place.

Definition at line 184 of file CbcCutGenerator.hpp.

**7.23.3.20** `bool CbcCutGenerator::atSolution ( ) const [inline]`

Get whether the cut generator should be called when a solution is found.

Definition at line 189 of file CbcCutGenerator.hpp.

**7.23.3.21** `void CbcCutGenerator::setAtSolution ( bool value ) [inline]`

Set whether the cut generator should be called when a solution is found.

Definition at line 193 of file CbcCutGenerator.hpp.

**7.23.3.22** `bool CbcCutGenerator::whenInfeasible ( ) const [inline]`

Get whether the cut generator should be called when the subproblem is found to be infeasible.

Definition at line 200 of file CbcCutGenerator.hpp.

**7.23.3.23** `void CbcCutGenerator::setWhenInfeasible ( bool value ) [inline]`

Set whether the cut generator should be called when the subproblem is found to be infeasible.

Definition at line 206 of file CbcCutGenerator.hpp.

**7.23.3.24** `bool CbcCutGenerator::timing ( ) const [inline]`

Get whether the cut generator is being timed.

Definition at line 211 of file CbcCutGenerator.hpp.

**7.23.3.25** `void CbcCutGenerator::setTiming ( bool value ) [inline]`

Set whether the cut generator is being timed.

Definition at line 215 of file CbcCutGenerator.hpp.

**7.23.3.26** `double CbcCutGenerator::timeInCutGenerator ( ) const [inline]`

Return time taken in cut generator.

Definition at line 221 of file CbcCutGenerator.hpp.

**7.23.3.27** `void CbcCutGenerator::incrementTimeInCutGenerator ( double value )` `[inline]`

Definition at line 224 of file CbcCutGenerator.hpp.

**7.23.3.28** `CglCutGenerator* CbcCutGenerator::generator ( ) const` `[inline]`

Get the `CglCutGenerator` corresponding to this `CbcCutGenerator`.

Definition at line 228 of file CbcCutGenerator.hpp.

**7.23.3.29** `int CbcCutGenerator::numberOfTimesEntered ( ) const` `[inline]`

Number times cut generator entered.

Definition at line 232 of file CbcCutGenerator.hpp.

**7.23.3.30** `void CbcCutGenerator::setNumberOfTimesEntered ( int value )` `[inline]`

Definition at line 235 of file CbcCutGenerator.hpp.

**7.23.3.31** `void CbcCutGenerator::incrementNumberOfTimesEntered ( int value = 1 )` `[inline]`

Definition at line 238 of file CbcCutGenerator.hpp.

**7.23.3.32** `int CbcCutGenerator::numberOfCutsInTotal ( ) const` `[inline]`

Total number of cuts added.

Definition at line 242 of file CbcCutGenerator.hpp.

**7.23.3.33** `void CbcCutGenerator::setNumberOfCutsInTotal ( int value )` `[inline]`

Definition at line 245 of file CbcCutGenerator.hpp.

**7.23.3.34** `void CbcCutGenerator::incrementNumberOfCutsInTotal ( int value = 1 )` `[inline]`

Definition at line 248 of file CbcCutGenerator.hpp.

**7.23.3.35** `int CbcCutGenerator::numberOfElementsInTotal ( ) const` `[inline]`

Total number of elements added.

Definition at line 252 of file CbcCutGenerator.hpp.

**7.23.3.36** `void CbcCutGenerator::setNumberOfElementsInTotal ( int value )` `[inline]`

Definition at line 255 of file CbcCutGenerator.hpp.

**7.23.3.37** `void CbcCutGenerator::incrementNumberOfElementsInTotal ( int value = 1 )` `[inline]`

Definition at line 258 of file CbcCutGenerator.hpp.

**7.23.3.38** `int CbcCutGenerator::numberOfColumnCuts ( ) const` `[inline]`

Total number of column cuts.

Definition at line 262 of file CbcCutGenerator.hpp.

7.23.3.39 void CbcCutGenerator::setNumberColumnCuts ( int *value* ) [inline]

Definition at line 265 of file CbcCutGenerator.hpp.

7.23.3.40 void CbcCutGenerator::incrementNumberColumnCuts ( int *value* = 1 ) [inline]

Definition at line 268 of file CbcCutGenerator.hpp.

7.23.3.41 int CbcCutGenerator::numberCutsActive ( ) const [inline]

Total number of cuts active after (at end of n cut passes at each node)

Definition at line 272 of file CbcCutGenerator.hpp.

7.23.3.42 void CbcCutGenerator::setNumberCutsActive ( int *value* ) [inline]

Definition at line 275 of file CbcCutGenerator.hpp.

7.23.3.43 void CbcCutGenerator::incrementNumberCutsActive ( int *value* = 1 ) [inline]

Definition at line 278 of file CbcCutGenerator.hpp.

7.23.3.44 void CbcCutGenerator::setSwitchOffIfLessThan ( int *value* ) [inline]

Definition at line 281 of file CbcCutGenerator.hpp.

7.23.3.45 int CbcCutGenerator::switchOffIfLessThan ( ) const [inline]

Definition at line 284 of file CbcCutGenerator.hpp.

7.23.3.46 bool CbcCutGenerator::needsOptimalBasis ( ) const [inline]

Say if optimal basis needed.

Definition at line 288 of file CbcCutGenerator.hpp.

7.23.3.47 void CbcCutGenerator::setNeedsOptimalBasis ( bool *yesNo* ) [inline]

Set if optimal basis needed.

Definition at line 292 of file CbcCutGenerator.hpp.

7.23.3.48 bool CbcCutGenerator::mustCallAgain ( ) const [inline]

Whether generator MUST be called again if any cuts (i.e. ignore break from loop)

Definition at line 297 of file CbcCutGenerator.hpp.

7.23.3.49 void CbcCutGenerator::setMustCallAgain ( bool *yesNo* ) [inline]

Set whether generator MUST be called again if any cuts (i.e. ignore break from loop)

Definition at line 301 of file CbcCutGenerator.hpp.

7.23.3.50 bool CbcCutGenerator::switchedOff ( ) const [inline]

Whether generator switched off for moment.

Definition at line 306 of file CbcCutGenerator.hpp.

**7.23.3.51** `void CbcCutGenerator::setSwitchedOff ( bool yesNo ) [inline]`

Set whether generator switched off for moment.

Definition at line 310 of file CbcCutGenerator.hpp.

**7.23.3.52** `bool CbcCutGenerator::ineffectualCuts ( ) const [inline]`

Whether last round of cuts did little.

Definition at line 315 of file CbcCutGenerator.hpp.

**7.23.3.53** `void CbcCutGenerator::setIneffectualCuts ( bool yesNo ) [inline]`

Set whether last round of cuts did little.

Definition at line 319 of file CbcCutGenerator.hpp.

**7.23.3.54** `bool CbcCutGenerator::whetherToUse ( ) const [inline]`

Whether to use if any cuts generated.

Definition at line 324 of file CbcCutGenerator.hpp.

**7.23.3.55** `void CbcCutGenerator::setWhetherToUse ( bool yesNo ) [inline]`

Set whether to use if any cuts generated.

Definition at line 328 of file CbcCutGenerator.hpp.

**7.23.3.56** `bool CbcCutGenerator::whetherInMustCallAgainMode ( ) const [inline]`

Whether in must call again mode (or after others)

Definition at line 333 of file CbcCutGenerator.hpp.

**7.23.3.57** `void CbcCutGenerator::setWhetherInMustCallAgainMode ( bool yesNo ) [inline]`

Set whether in must call again mode (or after others)

Definition at line 337 of file CbcCutGenerator.hpp.

**7.23.3.58** `bool CbcCutGenerator::whetherCallAtEnd ( ) const [inline]`

Whether to call at end.

Definition at line 342 of file CbcCutGenerator.hpp.

**7.23.3.59** `void CbcCutGenerator::setWhetherCallAtEnd ( bool yesNo ) [inline]`

Set whether to call at end.

Definition at line 346 of file CbcCutGenerator.hpp.

**7.23.3.60** `int CbcCutGenerator::numberCutsAtRoot ( ) const [inline]`

Number of cuts generated at root.

Definition at line 351 of file CbcCutGenerator.hpp.

7.23.3.61 void CbcCutGenerator::setNumberCutsAtRoot ( int *value* ) [inline]

Definition at line 354 of file CbcCutGenerator.hpp.

7.23.3.62 int CbcCutGenerator::numberActiveCutsAtRoot ( ) const [inline]

Number of cuts active at root.

Definition at line 358 of file CbcCutGenerator.hpp.

7.23.3.63 void CbcCutGenerator::setNumberActiveCutsAtRoot ( int *value* ) [inline]

Definition at line 361 of file CbcCutGenerator.hpp.

7.23.3.64 int CbcCutGenerator::numberShortCutsAtRoot ( ) const [inline]

Number of short cuts at root.

Definition at line 365 of file CbcCutGenerator.hpp.

7.23.3.65 void CbcCutGenerator::setNumberShortCutsAtRoot ( int *value* ) [inline]

Definition at line 368 of file CbcCutGenerator.hpp.

7.23.3.66 void CbcCutGenerator::setModel ( CbcModel \* *model* ) [inline]

Set model.

Definition at line 372 of file CbcCutGenerator.hpp.

7.23.3.67 bool CbcCutGenerator::globalCutsAtRoot ( ) const [inline]

Whether global cuts at root.

Definition at line 376 of file CbcCutGenerator.hpp.

7.23.3.68 void CbcCutGenerator::setGlobalCutsAtRoot ( bool *yesNo* ) [inline]

Set whether global cuts at root.

Definition at line 380 of file CbcCutGenerator.hpp.

7.23.3.69 bool CbcCutGenerator::globalCuts ( ) const [inline]

Whether global cuts.

Definition at line 385 of file CbcCutGenerator.hpp.

7.23.3.70 void CbcCutGenerator::setGlobalCuts ( bool *yesNo* ) [inline]

Set whether global cuts.

Definition at line 389 of file CbcCutGenerator.hpp.

7.23.3.71 void CbcCutGenerator::addStatistics ( const CbcCutGenerator \* *other* )

Add in statistics from other.

7.23.3.72 void CbcCutGenerator::scaleBackStatistics ( int *factor* )

Scale back statistics by factor.

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

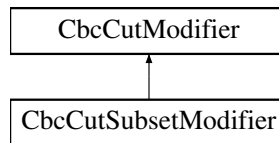
- [/home/ted/COIN/trunk/Cbc/src/CbcCutGenerator.hpp](#)

## 7.24 CbcCutModifier Class Reference

Abstract cut modifier base class.

```
#include <CbcCutModifier.hpp>
```

Inheritance diagram for CbcCutModifier:



### Public Member Functions

- [CbcCutModifier](#) ()  
*Default Constructor.*
- [CbcCutModifier](#) (const [CbcCutModifier](#) &)
- virtual [~CbcCutModifier](#) ()  
*Destructor.*
- [CbcCutModifier](#) & [operator=](#) (const [CbcCutModifier](#) &rhs)  
*Assignment.*
- virtual [CbcCutModifier](#) \* [clone](#) () const =0  
*Clone.*
- virtual int [modify](#) (const OsiSolverInterface \*solver, OsiRowCut &cut)=0  
*Returns 0 unchanged 1 strengthened 2 weakened 3 deleted.*
- virtual void [generateCpp](#) (FILE \*)  
*Create C++ lines to get to current state.*

### 7.24.1 Detailed Description

Abstract cut modifier base class.

In exotic circumstances - cuts may need to be modified a) strengthened - changed b) weakened - changed c) deleted - set to NULL d) unchanged

Definition at line 27 of file CbcCutModifier.hpp.

### 7.24.2 Constructor & Destructor Documentation

#### 7.24.2.1 CbcCutModifier::CbcCutModifier ( )

Default Constructor.

7.24.2.2 `CbcCutModifier::CbcCutModifier ( const CbcCutModifier & )`

7.24.2.3 `virtual CbcCutModifier::~~CbcCutModifier ( ) [virtual]`

Destructor.

### 7.24.3 Member Function Documentation

7.24.3.1 `CbcCutModifier& CbcCutModifier::operator= ( const CbcCutModifier & rhs )`

Assignment.

7.24.3.2 `virtual CbcCutModifier* CbcCutModifier::clone ( ) const [pure virtual]`

Clone.

Implemented in [CbcCutSubsetModifier](#).

7.24.3.3 `virtual int CbcCutModifier::modify ( const OsiSolverInterface * solver, OsiRowCut & cut ) [pure virtual]`

Returns 0 unchanged 1 strengthened 2 weakened 3 deleted.

Implemented in [CbcCutSubsetModifier](#).

7.24.3.4 `virtual void CbcCutModifier::generateCpp ( FILE * ) [inline],[virtual]`

Create C++ lines to get to current state.

Reimplemented in [CbcCutSubsetModifier](#).

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

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

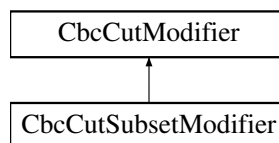
- [/home/ted/COIN/trunk/Cbc/src/CbcCutModifier.hpp](#)

## 7.25 CbcCutSubsetModifier Class Reference

Simple cut modifier base class.

```
#include <CbcCutSubsetModifier.hpp>
```

Inheritance diagram for `CbcCutSubsetModifier`:



### Public Member Functions

- [CbcCutSubsetModifier \( \)](#)  
*Default Constructor.*
- [CbcCutSubsetModifier \(int firstOdd\)](#)

*Useful Constructor.*

- `CbcCutSubsetModifier` (const `CbcCutSubsetModifier` &)
- virtual `~CbcCutSubsetModifier` ()

*Destructor.*

- `CbcCutSubsetModifier` & `operator=` (const `CbcCutSubsetModifier` &rhs)

*Assignment.*

- virtual `CbcCutModifier` \* `clone` () const

*Clone.*

- virtual int `modify` (const `OsiSolverInterface` \*solver, `OsiRowCut` &cut)

*Returns 0 unchanged 1 strengthened 2 weakened 3 deleted.*

- virtual void `generateCpp` (FILE \*)

*Create C++ lines to get to current state.*

## Protected Attributes

- int `firstOdd_`

*data First odd variable*

### 7.25.1 Detailed Description

Simple cut modifier base class.

In exotic circumstances - cuts may need to be modified a) strengthened - changed b) weakened - changed c) deleted - set to NULL d) unchanged

initially get rid of cuts with variables  $\geq k$  could weaken

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

### 7.25.2 Constructor & Destructor Documentation

#### 7.25.2.1 `CbcCutSubsetModifier::CbcCutSubsetModifier ( )`

Default Constructor.

#### 7.25.2.2 `CbcCutSubsetModifier::CbcCutSubsetModifier ( int firstOdd )`

Useful Constructor.

#### 7.25.2.3 `CbcCutSubsetModifier::CbcCutSubsetModifier ( const CbcCutSubsetModifier & )`

#### 7.25.2.4 `virtual CbcCutSubsetModifier::~~CbcCutSubsetModifier ( )` [virtual]

Destructor.

### 7.25.3 Member Function Documentation

#### 7.25.3.1 `CbcCutSubsetModifier& CbcCutSubsetModifier::operator= ( const CbcCutSubsetModifier & rhs )`

Assignment.



7.25.3.2 `virtual CbcCutModifier* CbcCutSubsetModifier::clone ( ) const` [virtual]

Clone.

Implements [CbcCutModifier](#).

7.25.3.3 `virtual int CbcCutSubsetModifier::modify ( const OsiSolverInterface * solver, OsiRowCut & cut )` [virtual]

Returns 0 unchanged 1 strengthened 2 weakened 3 deleted.

Implements [CbcCutModifier](#).

7.25.3.4 `virtual void CbcCutSubsetModifier::generateCpp ( FILE * )` [inline],[virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcCutModifier](#).

Definition at line 58 of file `CbcCutSubsetModifier.hpp`.

## 7.25.4 Member Data Documentation

7.25.4.1 `int CbcCutSubsetModifier::firstOdd_` [protected]

data First odd variable

Definition at line 62 of file `CbcCutSubsetModifier.hpp`.

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

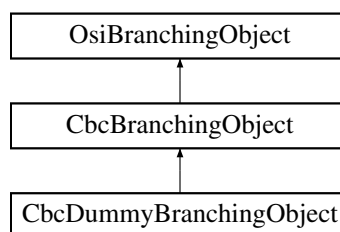
- `/home/ted/COIN/trunk/Cbc/src/CbcCutSubsetModifier.hpp`

## 7.26 CbcDummyBranchingObject Class Reference

Dummy branching object.

```
#include <CbcDummyBranchingObject.hpp>
```

Inheritance diagram for `CbcDummyBranchingObject`:



### Public Member Functions

- [CbcDummyBranchingObject](#) (`CbcModel *model=NULL`)  
*Default constructor.*
- [CbcDummyBranchingObject](#) (const [CbcDummyBranchingObject](#) &)  
*Copy constructor.*
- [CbcDummyBranchingObject](#) & `operator=` (const [CbcDummyBranchingObject](#) &rhs)

*Assignment operator.*

- virtual `CbcBranchingObject * clone () const`

*Clone.*

- virtual `~CbcDummyBranchingObject ()`

*Destructor.*

- virtual double `branch ()`

*Dummy branch.*

- virtual void `print ()`

*Print something about branch - only if log level high.*

- virtual `CbcBranchObjType type () const`

*Return the type (an integer identifier) of this.*

- virtual int `compareOriginalObject (const CbcBranchingObject *brObj) const`

*Compare the original object of this with the original object of brObj.*

- virtual `CbcRangeCompare compareBranchingObject (const CbcBranchingObject *brObj, const bool replaceIfOverlap=false)`

*Compare the this with brObj.*

## Additional Inherited Members

### 7.26.1 Detailed Description

Dummy branching object.

This object specifies a one-way dummy branch. This is so one can carry on branching even when it looks feasible

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

### 7.26.2 Constructor & Destructor Documentation

#### 7.26.2.1 `CbcDummyBranchingObject::CbcDummyBranchingObject ( CbcModel * model = NULL )`

Default constructor.

#### 7.26.2.2 `CbcDummyBranchingObject::CbcDummyBranchingObject ( const CbcDummyBranchingObject & )`

Copy constructor.

#### 7.26.2.3 `virtual CbcDummyBranchingObject::~~CbcDummyBranchingObject ( ) [virtual]`

Destructor.

### 7.26.3 Member Function Documentation

#### 7.26.3.1 `CbcDummyBranchingObject& CbcDummyBranchingObject::operator= ( const CbcDummyBranchingObject & rhs )`

Assignment operator.

#### 7.26.3.2 `virtual CbcBranchingObject* CbcDummyBranchingObject::clone ( ) const [virtual]`

Clone.

Implements `CbcBranchingObject`.

7.26.3.3 `virtual double CbcDummyBranchingObject::branch ( ) [virtual]`

Dummy branch.

Implements [CbcBranchingObject](#).

7.26.3.4 `virtual void CbcDummyBranchingObject::print ( ) [virtual]`

Print something about branch - only if log level high.

7.26.3.5 `virtual CbcBranchObjType CbcDummyBranchingObject::type ( ) const [inline],[virtual]`

Return the type (an integer identifier) of `this`.

Implements [CbcBranchingObject](#).

Definition at line 56 of file `CbcDummyBranchingObject.hpp`.

7.26.3.6 `virtual int CbcDummyBranchingObject::compareOriginalObject ( const CbcBranchingObject * brObj ) const [virtual]`

Compare the original object of `this` with the original object of `brObj`.

Assumes that there is an ordering of the original objects. This method should be invoked only if `this` and `brObj` are of the same type. Return negative/0/positive depending on whether `this` is smaller/same/larger than the argument.

Reimplemented from [CbcBranchingObject](#).

7.26.3.7 `virtual CbcRangeCompare CbcDummyBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false ) [virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

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

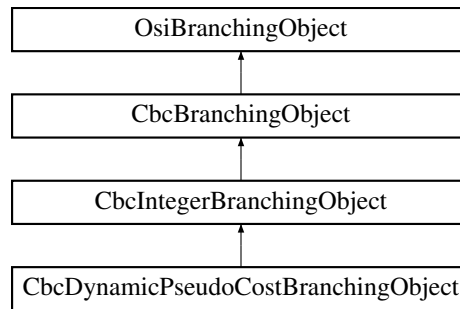
- `/home/ted/COIN/trunk/Cbc/src/CbcDummyBranchingObject.hpp`

## 7.27 CbcDynamicPseudoCostBranchingObject Class Reference

Simple branching object for an integer variable with pseudo costs.

```
#include <CbcBranchDynamic.hpp>
```

Inheritance diagram for `CbcDynamicPseudoCostBranchingObject`:



### Public Member Functions

- `CbcDynamicPseudoCostBranchingObject` ()  
*Default constructor.*
- `CbcDynamicPseudoCostBranchingObject` (`CbcModel` \*model, int variable, int way, double value, `CbcSimpleIntegerDynamicPseudoCost` \*object)  
*Create a standard floor/ceiling branch object.*
- `CbcDynamicPseudoCostBranchingObject` (`CbcModel` \*model, int variable, int way, double lowerValue, double upperValue)  
*Create a degenerate branch object.*
- `CbcDynamicPseudoCostBranchingObject` (const `CbcDynamicPseudoCostBranchingObject` &)  
*Copy constructor.*
- `CbcDynamicPseudoCostBranchingObject` & operator= (const `CbcDynamicPseudoCostBranchingObject` &rhs)  
*Assignment operator.*
- virtual `CbcBranchingObject` \* clone () const  
*Clone.*
- virtual ~`CbcDynamicPseudoCostBranchingObject` ()  
*Destructor.*
- void fillPart (int variable, int way, double value, `CbcSimpleIntegerDynamicPseudoCost` \*object)  
*Does part of constructor.*
- virtual double branch ()  
*Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.*
- virtual int fillStrongInfo (`CbcStrongInfo` &info)  
*Some branchingObjects may claim to be able to skip strong branching.*
- double changeInGuessed () const  
*Change in guessed.*
- void setChangeInGuessed (double value)  
*Set change in guessed.*
- `CbcSimpleIntegerDynamicPseudoCost` \* object () const  
*Return object.*
- void setObject (`CbcSimpleIntegerDynamicPseudoCost` \*object)  
*Set object.*
- virtual `CbcBranchObjType` type () const  
*Return the type (an integer identifier) of this.*

## Protected Attributes

- double [changeInGuessed\\_](#)  
*Change in guessed objective value for next branch.*
- [CbcSimpleIntegerDynamicPseudoCost](#) \* [object\\_](#)  
*Pointer back to object.*

## 7.27.1 Detailed Description

Simple branching object for an integer variable with pseudo costs.

This object can specify a two-way branch on an integer variable. For each arm of the branch, the upper and lower bounds on the variable can be independently specified.

Variable\_ holds the index of the integer variable in the integerVariable\_ array of the model.

Definition at line 111 of file CbcBranchDynamic.hpp.

## 7.27.2 Constructor &amp; Destructor Documentation

## 7.27.2.1 CbcDynamicPseudoCostBranchingObject::CbcDynamicPseudoCostBranchingObject ( )

Default constructor.

## 7.27.2.2 CbcDynamicPseudoCostBranchingObject::CbcDynamicPseudoCostBranchingObject ( CbcModel \* model, int variable, int way, double value, CbcSimpleIntegerDynamicPseudoCost \* object )

Create a standard floor/ceiling branch object.

Specifies a simple two-way branch. Let `value = x*`. One arm of the branch will be  $lb \leq x \leq \text{floor}(x^*)$ , the other  $\text{ceil}(x^*) \leq x \leq ub$ . Specify `way = -1` to set the object state to perform the down arm first, `way = 1` for the up arm.

## 7.27.2.3 CbcDynamicPseudoCostBranchingObject::CbcDynamicPseudoCostBranchingObject ( CbcModel \* model, int variable, int way, double lowerValue, double upperValue )

Create a degenerate branch object.

Specifies a 'one-way branch'. Calling [branch\(\)](#) for this object will always result in  $\text{lowerValue} \leq x \leq \text{upperValue}$ . Used to fix a variable when  $\text{lowerValue} = \text{upperValue}$ .

## 7.27.2.4 CbcDynamicPseudoCostBranchingObject::CbcDynamicPseudoCostBranchingObject ( const CbcDynamicPseudoCostBranchingObject &amp; )

Copy constructor.

## 7.27.2.5 virtual CbcDynamicPseudoCostBranchingObject::~CbcDynamicPseudoCostBranchingObject ( ) [virtual]

Destructor.

## 7.27.3 Member Function Documentation

## 7.27.3.1 CbcDynamicPseudoCostBranchingObject&amp; CbcDynamicPseudoCostBranchingObject::operator= ( const CbcDynamicPseudoCostBranchingObject &amp; rhs )

Assignment operator.

**7.27.3.2** `virtual CbcBranchingObject* CbcDynamicPseudoCostBranchingObject::clone ( ) const` [virtual]

Clone.

Reimplemented from [CbcIntegerBranchingObject](#).

**7.27.3.3** `void CbcDynamicPseudoCostBranchingObject::fillPart ( int variable, int way, double value,  
CbcSimpleIntegerDynamicPseudoCost * object )`

Does part of constructor.

**7.27.3.4** `virtual double CbcDynamicPseudoCostBranchingObject::branch ( )` [virtual]

Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.

This version also changes guessed objective value

Reimplemented from [CbcIntegerBranchingObject](#).

**7.27.3.5** `virtual int CbcDynamicPseudoCostBranchingObject::fillStrongInfo ( CbcStrongInfo & info )` [virtual]

Some branchingObjects may claim to be able to skip strong branching.

If so they have to fill in [CbcStrongInfo](#). The object mention in incoming [CbcStrongInfo](#) must match. Returns nonzero if skip is wanted

Reimplemented from [CbcBranchingObject](#).

**7.27.3.6** `double CbcDynamicPseudoCostBranchingObject::changeInGuessed ( ) const` [inline]

Change in guessed.

Definition at line 170 of file CbcBranchDynamic.hpp.

**7.27.3.7** `void CbcDynamicPseudoCostBranchingObject::setChangeInGuessed ( double value )` [inline]

Set change in guessed.

Definition at line 174 of file CbcBranchDynamic.hpp.

**7.27.3.8** `CbcSimpleIntegerDynamicPseudoCost* CbcDynamicPseudoCostBranchingObject::object ( ) const`  
[inline]

Return object.

Definition at line 178 of file CbcBranchDynamic.hpp.

**7.27.3.9** `void CbcDynamicPseudoCostBranchingObject::setObject ( CbcSimpleIntegerDynamicPseudoCost * object )`  
[inline]

Set object.

Definition at line 182 of file CbcBranchDynamic.hpp.

**7.27.3.10** `virtual CbcBranchObjType CbcDynamicPseudoCostBranchingObject::type ( ) const` [inline],[virtual]

Return the type (an integer identifier) of `this`.

Reimplemented from [CbcIntegerBranchingObject](#).

Definition at line 187 of file CbcBranchDynamic.hpp.

## 7.27.4 Member Data Documentation

7.27.4.1 `double CbcDynamicPseudoCostBranchingObject::changeInGuessed_` [protected]

Change in guessed objective value for next branch.

Definition at line 199 of file CbcBranchDynamic.hpp.

7.27.4.2 `CbcSimpleIntegerDynamicPseudoCost* CbcDynamicPseudoCostBranchingObject::object_` [protected]

Pointer back to object.

Definition at line 201 of file CbcBranchDynamic.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcBranchDynamic.hpp](#)

## 7.28 CbcEventHandler Class Reference

Base class for Cbc event handling.

```
#include <CbcEventHandler.hpp>
```

## Public Types

- enum [CbcEvent](#) {  
[node](#) = 200, [treeStatus](#), [solution](#), [heuristicSolution](#),  
[beforeSolution1](#), [beforeSolution2](#), [afterHeuristic](#), [smallBranchAndBound](#),  
[heuristicPass](#), [endSearch](#) }  
*Events known to cbc.*
- enum [CbcAction](#) {  
[noAction](#) = -1, [stop](#) = 0, [restart](#), [restartRoot](#),  
[addCuts](#), [killSolution](#), [takeAction](#) }  
*Action codes returned by the event handler.*
- typedef std::map< [CbcEvent](#),  
[CbcAction](#) > [eaMapPair](#)  
*Data type for event/action pairs.*

## Public Member Functions

## Event Processing

- virtual [CbcAction event](#) ([CbcEvent](#) whichEvent)  
*Return the action to be taken for an event.*
- virtual [CbcAction event](#) ([CbcEvent](#) whichEvent, void \*data)  
*Return the action to be taken for an event - and modify data.*

## Constructors and destructors

- [CbcEventHandler](#) ([CbcModel](#) \*model=0)  
*Default constructor.*
- [CbcEventHandler](#) (const [CbcEventHandler](#) &orig)

- *Copy constructor.*
- `CbcEventHandler & operator= (const CbcEventHandler &rhs)`  
*Assignment.*
- `virtual CbcEventHandler * clone () const`  
*Clone (virtual) constructor.*
- `virtual ~CbcEventHandler ()`  
*Destructor.*

#### Set/Get methods

- `void setModel (CbcModel *model)`  
*Set model.*
- `const CbcModel * getModel () const`  
*Get model.*
- `void setDfltAction (CbcAction action)`  
*Set the default action.*
- `void setAction (CbcEvent event, CbcAction action)`  
*Set the action code associated with an event.*

#### Protected Attributes

##### Data members

*Protected (as opposed to private) to allow access by derived classes.*

- `CbcModel * model_`  
*Pointer to associated CbcModel.*
- `CbcAction dfltAction_`  
*Default action.*
- `eaMapPair * eaMap_`  
*Pointer to a map that holds non-default event/action pairs.*

#### 7.28.1 Detailed Description

Base class for Cbc event handling.

Up front: We're not talking about unanticipated events here. We're talking about anticipated events, in the sense that the code is going to make a call to `event()` and is prepared to obey the return value that it receives.

The general pattern for usage is as follows:

1. Create a `CbcEventHandler` object. This will be initialised with a set of default actions for every recognised event.
2. Attach the event handler to the `CbcModel` object.
3. When execution reaches the point where an event occurs, call the event handler as `CbcEventHandler::event(the event)`. The return value will specify what the code should do in response to the event.

The return value associated with an event can be changed at any time.

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

#### 7.28.2 Member Typedef Documentation

##### 7.28.2.1 `typedef std::map<CbcEvent, CbcAction> CbcEventHandler::eaMapPair`

Data type for event/action pairs.

Definition at line 135 of file `CbcEventHandler.hpp`.



## 7.28.3 Member Enumeration Documentation

## 7.28.3.1 enum CbcEventHandler::CbcEvent

Events known to cbc.

## Enumerator

- node** Processing of the current node is complete.
- treeStatus** A tree status interval has arrived.
- solution** A solution has been found.
- heuristicSolution** A heuristic solution has been found.
- beforeSolution1** A solution will be found unless user takes action (first check).
- beforeSolution2** A solution will be found unless user takes action (thorough check).
- afterHeuristic** After failed heuristic.
- smallBranchAndBound** On entry to small branch and bound.
- heuristicPass** After a pass of heuristic.
- endSearch** End of search.

Definition at line 87 of file CbcEventHandler.hpp.

## 7.28.3.2 enum CbcEventHandler::CbcAction

Action codes returned by the event handler.

Specific values are chosen to match ClpEventHandler return codes.

## Enumerator

- noAction** Continue — no action required.
- stop** Stop — abort the current run at the next opportunity.
- restart** Restart — restart branch-and-cut search; do not undo root node processing.
- restartRoot** RestartRoot — undo root node and start branch-and-cut afresh.
- addCuts** Add special cuts.
- killSolution** Pretend solution never happened.
- takeAction** Take action on modified data.

Definition at line 114 of file CbcEventHandler.hpp.

## 7.28.4 Constructor &amp; Destructor Documentation

## 7.28.4.1 CbcEventHandler::CbcEventHandler ( CbcModel \* model = 0 )

Default constructor.

## 7.28.4.2 CbcEventHandler::CbcEventHandler ( const CbcEventHandler &amp; orig )

Copy constructor.

## 7.28.4.3 virtual CbcEventHandler::~CbcEventHandler ( ) [virtual]

Destructor.

### 7.28.5 Member Function Documentation

#### 7.28.5.1 `virtual CbcAction CbcEventHandler::event ( CbcEvent whichEvent )` [virtual]

Return the action to be taken for an event.

Return the action that should be taken in response to the event passed as the parameter. The default implementation simply reads a return code from a map.

#### 7.28.5.2 `virtual CbcAction CbcEventHandler::event ( CbcEvent whichEvent, void * data )` [virtual]

Return the action to be taken for an event - and modify data.

Return the action that should be taken in response to the event passed as the parameter. The default implementation simply reads a return code from a map.

#### 7.28.5.3 `CbcEventHandler& CbcEventHandler::operator= ( const CbcEventHandler & rhs )`

Assignment.

#### 7.28.5.4 `virtual CbcEventHandler* CbcEventHandler::clone ( ) const` [virtual]

Clone (virtual) constructor.

#### 7.28.5.5 `void CbcEventHandler::setModel ( CbcModel * model )` [inline]

Set model.

Definition at line 190 of file CbcEventHandler.hpp.

#### 7.28.5.6 `const CbcModel* CbcEventHandler::getModel ( ) const` [inline]

Get model.

Definition at line 196 of file CbcEventHandler.hpp.

#### 7.28.5.7 `void CbcEventHandler::setDefaultAction ( CbcAction action )` [inline]

Set the default action.

Definition at line 202 of file CbcEventHandler.hpp.

#### 7.28.5.8 `void CbcEventHandler::setAction ( CbcEvent event, CbcAction action )` [inline]

Set the action code associated with an event.

Definition at line 208 of file CbcEventHandler.hpp.

### 7.28.6 Member Data Documentation

#### 7.28.6.1 `CbcModel* CbcEventHandler::model_` [protected]

Pointer to associated [CbcModel](#).

Definition at line 228 of file CbcEventHandler.hpp.

#### 7.28.6.2 `CbcAction CbcEventHandler::defaultAction_` [protected]

Default action.

Definition at line 232 of file CbcEventHandler.hpp.

### 7.28.6.3 eaMapPair\* CbcEventHandler::eaMap\_ [protected]

Pointer to a map that holds non-default event/action pairs.

Definition at line 236 of file CbcEventHandler.hpp.

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

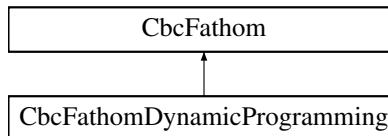
- /home/ted/COIN/trunk/Cbc/src/CbcEventHandler.hpp

## 7.29 CbcFathom Class Reference

Fathom base class.

```
#include <CbcFathom.hpp>
```

Inheritance diagram for CbcFathom:



### Public Member Functions

- [CbcFathom](#) ()
- [CbcFathom](#) (CbcModel &model)
- virtual [~CbcFathom](#) ()
- virtual void [setModel](#) (CbcModel \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual [CbcFathom](#) \* [clone](#) () const =0  
*Clone.*
- virtual void [resetModel](#) (CbcModel \*model)=0  
*Resets stuff if model changes.*
- virtual int [fathom](#) (double \*&newSolution)=0  
*returns 0 if no fathoming attempted, 1 fully fathomed, 2 incomplete search, 3 incomplete search but treat as complete.*
- bool [possible](#) () const

### Protected Attributes

- [CbcModel](#) \* [model\\_](#)  
*Model.*
- bool [possible\\_](#)  
*Possible - if this method of fathoming can be used.*

### 7.29.1 Detailed Description

Fathom base class.

The idea is that after some branching the problem will be effectively smaller than the original problem and maybe there will be a more specialized technique which can completely fathom this branch quickly.

One method is to presolve the problem to give a much smaller new problem and then do branch and cut on that. Another might be dynamic programming.

Definition at line 32 of file CbcFathom.hpp.

### 7.29.2 Constructor & Destructor Documentation

#### 7.29.2.1 CbcFathom::CbcFathom ( )

#### 7.29.2.2 CbcFathom::CbcFathom ( CbcModel & model )

#### 7.29.2.3 virtual CbcFathom::~~CbcFathom ( ) [virtual]

### 7.29.3 Member Function Documentation

#### 7.29.3.1 virtual void CbcFathom::setModel ( CbcModel \* model ) [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented in [CbcFathomDynamicProgramming](#).

#### 7.29.3.2 virtual CbcFathom\* CbcFathom::clone ( ) const [pure virtual]

Clone.

Implemented in [CbcFathomDynamicProgramming](#).

#### 7.29.3.3 virtual void CbcFathom::resetModel ( CbcModel \* model ) [pure virtual]

Resets stuff if model changes.

Implemented in [CbcFathomDynamicProgramming](#).

#### 7.29.3.4 virtual int CbcFathom::fathom ( double \*& newSolution ) [pure virtual]

returns 0 if no fathoming attempted, 1 fully fathomed, 2 incomplete search, 3 incomplete search but treat as complete.

If solution then newSolution will not be NULL and will be freed by [CbcModel](#). It is expected that the solution is better than best so far but [CbcModel](#) will double check.

If returns 3 then of course there is no guarantee of global optimum

Implemented in [CbcFathomDynamicProgramming](#).

#### 7.29.3.5 bool CbcFathom::possible ( ) const [inline]

Definition at line 62 of file CbcFathom.hpp.

### 7.29.4 Member Data Documentation

## 7.29.4.1 CbcModel\* CbcFathom::model\_ [protected]

Model.

Definition at line 69 of file CbcFathom.hpp.

## 7.29.4.2 bool CbcFathom::possible\_ [protected]

Possible - if this method of fathoming can be used.

Definition at line 71 of file CbcFathom.hpp.

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

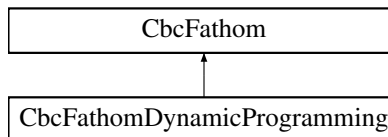
- </home/ted/COIN/trunk/Cbc/src/CbcFathom.hpp>

## 7.30 CbcFathomDynamicProgramming Class Reference

FathomDynamicProgramming class.

```
#include <CbcFathomDynamicProgramming.hpp>
```

Inheritance diagram for CbcFathomDynamicProgramming:



## Public Member Functions

- [CbcFathomDynamicProgramming](#) ()
- [CbcFathomDynamicProgramming](#) (CbcModel &model)
- [CbcFathomDynamicProgramming](#) (const [CbcFathomDynamicProgramming](#) &rhs)
- virtual [~CbcFathomDynamicProgramming](#) ()
- virtual void [setModel](#) (CbcModel \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual [CbcFathom](#) \* [clone](#) () const  
*Clone.*
- virtual void [resetModel](#) (CbcModel \*model)  
*Resets stuff if model changes.*
- virtual int [fathom](#) (double \*&newSolution)  
*returns 0 if no fathoming attempted, 1 fully fathomed, 2 incomplete search, 3 incomplete search but treat as complete.*
- int [maximumSize](#) () const  
*Maximum size allowed.*
- void [setMaximumSize](#) (int value)
- int [checkPossible](#) (int allowableSize=0)  
*Returns type of algorithm and sets up arrays.*
- void [setAlgorithm](#) (int value)
- bool [tryColumn](#) (int numberElements, const int \*rows, const double \*coefficients, double [cost](#), int upper=COIN\_INT\_MAX)

*Tries a column returns true if was used in making any changes.*

- const double \* [cost](#) () const

*Returns cost array.*

- const int \* [back](#) () const

*Returns back array.*

- int [target](#) () const

*Gets bit pattern for target result.*

- void [setTarget](#) (int value)

*Sets bit pattern for target result.*

## Protected Attributes

- int [size\\_](#)

*Size of states (power of 2 unless just one constraint)*

- int [type\\_](#)

*Type - 0 coefficients and rhs all 1, 1 - coefficients > 1 or rhs > 1.*

- double \* [cost\\_](#)

*Space for states.*

- int \* [back\\_](#)

*Which state produced this cheapest one.*

- int \* [lookup\\_](#)

*Some rows may be satisfied so we need a lookup.*

- int \* [indices\\_](#)

*Space for sorted indices.*

- int [numberActive\\_](#)

*Number of active rows.*

- int [maximumSizeAllowed\\_](#)

*Maximum size allowed.*

- int \* [startBit\\_](#)

*Start bit for each active row.*

- int \* [numberBits\\_](#)

*Number bits for each active row.*

- int \* [rhs\\_](#)

*Effective rhs.*

- int \* [coefficients\\_](#)

*Space for sorted coefficients.*

- int [target\\_](#)

*Target pattern.*

- int [numberNonOne\\_](#)

*Number of Non 1 rhs.*

- int [bitPattern\\_](#)

*Current bit pattern.*

- int [algorithm\\_](#)

*Current algorithm.*

## 7.30.1 Detailed Description

FathomDynamicProgramming class.

The idea is that after some branching the problem will be effectively smaller than the original problem and maybe there will be a more specialized technique which can completely fathom this branch quickly.

This is a dynamic programming implementation which is very fast for some specialized problems. It expects small integral rhs, an all integer problem and positive integral coefficients. At present it can not do general set covering problems just set partitioning. It can find multiple optima for various rhs combinations.

The main limiting factor is size of state space. Each 1 rhs doubles the size of the problem. 2 or 3 rhs quadruples, 4,5,6,7 by 8 etc.

Definition at line 28 of file CbcFathomDynamicProgramming.hpp.

## 7.30.2 Constructor &amp; Destructor Documentation

7.30.2.1 CbcFathomDynamicProgramming::CbcFathomDynamicProgramming ( )

7.30.2.2 CbcFathomDynamicProgramming::CbcFathomDynamicProgramming ( CbcModel & model )

7.30.2.3 CbcFathomDynamicProgramming::CbcFathomDynamicProgramming ( const CbcFathomDynamicProgramming & rhs )

7.30.2.4 virtual CbcFathomDynamicProgramming::~CbcFathomDynamicProgramming ( ) [virtual]

## 7.30.3 Member Function Documentation

7.30.3.1 virtual void CbcFathomDynamicProgramming::setModel ( CbcModel \* model ) [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcFathom](#).

7.30.3.2 virtual CbcFathom\* CbcFathomDynamicProgramming::clone ( ) const [virtual]

Clone.

Implements [CbcFathom](#).

7.30.3.3 virtual void CbcFathomDynamicProgramming::resetModel ( CbcModel \* model ) [virtual]

Resets stuff if model changes.

Implements [CbcFathom](#).

7.30.3.4 virtual int CbcFathomDynamicProgramming::fathom ( double \*& newSolution ) [virtual]

returns 0 if no fathoming attempted, 1 fully fathomed , 2 incomplete search, 3 incomplete search but treat as complete.

If solution then newSolution will not be NULL and will be freed by [CbcModel](#). It is expected that the solution is better than best so far but [CbcModel](#) will double check.

If returns 3 then of course there is no guarantee of global optimum

Implements [CbcFathom](#).

7.30.3.5 `int CbcFathomDynamicProgramming::maximumSize ( ) const` `[inline]`

Maximum size allowed.

Definition at line 60 of file CbcFathomDynamicProgramming.hpp.

7.30.3.6 `void CbcFathomDynamicProgramming::setMaximumSize ( int value )` `[inline]`

Definition at line 63 of file CbcFathomDynamicProgramming.hpp.

7.30.3.7 `int CbcFathomDynamicProgramming::checkPossible ( int allowableSize = 0 )`

Returns type of algorithm and sets up arrays.

7.30.3.8 `void CbcFathomDynamicProgramming::setAlgorithm ( int value )` `[inline]`

Definition at line 69 of file CbcFathomDynamicProgramming.hpp.

7.30.3.9 `bool CbcFathomDynamicProgramming::tryColumn ( int numberElements, const int * rows, const double * coefficients, double cost, int upper = COIN_INT_MAX )`

Tries a column returns true if was used in making any changes.

7.30.3.10 `const double* CbcFathomDynamicProgramming::cost ( ) const` `[inline]`

Returns cost array.

Definition at line 79 of file CbcFathomDynamicProgramming.hpp.

7.30.3.11 `const int* CbcFathomDynamicProgramming::back ( ) const` `[inline]`

Returns back array.

Definition at line 83 of file CbcFathomDynamicProgramming.hpp.

7.30.3.12 `int CbcFathomDynamicProgramming::target ( ) const` `[inline]`

Gets bit pattern for target result.

Definition at line 87 of file CbcFathomDynamicProgramming.hpp.

7.30.3.13 `void CbcFathomDynamicProgramming::setTarget ( int value )` `[inline]`

Sets bit pattern for target result.

Definition at line 91 of file CbcFathomDynamicProgramming.hpp.

#### 7.30.4 Member Data Documentation

7.30.4.1 `int CbcFathomDynamicProgramming::size_` `[protected]`

Size of states (power of 2 unless just one constraint)

Definition at line 128 of file CbcFathomDynamicProgramming.hpp.

7.30.4.2 `int CbcFathomDynamicProgramming::type_` `[protected]`

Type - 0 coefficients and rhs all 1, 1 - coefficients > 1 or rhs > 1.

Definition at line 132 of file CbcFathomDynamicProgramming.hpp.



**7.30.4.3** `double* CbcFathomDynamicProgramming::cost_` `[protected]`

Space for states.

Definition at line 134 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.4** `int* CbcFathomDynamicProgramming::back_` `[protected]`

Which state produced this cheapest one.

Definition at line 136 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.5** `int* CbcFathomDynamicProgramming::lookup_` `[protected]`

Some rows may be satisfied so we need a lookup.

Definition at line 138 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.6** `int* CbcFathomDynamicProgramming::indices_` `[protected]`

Space for sorted indices.

Definition at line 140 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.7** `int CbcFathomDynamicProgramming::numberActive_` `[protected]`

Number of active rows.

Definition at line 142 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.8** `int CbcFathomDynamicProgramming::maximumSizeAllowed_` `[protected]`

Maximum size allowed.

Definition at line 144 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.9** `int* CbcFathomDynamicProgramming::startBit_` `[protected]`

Start bit for each active row.

Definition at line 146 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.10** `int* CbcFathomDynamicProgramming::numberBits_` `[protected]`

Number bits for each active row.

Definition at line 148 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.11** `int* CbcFathomDynamicProgramming::rhs_` `[protected]`

Effective rhs.

Definition at line 150 of file CbcFathomDynamicProgramming.hpp.

**7.30.4.12** `int* CbcFathomDynamicProgramming::coefficients_` `[protected]`

Space for sorted coefficients.

Definition at line 152 of file CbcFathomDynamicProgramming.hpp.

#### 7.30.4.13 `int CbcFathomDynamicProgramming::target_` [protected]

Target pattern.

Definition at line 154 of file `CbcFathomDynamicProgramming.hpp`.

#### 7.30.4.14 `int CbcFathomDynamicProgramming::numberNonOne_` [protected]

Number of Non 1 rhs.

Definition at line 156 of file `CbcFathomDynamicProgramming.hpp`.

#### 7.30.4.15 `int CbcFathomDynamicProgramming::bitPattern_` [protected]

Current bit pattern.

Definition at line 158 of file `CbcFathomDynamicProgramming.hpp`.

#### 7.30.4.16 `int CbcFathomDynamicProgramming::algorithm_` [protected]

Current algorithm.

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

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

- [/home/ted/COIN/trunk/Cbc/src/CbcFathomDynamicProgramming.hpp](#)

## 7.31 CbcFeasibilityBase Class Reference

```
#include <CbcFeasibilityBase.hpp>
```

### Public Member Functions

- [CbcFeasibilityBase](#) ()
- virtual int [feasible](#) ([CbcModel](#) \*, int)  
*On input mode: 0 - called after a solve but before any cuts -1 - called after strong branching Returns : 0 - no opinion -1 pretend infeasible 1 pretend integer solution.*
- virtual [~CbcFeasibilityBase](#) ()
- [CbcFeasibilityBase](#) (const [CbcFeasibilityBase](#) &)
- [CbcFeasibilityBase](#) & [operator=](#) (const [CbcFeasibilityBase](#) &)
- virtual [CbcFeasibilityBase](#) \* [clone](#) () const  
*Clone.*

#### 7.31.1 Detailed Description

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

#### 7.31.2 Constructor & Destructor Documentation

##### 7.31.2.1 `CbcFeasibilityBase::CbcFeasibilityBase ( )` [inline]

Definition at line 25 of file `CbcFeasibilityBase.hpp`.

7.31.2.2 `virtual CbcFeasibilityBase::~~CbcFeasibilityBase ( ) [inline],[virtual]`

Definition at line 40 of file CbcFeasibilityBase.hpp.

7.31.2.3 `CbcFeasibilityBase::CbcFeasibilityBase ( const CbcFeasibilityBase & ) [inline]`

Definition at line 43 of file CbcFeasibilityBase.hpp.

### 7.31.3 Member Function Documentation

7.31.3.1 `virtual int CbcFeasibilityBase::feasible ( CbcModel *, int ) [inline],[virtual]`

On input mode: 0 - called after a solve but before any cuts -1 - called after strong branching Returns : 0 - no opinion -1 pretend infeasible 1 pretend integer solution.

Definition at line 36 of file CbcFeasibilityBase.hpp.

7.31.3.2 `CbcFeasibilityBase& CbcFeasibilityBase::operator= ( const CbcFeasibilityBase & ) [inline]`

Definition at line 46 of file CbcFeasibilityBase.hpp.

7.31.3.3 `virtual CbcFeasibilityBase* CbcFeasibilityBase::clone ( ) const [inline],[virtual]`

Clone.

Definition at line 51 of file CbcFeasibilityBase.hpp.

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

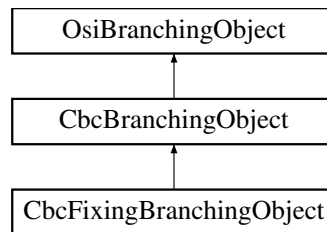
- [/home/ted/COIN/trunk/Cbc/src/CbcFeasibilityBase.hpp](#)

## 7.32 CbcFixingBranchingObject Class Reference

General Branching Object class.

```
#include <CbcFollowOn.hpp>
```

Inheritance diagram for CbcFixingBranchingObject:



### Public Member Functions

- [CbcFixingBranchingObject \( \)](#)
- [CbcFixingBranchingObject \(CbcModel \\*model, int way, int numberOnDownSide, const int \\*down, int numberOnUpSide, const int \\*up\)](#)
- [CbcFixingBranchingObject \(const CbcFixingBranchingObject &\)](#)
- [CbcFixingBranchingObject & operator= \(const CbcFixingBranchingObject &rhs\)](#)

- virtual [CbcBranchingObject](#) \* [clone](#) () const  
*Clone.*
- virtual [~CbcFixingBranchingObject](#) ()
- virtual double [branch](#) ()  
*Does next branch and updates state.*
- virtual void [print](#) ()  
*Print something about branch - only if log level high.*
- virtual [CbcBranchObjType](#) [type](#) () const  
*Return the type (an integer identifier) of this.*
- virtual int [compareOriginalObject](#) (const [CbcBranchingObject](#) \*brObj) const  
*Compare the original object of this with the original object of brObj.*
- virtual [CbcRangeCompare](#) [compareBranchingObject](#) (const [CbcBranchingObject](#) \*brObj, const bool [replacelf-Overlap](#)=false)  
*Compare the this with brObj.*

### Additional Inherited Members

#### 7.32.1 Detailed Description

General Branching Object class.

Each way fixes some variables to lower bound

Definition at line 74 of file CbcFollowOn.hpp.

#### 7.32.2 Constructor & Destructor Documentation

7.32.2.1 [CbcFixingBranchingObject::CbcFixingBranchingObject](#) ( )

7.32.2.2 [CbcFixingBranchingObject::CbcFixingBranchingObject](#) ( [CbcModel](#) \* *model*, int *way*, int *numberOnDownSide*, const int \* *down*, int *numberOnUpSide*, const int \* *up* )

7.32.2.3 [CbcFixingBranchingObject::CbcFixingBranchingObject](#) ( const [CbcFixingBranchingObject](#) & )

7.32.2.4 virtual [CbcFixingBranchingObject::~CbcFixingBranchingObject](#) ( ) [virtual]

#### 7.32.3 Member Function Documentation

7.32.3.1 [CbcFixingBranchingObject&](#) [CbcFixingBranchingObject::operator=](#) ( const [CbcFixingBranchingObject](#) & *rhs* )

7.32.3.2 virtual [CbcBranchingObject\\*](#) [CbcFixingBranchingObject::clone](#) ( ) const [virtual]

Clone.

Implements [CbcBranchingObject](#).

7.32.3.3 virtual double [CbcFixingBranchingObject::branch](#) ( ) [virtual]

Does next branch and updates state.

Implements [CbcBranchingObject](#).

7.32.3.4 virtual void [CbcFixingBranchingObject::print](#) ( ) [virtual]

Print something about branch - only if log level high.

7.32.3.5 `virtual CbcBranchObjType CbcFixingBranchingObject::type ( ) const` `[inline],[virtual]`

Return the type (an integer identifier) of `this`.

Implements [CbcBranchingObject](#).

Definition at line 117 of file `CbcFollowOn.hpp`.

7.32.3.6 `virtual int CbcFixingBranchingObject::compareOriginalObject ( const CbcBranchingObject * brObj ) const` `[virtual]`

Compare the original object of `this` with the original object of `brObj`.

Assumes that there is an ordering of the original objects. This method should be invoked only if `this` and `brObj` are of the same type. Return negative/0/positive depending on whether `this` is smaller/same/larger than the argument.

Reimplemented from [CbcBranchingObject](#).

7.32.3.7 `virtual CbcRangeCompare CbcFixingBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false )` `[virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

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

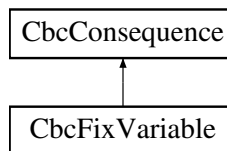
- `/home/ted/COIN/trunk/Cbc/src/CbcFollowOn.hpp`

## 7.33 CbcFixVariable Class Reference

Class for consequent bounds.

```
#include <CbcFixVariable.hpp>
```

Inheritance diagram for `CbcFixVariable`:



### Public Member Functions

- [CbcFixVariable](#) ( )
- [CbcFixVariable](#) (int numberStates, const int \*states, const int \*numberNewLower, const int \*\*newLowerValue, const int \*\*lowerColumn, const int \*numberNewUpper, const int \*\*newUpperValue, const int \*\*upperColumn)
- [CbcFixVariable](#) (const [CbcFixVariable](#) &rhs)
- [CbcFixVariable](#) & `operator=` (const [CbcFixVariable](#) &rhs)
- virtual [CbcConsequence](#) \* `clone` ( ) const

*Clone.*

- virtual `~CbcFixVariable ( )`  
*Destructor.*
- virtual void `applyToSolver (OsiSolverInterface *solver, int state) const`  
*Apply to an LP solver.*

#### Protected Attributes

- int `numberStates_`  
*Number of states.*
- int \* `states_`  
*Values of integers for various states.*
- int \* `startLower_`  
*Start of information for each state (setting new lower)*
- int \* `startUpper_`  
*Start of information for each state (setting new upper)*
- double \* `newBound_`  
*For each variable new bounds.*
- int \* `variable_`  
*Variable.*

#### 7.33.1 Detailed Description

Class for consequent bounds.

When a variable is branched on it normally interacts with other variables by means of equations. There are cases where we want to step outside LP and do something more directly e.g. fix bounds. This class is for that.

A state of -9999 means at LB, +9999 means at UB, others mean if fixed to that value.

Definition at line 22 of file CbcFixVariable.hpp.

#### 7.33.2 Constructor & Destructor Documentation

##### 7.33.2.1 CbcFixVariable::CbcFixVariable ( )

##### 7.33.2.2 CbcFixVariable::CbcFixVariable ( int *numberStates*, const int \* *states*, const int \* *numberNewLower*, const int \*\* *newLowerValue*, const int \*\* *lowerColumn*, const int \* *numberNewUpper*, const int \*\* *newUpperValue*, const int \*\* *upperColumn* )

##### 7.33.2.3 CbcFixVariable::CbcFixVariable ( const CbcFixVariable & *rhs* )

##### 7.33.2.4 virtual CbcFixVariable::~~CbcFixVariable ( ) [virtual]

Destructor.

#### 7.33.3 Member Function Documentation

##### 7.33.3.1 CbcFixVariable& CbcFixVariable::operator= ( const CbcFixVariable & *rhs* )

##### 7.33.3.2 virtual CbcConsequence\* CbcFixVariable::clone ( ) const [virtual]

Clone.

Implements [CbcConsequence](#).

**7.33.3.3** `virtual void CbcFixVariable::applyToSolver ( OsiSolverInterface * solver, int state ) const` [virtual]

Apply to an LP solver.

Action depends on state

Implements [CbcConsequence](#).

#### 7.33.4 Member Data Documentation

**7.33.4.1** `int CbcFixVariable::numberStates_` [protected]

Number of states.

Definition at line 53 of file CbcFixVariable.hpp.

**7.33.4.2** `int* CbcFixVariable::states_` [protected]

Values of integers for various states.

Definition at line 55 of file CbcFixVariable.hpp.

**7.33.4.3** `int* CbcFixVariable::startLower_` [protected]

Start of information for each state (setting new lower)

Definition at line 57 of file CbcFixVariable.hpp.

**7.33.4.4** `int* CbcFixVariable::startUpper_` [protected]

Start of information for each state (setting new upper)

Definition at line 59 of file CbcFixVariable.hpp.

**7.33.4.5** `double* CbcFixVariable::newBound_` [protected]

For each variable new bounds.

Definition at line 61 of file CbcFixVariable.hpp.

**7.33.4.6** `int* CbcFixVariable::variable_` [protected]

Variable.

Definition at line 63 of file CbcFixVariable.hpp.

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

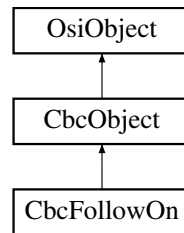
- [/home/ted/COIN/trunk/Cbc/src/CbcFixVariable.hpp](#)

## 7.34 CbcFollowOn Class Reference

Define a follow on class.

```
#include <CbcFollowOn.hpp>
```

Inheritance diagram for CbcFollowOn:



### Public Member Functions

- [CbcFollowOn](#) ()
- [CbcFollowOn](#) ([CbcModel](#) \*[model](#))  
*Useful constructor.*
- [CbcFollowOn](#) (const [CbcFollowOn](#) &)
- virtual [CbcObject](#) \* [clone](#) () const  
*Clone.*
- [CbcFollowOn](#) & [operator=](#) (const [CbcFollowOn](#) &[rhs](#))
- [~CbcFollowOn](#) ()
- virtual double [infeasibility](#) (const [OsiBranchingInformation](#) \*[info](#), int &[preferredWay](#)) const  
*Infeasibility - large is 0.5.*
- virtual void [feasibleRegion](#) ()  
*This looks at solution and sets bounds to contain solution.*
- virtual [CbcBranchingObject](#) \* [createCbcBranch](#) ([OsiSolverInterface](#) \*[solver](#), const [OsiBranchingInformation](#) \*[info](#), int [way](#))  
*Creates a branching object.*
- virtual int [gutsOffFollowOn](#) (int &[otherRow](#), int &[preferredWay](#)) const  
*As some computation is needed in more than one place - returns row.*

### Protected Attributes

- [CoinPackedMatrix](#) [matrix\\_](#)  
*data Matrix*
- [CoinPackedMatrix](#) [matrixByRow\\_](#)  
*Matrix by row.*
- int \* [rhs\\_](#)  
*Possible rhs (if 0 then not possible)*

#### 7.34.1 Detailed Description

Define a follow on class.

The idea of this is that in air-crew scheduling problems crew may fly in on flight A and out on flight B or on some other flight. A useful branch is one which on one side fixes all which go out on flight B to 0, while the other branch fixes all those that do NOT go out on flight B to 0.

This branching rule should be in addition to normal rules and have a high priority.

Definition at line 25 of file [CbcFollowOn.hpp](#).



## 7.34.2 Constructor &amp; Destructor Documentation

## 7.34.2.1 CbcFollowOn::CbcFollowOn ( )

7.34.2.2 CbcFollowOn::CbcFollowOn ( CbcModel \* *model* )

Useful constructor.

## 7.34.2.3 CbcFollowOn::CbcFollowOn ( const CbcFollowOn &amp; )

## 7.34.2.4 CbcFollowOn::~CbcFollowOn ( )

## 7.34.3 Member Function Documentation

## 7.34.3.1 virtual CbcObject\* CbcFollowOn::clone ( ) const [virtual]

Clone.

Implements [CbcObject](#).

7.34.3.2 CbcFollowOn& CbcFollowOn::operator= ( const CbcFollowOn & *rhs* )7.34.3.3 virtual double CbcFollowOn::infeasibility ( const OsiBranchingInformation \* *info*, int & *preferredWay* ) const [virtual]

Infeasibility - large is 0.5.

Reimplemented from [CbcObject](#).

## 7.34.3.4 virtual void CbcFollowOn::feasibleRegion ( ) [virtual]

This looks at solution and sets bounds to contain solution.

Implements [CbcObject](#).

7.34.3.5 virtual CbcBranchingObject\* CbcFollowOn::createCbcBranch ( OsiSolverInterface \* *solver*, const OsiBranchingInformation \* *info*, int *way* ) [virtual]

Creates a branching object.

Reimplemented from [CbcObject](#).

7.34.3.6 virtual int CbcFollowOn::gutsOfFollowOn ( int & *otherRow*, int & *preferredWay* ) const [virtual]

As some computation is needed in more than one place - returns row.

## 7.34.4 Member Data Documentation

## 7.34.4.1 CoinPackedMatrix CbcFollowOn::matrix\_ [protected]

data Matrix

Definition at line 64 of file CbcFollowOn.hpp.

## 7.34.4.2 CoinPackedMatrix CbcFollowOn::matrixByRow\_ [protected]

Matrix by row.

Definition at line 66 of file CbcFollowOn.hpp.

#### 7.34.4.3 int\* CbcFollowOn::rhs\_ [protected]

Possible rhs (if 0 then not possible)

Definition at line 68 of file CbcFollowOn.hpp.

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

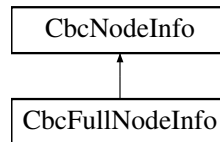
- /home/ted/COIN/trunk/Cbc/src/CbcFollowOn.hpp

### 7.35 CbcFullNodeInfo Class Reference

Information required to recreate the subproblem at this node.

```
#include <CbcFullNodeInfo.hpp>
```

Inheritance diagram for CbcFullNodeInfo:



#### Public Member Functions

- virtual void [applyToModel](#) ([CbcModel](#) \*model, CoinWarmStartBasis \*&basis, [CbcCountRowCut](#) \*\*addCuts, int &currentNumberCuts) const  
*Modify model according to information at node.*
- virtual int [applyBounds](#) (int iColumn, double &lower, double &upper, int force)  
*Just apply bounds to one variable - force means overwrite by lower,upper (1=>infeasible)*
- virtual [CbcNodeInfo](#) \* [buildRowBasis](#) (CoinWarmStartBasis &basis) const  
*Builds up row basis backwards (until original model).*
- [CbcFullNodeInfo](#) ()
- [CbcFullNodeInfo](#) ([CbcModel](#) \*model, int numberOfRowsAtContinuous)  
*Constructor from continuous or satisfied.*
- [CbcFullNodeInfo](#) (const [CbcFullNodeInfo](#) &)
- [~CbcFullNodeInfo](#) ()
- virtual [CbcNodeInfo](#) \* [clone](#) () const  
*Clone.*
- const double \* [lower](#) () const  
*Lower bounds.*
- void [setColLower](#) (int sequence, double value)  
*Set a bound.*
- double \* [mutableLower](#) () const  
*Mutable lower bounds.*
- const double \* [upper](#) () const  
*Upper bounds.*
- void [setColUpper](#) (int sequence, double value)  
*Set a bound.*
- double \* [mutableUpper](#) () const  
*Mutable upper bounds.*

## Protected Attributes

- CoinWarmStartBasis \* [basis\\_](#)  
*Full basis.*
- int [numberIntegers\\_](#)
- double \* [lower\\_](#)
- double \* [upper\\_](#)

## 7.35.1 Detailed Description

Information required to recreate the subproblem at this node.

When a subproblem is initially created, it is represented by a [CbcNode](#) object and an attached [CbcNodeInfo](#) object.

The [CbcNode](#) contains information needed while the subproblem remains live. The [CbcNode](#) is deleted when the last branch arm has been evaluated.

The [CbcNodeInfo](#) contains information required to maintain the branch-and-cut search tree structure (links and reference counts) and to recreate the subproblem for this node (basis, variable bounds, cutting planes). A [CbcNodeInfo](#) object remains in existence until all nodes have been pruned from the subtree rooted at this node.

The principle used to maintain the reference count is that the reference count is always the sum of all potential and actual children of the node. Specifically,

- Once it's determined how the node will branch, the reference count is set to the number of potential children (*i.e.*, the number of arms of the branch).
- As each child is created by [CbcNode::branch\(\)](#) (converting a potential child to the active subproblem), the reference count is decremented.
- If the child survives and will become a node in the search tree (converting the active subproblem into an actual child), increment the reference count.

Notice that the active subproblem lives in a sort of limbo, neither a potential or an actual node in the branch-and-cut tree.

[CbcNodeInfo](#) objects come in two flavours. A [CbcFullNodeInfo](#) object contains a full record of the information required to recreate a subproblem. A [CbcPartialNodeInfo](#) object expresses this information in terms of differences from the parent. Holds complete information for recreating a subproblem.

A [CbcFullNodeInfo](#) object contains all necessary information (bounds, basis, and cuts) required to recreate a subproblem.

**Todo** While there's no explicit statement, the code often makes the implicit assumption that an [CbcFullNodeInfo](#) structure will appear only at the root node of the search tree. Things will break if this assumption is violated.

Definition at line 81 of file [CbcFullNodeInfo.hpp](#).

## 7.35.2 Constructor &amp; Destructor Documentation

## 7.35.2.1 CbcFullNodeInfo::CbcFullNodeInfo ( )

## 7.35.2.2 CbcFullNodeInfo::CbcFullNodeInfo ( CbcModel \* model, int numberRowsAtContinuous )

Constructor from continuous or satisfied.

7.35.2.3 `CbcFullNodeInfo::CbcFullNodeInfo ( const CbcFullNodeInfo & )`

7.35.2.4 `CbcFullNodeInfo::~~CbcFullNodeInfo ( )`

### 7.35.3 Member Function Documentation

7.35.3.1 `virtual void CbcFullNodeInfo::applyToModel ( CbcModel * model, CoinWarmStartBasis *& basis, CbcCountRowCut ** addCuts, int & currentNumberCuts ) const [virtual]`

Modify model according to information at node.

The routine modifies the model according to bound information at node, creates a new basis according to information at node, but with the size passed in through basis, and adds any cuts to the addCuts array.

#### Note

The basis passed in via basis is solely a vehicle for passing in the desired basis size. It will be deleted and a new basis returned.

Implements [CbcNodeInfo](#).

7.35.3.2 `virtual int CbcFullNodeInfo::applyBounds ( int iColumn, double & lower, double & upper, int force ) [virtual]`

Just apply bounds to one variable - force means overwrite by lower,upper (1=>infeasible)

Implements [CbcNodeInfo](#).

7.35.3.3 `virtual CbcNodeInfo* CbcFullNodeInfo::buildRowBasis ( CoinWarmStartBasis & basis ) const [virtual]`

Builds up row basis backwards (until original model).

Returns NULL or previous one to apply . Depends on Free being 0 and impossible for cuts

Implements [CbcNodeInfo](#).

7.35.3.4 `virtual CbcNodeInfo* CbcFullNodeInfo::clone ( ) const [virtual]`

Clone.

Implements [CbcNodeInfo](#).

7.35.3.5 `const double* CbcFullNodeInfo::lower ( ) const [inline]`

Lower bounds.

Definition at line 123 of file CbcFullNodeInfo.hpp.

7.35.3.6 `void CbcFullNodeInfo::setColLower ( int sequence, double value ) [inline]`

Set a bound.

Definition at line 127 of file CbcFullNodeInfo.hpp.

7.35.3.7 `double* CbcFullNodeInfo::mutableLower ( ) const [inline]`

Mutable lower bounds.

Definition at line 130 of file CbcFullNodeInfo.hpp.

7.35.3.8 `const double* CbcFullNodeInfo::upper ( ) const` [inline]

Upper bounds.

Definition at line 134 of file CbcFullNodeInfo.hpp.

7.35.3.9 `void CbcFullNodeInfo::setColUpper ( int sequence, double value )` [inline]

Set a bound.

Definition at line 138 of file CbcFullNodeInfo.hpp.

7.35.3.10 `double* CbcFullNodeInfo::mutableUpper ( ) const` [inline]

Mutable upper bounds.

Definition at line 141 of file CbcFullNodeInfo.hpp.

#### 7.35.4 Member Data Documentation

7.35.4.1 `CoinWarmStartBasis* CbcFullNodeInfo::basis_` [protected]

Full basis.

This MUST BE A POINTER to avoid cutting extra information in derived warm start classes.

Definition at line 151 of file CbcFullNodeInfo.hpp.

7.35.4.2 `int CbcFullNodeInfo::numberIntegers_` [protected]

Definition at line 152 of file CbcFullNodeInfo.hpp.

7.35.4.3 `double* CbcFullNodeInfo::lower_` [protected]

Definition at line 154 of file CbcFullNodeInfo.hpp.

7.35.4.4 `double* CbcFullNodeInfo::upper_` [protected]

Definition at line 155 of file CbcFullNodeInfo.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcFullNodeInfo.hpp](#)

## 7.36 CbcGenCtlBlk Class Reference

```
#include <CbcGenCtlBlk.hpp>
```

### Classes

- struct [babState\\_struct](#)  
*State of branch-and-cut.*
- struct [cbcParamsInfo\\_struct](#)  
*Start and end of [CbcModel](#) parameters in parameter vector.*
- struct [chooseStrongCtl\\_struct](#)  
*Control variables for a strong branching method.*

- struct [debugSolInfo\\_struct](#)  
*Array of primal variable values for debugging.*
- struct [djFixCtl\\_struct](#)  
*Control use of reduced cost fixing prior to B&C.*
- struct [genParamsInfo\\_struct](#)  
*Start and end of cbc-generic parameters in parameter vector.*
- struct [osiParamsInfo\\_struct](#)  
*Start and end of OsiSolverInterface parameters in parameter vector.*

## Public Types

### Enumeration types used for cbc-generic control variables

- enum [IPPCtrl](#) {  
[IPPOff](#) = 0, [IPPOn](#), [IPPSave](#), [IPPEqual](#),  
[IPPSOS](#), [IPPTrySOS](#), [IPPEqualAll](#), [IPPStrategy](#) }  
*Codes to control integer preprocessing.*
- enum [CGCtrl](#) {  
[CGOff](#), [CGOn](#), [CGRoot](#), [CGIfMove](#),  
[CGForceOn](#), [CGForceBut](#), [CGMarker](#) }  
*Codes to control the use of cut generators and heuristics.*
- enum [BPControl](#) { [BPOff](#), [BPCost](#), [BPOrder](#), [BPExt](#) }  
*Codes to specify the assignment of branching priorities.*
- enum [BACMajor](#) {  
[BACInvalid](#) = -1, [BACFinish](#) = 0, [BACStop](#) = 1, [BACAbandon](#) = 2,  
[BACNotRun](#), [BACUser](#) = 5 }  
*Major status codes for branch-and-cut.*
- enum [BACMinor](#) {  
[BACmInvalid](#) = -1, [BACmFinish](#) = 0, [BACmInfeas](#), [BACmUbound](#),  
[BACmGap](#), [BACmNodeLimit](#), [BACmTimeLimit](#), [BACmSolnLimit](#),  
[BACmUser](#), [BACmOther](#) }  
*Minor status codes.*
- enum [BACWhere](#) {  
[BACwInvalid](#) = -1, [BACwNotStarted](#) = 0, [BACwBareRoot](#), [BACwIPP](#),  
[BACwIPPRelax](#), [BACwBAC](#) }  
*Codes to specify where branch-and-cut stopped.*

## Public Member Functions

### Constructors and destructors

- [CbcGenCtlBlk](#) ()  
*Default constructor.*
- [~CbcGenCtlBlk](#) ()  
*Destructor.*

### Access and Control Functions for Cut Generators and Heuristics

*Control functions, plus lazy creation functions for cut generators and heuristics*

*cbc-generic avoids creating objects for cut generators and heuristics unless they're actually used. For cut generators, a prototype is created and reused. For heuristics, the default is to create a new object with each call, because the*

model may have changed. The object is returned through the reference parameter. The return value of the function is the current action state.

Cut generator and heuristic objects created by these calls will be deleted with the destruction of the [CbcGenCtlBlk](#) object.

- `int getCutDepth ()`  
*Get cut depth setting.*
- `void setCutDepth (int cutDepth)`  
*Set cut depth setting.*
- `IPPControl getIPPAction ()`
- `void setIPPAction (IPPControl action)`  
*Set action state for use of integer preprocessing.*
- `CGControl getProbing (CglCutGenerator *&gen)`  
*Obtain a prototype for a probing cut generator.*
- `void setProbingAction (CGControl action)`  
*Set action state for use of probing cut generator.*
- `CGControl getClique (CglCutGenerator *&gen)`  
*Obtain a prototype for a clique cut generator.*
- `void setCliqueAction (CGControl action)`  
*Set action state for use of clique cut generator.*
- `CGControl getFlow (CglCutGenerator *&gen)`  
*Obtain a prototype for a flow cover cut generator.*
- `void setFlowAction (CGControl action)`  
*Set action state for use of flow cover cut generator.*
- `CGControl getGomory (CglCutGenerator *&gen)`  
*Obtain a prototype for a Gomory cut generator.*
- `void setGomoryAction (CGControl action)`  
*Set action state for use of Gomory cut generator.*
- `CGControl getKnapsack (CglCutGenerator *&gen)`  
*Obtain a prototype for a knapsack cover cut generator.*
- `void setKnapsackAction (CGControl action)`  
*Set action state for use of knapsack cut generator.*
- `CGControl getMir (CglCutGenerator *&gen)`  
*Obtain a prototype for a mixed integer rounding (MIR) cut generator.*
- `void setMirAction (CGControl action)`  
*Set action state for use of MIR cut generator.*
- `CGControl getRedSplit (CglCutGenerator *&gen)`  
*Obtain a prototype for a reduce and split cut generator.*
- `void setRedSplitAction (CGControl action)`  
*Set action state for use of reduce and split cut generator.*
- `CGControl getTwomir (CglCutGenerator *&gen)`  
*Obtain a prototype for a 2-MIR cut generator.*
- `void setTwomirAction (CGControl action)`  
*Set action state for use of 2-MIR cut generator.*
- `CGControl getFPump (CbcHeuristic *&gen, CbcModel *model, bool alwaysCreate=true)`  
*Obtain a feasibility pump heuristic.*
- `void setFPumpAction (CGControl action)`  
*Set action state for use of feasibility pump heuristic.*
- `CGControl getCombine (CbcHeuristic *&gen, CbcModel *model, bool alwaysCreate=true)`  
*Obtain a local search/combine heuristic.*
- `void setCombineAction (CGControl action)`  
*Set action state for use of local search/combine heuristic.*
- `CGControl getGreedyCover (CbcHeuristic *&gen, CbcModel *model, bool alwaysCreate=true)`  
*Obtain a greedy cover heuristic.*
- `void setGreedyCoverAction (CGControl action)`

- Set action state for use of greedy cover heuristic.*
- `CGControl getGreedyEquality (CbcHeuristic *&gen, CbcModel *model, bool alwaysCreate=true)`  
*Obtain a greedy equality heuristic.*
- `void setGreedyEqualityAction (CGControl action)`  
*Set action state for use of greedy equality heuristic.*
- `CGControl getRounding (CbcHeuristic *&gen, CbcModel *model, bool alwaysCreate=true)`  
*Obtain a simple rounding heuristic.*
- `void setRoundingAction (CGControl action)`  
*Set action state for use of simple rounding heuristic.*
- `CGControl getTreeLocal (CbcTreeLocal *&localTree, CbcModel *model, bool alwaysCreate=true)`  
*Obtain a local search tree object.*
- `void setTreeLocalAction (CGControl action)`  
*Set action state for use of local tree.*

## Status Functions

Convenience routines for status codes.

- `void setBaBStatus (BACMajor majorStatus, BACMinor minorStatus, BACWhere where, bool haveAnswer, OsiSolverInterface *answerSolver)`  
*Set the result of branch-and-cut search.*
- `void setBaBStatus (const CbcModel *model, BACWhere where, bool haveAnswer=false, OsiSolverInterface *answerSolver=0)`  
*Set the result of branch-and-cut search.*
- `BACMajor translateMajor (int status)`  
*Translate CbcModel major status to BACMajor.*
- `BACMinor translateMinor (int status)`  
*Translate CbcModel minor status to BACMinor.*
- `BACMinor translateMinor (const OsiSolverInterface *osi)`  
*Translate OsiSolverInterface status to BACMinor.*
- `void printBaBStatus ()`  
*Print the status block.*

## Public Attributes

### Parameter parsing and input/output.

- `std::string version_`  
*cbc-generic version*
- `std::string dfltDirectory_`  
*Default directory prefix.*
- `std::string lastMpsIn_`  
*Last MPS input file.*
- `bool allowImportErrors_`  
*Allow/disallow errors when importing a model.*
- `std::string lastSolnOut_`  
*Last solution output file.*
- `int printMode_`  
*Solution printing mode.*
- `std::string printMask_`  
*Print mask.*
- `CoinParamVec * paramVec_`  
*The parameter vector.*
- `struct CbcGenCtlBlk::genParamsInfo_struct genParams_`



- struct  
[CbcGenCtlBlk::cbcParamsInfo\\_struct cbcParams\\_](#)
- struct  
[CbcGenCtlBlk::osiParamsInfo\\_struct osiParams\\_](#)
- int [verbose\\_](#)  
*Verbosity level for help messages.*
- int [paramsProcessed\\_](#)  
*Number of parameters processed.*
- std::vector< bool > [setByUser\\_](#)  
*Record of parameters changed by user command.*
- bool [defaultSettings\\_](#)  
*False if the user has made nontrivial modifications to the default control settings.*
- std::string [debugCreate\\_](#)  
*Control debug file creation.*
- std::string [debugFile\\_](#)  
*Last debug input file.*
- struct  
[CbcGenCtlBlk::debugSolInfo\\_struct debugSol\\_](#)
- double [totalTime\\_](#)  
*Total elapsed time for this run.*

#### Models of various flavours

- [CbcModel](#) \* [model\\_](#)  
*The reference [CbcModel](#) object.*
- [OsiSolverInterface](#) \* [dfltSolver\\_](#)  
*The current default LP solver.*
- bool [goodModel\\_](#)  
*True if we have a valid model loaded, false otherwise.*
- struct  
[CbcGenCtlBlk::babState\\_struct bab\\_](#)

#### Various algorithm control variables and settings

- struct  
[CbcGenCtlBlk::djFixCtl\\_struct djFix\\_](#)
- [BPCControl](#) [priorityAction\\_](#)  
*Control the assignment of branching priorities to integer variables.*

#### Branching Method Control

*Usage control and prototypes for branching methods.*

*Looking to the future, this covers only [OsiChoose](#) methods.*

- struct  
[CbcGenCtlBlk::chooseStrongCtl\\_struct chooseStrong\\_](#)

#### Friends

- void [CbcGenParamUtils::addCbcGenParams](#) (int &numParams, CoinParamVec &paramVec, [CbcGenCtlBlk](#) \*ctl-  
Blk)

## Messages and statistics

- int [printOpt\\_](#)  
*When greater than 0, integer presolve gives more information and branch-and-cut provides statistics.*
- CoinMessageHandler & [message](#) (CbcGenMsgCode inID)  
*Print a message.*
- void [passInMessageHandler](#) (CoinMessageHandler \*handler)  
*Supply a new message handler.*
- CoinMessageHandler \* [messageHandler](#) () const  
*Return a pointer to the message handler.*
- void [setMessages](#) (CoinMessages::Language lang=CoinMessages::us\_en)  
*Set up messages in the specified language.*
- void [setLogLevel](#) (int lvl)  
*Set log level.*
- int [logLevel](#) () const  
*Get log level.*

### 7.36.1 Detailed Description

Definition at line 67 of file CbcGenCtlBlk.hpp.

### 7.36.2 Member Enumeration Documentation

#### 7.36.2.1 enum CbcGenCtlBlk::IPPControl

Codes to control integer preprocessing.

- IPPOff: Integer preprocessing is off.
- IPPOn: Integer preprocessing is on.
- IPPSave: IPPOn, plus preprocessed system will be saved to presolved.mps.
- IPPEqual: IPPOn, plus ' $\leq$ ' cliques are converted to ' $=$ ' cliques.
- IPPSOS: IPPOn, plus will create SOS sets (see below).
- IPPTrySOS: IPPOn, plus will create SOS sets (see below).
- IPPEqualAll: IPPOn, plus turns all valid inequalities into equalities with integer slacks.
- IPPStrategy: look to [CbcStrategy](#) object for instructions.

IPPSOS will create SOS sets if all binary variables (except perhaps one) can be covered by SOS sets with no overlap between sets. IPPTrySOS will allow any number of binary variables to be uncovered.

#### Enumerator

***IPPOff***

***IPPOn***

***IPPSave***

***IPPEqual***

***IPPSOS***

***IPPTrySOS***

***IPPEqualAll***

***IPPStrategy***

Definition at line 99 of file CbcGenCtlBlk.hpp.

#### 7.36.2.2 enum CbcGenCtlBlk::CGControl

Codes to control the use of cut generators and heuristics.

- CGOff: the cut generator will not be installed
- CGOn: the cut generator will be installed; exactly how often it's activated depends on the settings at installation
- CGRoot: the cut generator will be installed with settings that restrict it to activation at the root node only.
- CGIfMove: the cut generator will be installed with settings that allow it to remain active only so long as it's generating cuts that tighten the relaxation.
- CGForceOn: the cut generator will be installed with settings that force it to be called at every node
- CGForceBut: the cut generator will be installed with settings that force it to be called at every node, but more active at root (probing only)
- CGMarker: a convenience to mark the end of the codes.

The same codes are used for heuristics.

Enumerator

***CGOff***

***CGOn***

***CGRoot***

***CGIfMove***

***CGForceOn***

***CGForceBut***

***CGMarker***

Definition at line 129 of file CbcGenCtlBlk.hpp.

#### 7.36.2.3 enum CbcGenCtlBlk::BPCControl

Codes to specify the assignment of branching priorities.

- BPOff: no priorities are passed to cbc
- BPCost: a priority vector is constructed based on objective coefficients
- BPOrder: a priority vector is constructed based on column order
- BPExt: the user has provided a priority vector

Enumerator

***BPOff***

***BPCost***

***BPOrder***

***BPExt***

Definition at line 141 of file CbcGenCtlBlk.hpp.

#### 7.36.2.4 enum CbcGenCtlBlk::BACMajor

Major status codes for branch-and-cut.

- BACInvalid: status not yet set
- BACNotRun: branch-and-cut has not yet run for the current problem
- BACFinish: branch-and-cut has finished normally
- BACStop: branch-and-cut has stopped on a limit
- BACAbandon: branch-and-cut abandoned the problem
- BACUser: branch-and-cut stopped on user signal

Consult minorStatus\_ for details.

These codes are (mostly) set to match the codes used by [CbcModel](#). Additions to [CbcModel](#) codes should be reflected here and in translateMajor.

Enumerator

***BACInvalid***

***BACFinish***

***BACStop***

***BACAbandon***

***BACNotRun***

***BACUser***

Definition at line 158 of file CbcGenCtlBlk.hpp.

#### 7.36.2.5 enum CbcGenCtlBlk::BACMinor

Minor status codes.

- BACmInvalid status not yet set
- BACmFinish search exhausted the tree; optimal solution found
- BACmInfeas problem is infeasible
- BACmUbnd problem is unbounded
- BACmGap stopped on integrality gap
- BACmNodeLimit stopped on node limit
- BACmTimeLimit stopped on time limit
- BACmSolnLimit stopped on number of solutions limit

- BACmUser stopped due to user event
- BACmOther nothing else is appropriate

It's not possible to make these codes agree with [CbcModel](#). The meaning varies according to context: if the BACWhere code specifies a relaxation, then the minor status reflects the underlying OSI solver. Otherwise, it reflects the integer problem.

Enumerator

***BACmInvalid***  
***BACmFinish***  
***BACmInfeas***  
***BACmUbnd***  
***BACmGap***  
***BACmNodeLimit***  
***BACmTimeLimit***  
***BACmSolnLimit***  
***BACmUser***  
***BACmOther***

Definition at line 181 of file CbcGenCtlBlk.hpp.

#### 7.36.2.6 enum CbcGenCtlBlk::BACWhere

Codes to specify where branch-and-cut stopped.

- BACwNotStarted stopped before we ever got going
- BACwBareRoot stopped after initial solve of root relaxation
- BACwIPP stopped after integer preprocessing
- BACwIPPRelax stopped after initial solve of preprocessed problem
- BACwBAC stopped at some point in branch-and-cut

Enumerator

***BACwInvalid***  
***BACwNotStarted***  
***BACwBareRoot***  
***BACwIPP***  
***BACwIPPRelax***  
***BACwBAC***

Definition at line 195 of file CbcGenCtlBlk.hpp.

#### 7.36.3 Constructor & Destructor Documentation

##### 7.36.3.1 CbcGenCtlBlk::CbcGenCtlBlk ( )

Default constructor.

### 7.36.3.2 CbcGenCtlBlk::~~CbcGenCtlBlk ( )

Destructor.

## 7.36.4 Member Function Documentation

### 7.36.4.1 int CbcGenCtlBlk::getCutDepth ( ) [inline]

Get cut depth setting.

The name is a bit of a misnomer. Essentially, this overrides the 'every so many nodes' control with 'execute when (depth in tree) mod (cut depth) == 0'.

Definition at line 236 of file CbcGenCtlBlk.hpp.

### 7.36.4.2 void CbcGenCtlBlk::setCutDepth ( int *cutDepth* ) [inline]

Set cut depth setting.

See comments for [getCutDepth\(\)](#).

Definition at line 245 of file CbcGenCtlBlk.hpp.

### 7.36.4.3 IPPControl CbcGenCtlBlk::getPPAction ( ) [inline]

Definition at line 251 of file CbcGenCtlBlk.hpp.

### 7.36.4.4 void CbcGenCtlBlk::setPPAction ( IPPControl *action* ) [inline]

Set action state for use of integer preprocessing.

Definition at line 257 of file CbcGenCtlBlk.hpp.

### 7.36.4.5 CGControl CbcGenCtlBlk::getProbing ( CglCutGenerator \*& *gen* )

Obtain a prototype for a probing cut generator.

### 7.36.4.6 void CbcGenCtlBlk::setProbingAction ( CGControl *action* ) [inline]

Set action state for use of probing cut generator.

Definition at line 267 of file CbcGenCtlBlk.hpp.

### 7.36.4.7 CGControl CbcGenCtlBlk::getClique ( CglCutGenerator \*& *gen* )

Obtain a prototype for a clique cut generator.

### 7.36.4.8 void CbcGenCtlBlk::setCliqueAction ( CGControl *action* ) [inline]

Set action state for use of clique cut generator.

Definition at line 277 of file CbcGenCtlBlk.hpp.

### 7.36.4.9 CGControl CbcGenCtlBlk::getFlow ( CglCutGenerator \*& *gen* )

Obtain a prototype for a flow cover cut generator.

### 7.36.4.10 void CbcGenCtlBlk::setFlowAction ( CGControl *action* ) [inline]

Set action state for use of flow cover cut generator.

Definition at line 287 of file CbcGenCtlBlk.hpp.

**7.36.4.11** `CGControl CbcGenCtlBlk::getGomory ( CglCutGenerator *& gen )`

Obtain a prototype for a Gomory cut generator.

**7.36.4.12** `void CbcGenCtlBlk::setGomoryAction ( CGControl action ) [inline]`

Set action state for use of Gomory cut generator.

Definition at line 297 of file CbcGenCtlBlk.hpp.

**7.36.4.13** `CGControl CbcGenCtlBlk::getKnapsack ( CglCutGenerator *& gen )`

Obtain a prototype for a knapsack cover cut generator.

**7.36.4.14** `void CbcGenCtlBlk::setKnapsackAction ( CGControl action ) [inline]`

Set action state for use of knapsack cut generator.

Definition at line 307 of file CbcGenCtlBlk.hpp.

**7.36.4.15** `CGControl CbcGenCtlBlk::getMir ( CglCutGenerator *& gen )`

Obtain a prototype for a mixed integer rounding (MIR) cut generator.

**7.36.4.16** `void CbcGenCtlBlk::setMirAction ( CGControl action ) [inline]`

Set action state for use of MIR cut generator.

Definition at line 329 of file CbcGenCtlBlk.hpp.

**7.36.4.17** `CGControl CbcGenCtlBlk::getRedSplit ( CglCutGenerator *& gen )`

Obtain a prototype for a reduce and split cut generator.

**7.36.4.18** `void CbcGenCtlBlk::setRedSplitAction ( CGControl action ) [inline]`

Set action state for use of reduce and split cut generator.

Definition at line 339 of file CbcGenCtlBlk.hpp.

**7.36.4.19** `CGControl CbcGenCtlBlk::getTwomir ( CglCutGenerator *& gen )`

Obtain a prototype for a 2-MIR cut generator.

**7.36.4.20** `void CbcGenCtlBlk::setTwomirAction ( CGControl action ) [inline]`

Set action state for use of 2-MIR cut generator.

Definition at line 349 of file CbcGenCtlBlk.hpp.

**7.36.4.21** `CGControl CbcGenCtlBlk::getFPump ( CbcHeuristic *& gen, CbcModel * model, bool alwaysCreate = true )`

Obtain a feasibility pump heuristic.

By default, any existing object is deleted and a new object is created and loaded with `model`. Set `alwaysCreate = false` to return an existing object if one exists.

**7.36.4.22** `void CbcGenCtlBlk::setFPumpAction ( CGControl action ) [inline]`

Set action state for use of feasibility pump heuristic.

Definition at line 366 of file CbcGenCtlBlk.hpp.

**7.36.4.23** `CGControl CbcGenCtlBlk::getCombine ( CbcHeuristic *& gen, CbcModel * model, bool alwaysCreate = true )`

Obtain a local search/combine heuristic.

By default, any existing object is deleted and a new object is created and loaded with `model`. Set `alwaysCreate = false` to return an existing object if one exists.

**7.36.4.24** `void CbcGenCtlBlk::setCombineAction ( CGControl action ) [inline]`

Set action state for use of local search/combine heuristic.

Definition at line 382 of file CbcGenCtlBlk.hpp.

**7.36.4.25** `CGControl CbcGenCtlBlk::getGreedyCover ( CbcHeuristic *& gen, CbcModel * model, bool alwaysCreate = true )`

Obtain a greedy cover heuristic.

By default, any existing object is deleted and a new object is created and loaded with `model`. Set `alwaysCreate = false` to return an existing object if one exists.

**7.36.4.26** `void CbcGenCtlBlk::setGreedyCoverAction ( CGControl action ) [inline]`

Set action state for use of greedy cover heuristic.

Definition at line 398 of file CbcGenCtlBlk.hpp.

**7.36.4.27** `CGControl CbcGenCtlBlk::getGreedyEquality ( CbcHeuristic *& gen, CbcModel * model, bool alwaysCreate = true )`

Obtain a greedy equality heuristic.

By default, any existing object is deleted and a new object is created and loaded with `model`. Set `alwaysCreate = false` to return an existing object if one exists.

**7.36.4.28** `void CbcGenCtlBlk::setGreedyEqualityAction ( CGControl action ) [inline]`

Set action state for use of greedy equality heuristic.

Definition at line 414 of file CbcGenCtlBlk.hpp.

**7.36.4.29** `CGControl CbcGenCtlBlk::getRounding ( CbcHeuristic *& gen, CbcModel * model, bool alwaysCreate = true )`

Obtain a simple rounding heuristic.

By default, any existing object is deleted and a new object is created and loaded with `model`. Set `alwaysCreate = false` to return an existing object if one exists.

**7.36.4.30** `void CbcGenCtlBlk::setRoundingAction ( CGControl action ) [inline]`

Set action state for use of simple rounding heuristic.

Definition at line 430 of file CbcGenCtlBlk.hpp.



**7.36.4.31** `CGControl CbcGenCtlBlk::getTreeLocal ( CbcTreeLocal *& localTree, CbcModel * model, bool alwaysCreate = true )`

Obtain a local search tree object.

By default, any existing object is deleted and a new object is created and loaded with `model`. Set `alwaysCreate = false` to return an existing object if one exists.

**7.36.4.32** `void CbcGenCtlBlk::setTreeLocalAction ( CGControl action ) [inline]`

Set action state for use of local tree.

Definition at line 446 of file `CbcGenCtlBlk.hpp`.

**7.36.4.33** `void CbcGenCtlBlk::setBaBStatus ( BACMajor majorStatus, BACMinor minorStatus, BACWhere where, bool haveAnswer, OsiSolverInterface * answerSolver ) [inline]`

Set the result of branch-and-cut search.

Definition at line 459 of file `CbcGenCtlBlk.hpp`.

**7.36.4.34** `void CbcGenCtlBlk::setBaBStatus ( const CbcModel * model, BACWhere where, bool haveAnswer = false, OsiSolverInterface * answerSolver = 0 )`

Set the result of branch-and-cut search.

This version will extract the necessary information from the [CbcModel](#) object and set appropriate status based on the value passed for `where`.

**7.36.4.35** `BACMajor CbcGenCtlBlk::translateMajor ( int status )`

Translate [CbcModel](#) major status to [BACMajor](#).

See the [BACMajor](#) enum for details.

**7.36.4.36** `BACMinor CbcGenCtlBlk::translateMinor ( int status )`

Translate [CbcModel](#) minor status to [BACMinor](#).

See the [BACMinor](#) enum for details.

**7.36.4.37** `BACMinor CbcGenCtlBlk::translateMinor ( const OsiSolverInterface * osi )`

Translate `OsiSolverInterface` status to [BACMinor](#).

See the [BACMinor](#) enum for details. Optimal, infeasible, and unbounded get their own codes; everything else maps to `BACmOther`.

**7.36.4.38** `void CbcGenCtlBlk::printBaBStatus ( )`

Print the status block.

**7.36.4.39** `CoinMessageHandler& CbcGenCtlBlk::message ( CbcGenMsgCode inID )`

Print a message.

Uses the current message handler and messages.

**7.36.4.40** `void CbcGenCtlBlk::passInMessageHandler ( CoinMessageHandler * handler )`

Supply a new message handler.

Replaces the current message handler. The current handler is destroyed if `ourMsgHandler_` is true, and the call will set `ourMsgHandler_ = true`.

**7.36.4.41** `CoinMessageHandler* CbcGenCtlBlk::messageHandler ( ) const [inline]`

Return a pointer to the message handler.

Definition at line 520 of file `CbcGenCtlBlk.hpp`.

**7.36.4.42** `void CbcGenCtlBlk::setMessages ( CoinMessages::Language lang = CoinMessages::us_en )`

Set up messages in the specified language.

Building a set of messages in a given language implies rebuilding the whole set of messages, for reasons explained in the body of the code. Hence there's no separate `setLanguage` routine. Use this routine for the initial setup of messages and any subsequent change in language. Note that the constructor gives you a message handler by default, but *not* messages. You need to call `setMessages` explicitly.

The default value specified here for `lang` effectively sets the default language.

**7.36.4.43** `void CbcGenCtlBlk::setLogLevel ( int lvl ) [inline]`

Set log level.

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

**7.36.4.44** `int CbcGenCtlBlk::logLevel ( ) const [inline]`

Get log level.

Definition at line 545 of file `CbcGenCtlBlk.hpp`.

## 7.36.5 Friends And Related Function Documentation

**7.36.5.1** `void CbcGenParamUtils::addCbcGenParams ( int & numParams, CoinParamVec & paramVec, CbcGenCtlBlk * ctlBlk ) [friend]`

## 7.36.6 Member Data Documentation

**7.36.6.1** `int CbcGenCtlBlk::printOpt_`

When greater than 0, integer presolve gives more information and branch-and-cut provides statistics.

Definition at line 552 of file `CbcGenCtlBlk.hpp`.

**7.36.6.2** `std::string CbcGenCtlBlk::version_`

cbc-generic version

Definition at line 560 of file `CbcGenCtlBlk.hpp`.

**7.36.6.3** `std::string CbcGenCtlBlk::dfltDirectory_`

Default directory prefix.

Definition at line 564 of file `CbcGenCtlBlk.hpp`.

**7.36.6.4** `std::string CbcGenCtlBlk::lastMpsIn_`

Last MPS input file.

Definition at line 568 of file CbcGenCtlBlk.hpp.

#### 7.36.6.5 bool CbcGenCtlBlk::allowImportErrors\_

Allow/disallow errors when importing a model.

Definition at line 571 of file CbcGenCtlBlk.hpp.

#### 7.36.6.6 std::string CbcGenCtlBlk::lastSolnOut\_

Last solution output file.

Definition at line 575 of file CbcGenCtlBlk.hpp.

#### 7.36.6.7 int CbcGenCtlBlk::printMode\_

Solution printing mode.

Controls the amount of information printed when printing a solution. Coding is set by the keyword declarations for the printingOptions command.

Definition at line 583 of file CbcGenCtlBlk.hpp.

#### 7.36.6.8 std::string CbcGenCtlBlk::printMask\_

Print mask.

Used to specify row/column names to be printed. Not implemented as of 060920.

Definition at line 590 of file CbcGenCtlBlk.hpp.

#### 7.36.6.9 CoinParamVec\* CbcGenCtlBlk::paramVec\_

The parameter vector.

Definition at line 594 of file CbcGenCtlBlk.hpp.

#### 7.36.6.10 struct CbcGenCtlBlk::genParamsInfo\_struct CbcGenCtlBlk::genParams\_

#### 7.36.6.11 struct CbcGenCtlBlk::cbcParamsInfo\_struct CbcGenCtlBlk::cbcParams\_

#### 7.36.6.12 struct CbcGenCtlBlk::osiParamsInfo\_struct CbcGenCtlBlk::osiParams\_

#### 7.36.6.13 int CbcGenCtlBlk::verbose\_

Verbosity level for help messages.

Interpretation is bitwise:

- (0): short help
- (1): long help
- (2): unused (for compatibility with cbc; indicates AMPL)
- (3): show parameters with display = false.

Definition at line 628 of file CbcGenCtlBlk.hpp.

#### 7.36.6.14 int CbcGenCtlBlk::paramsProcessed\_

Number of parameters processed.

Definition at line 632 of file CbcGenCtlBlk.hpp.

#### 7.36.6.15 `std::vector<bool> CbcGenCtlBlk::setByUser_`

Record of parameters changed by user command.

Definition at line 636 of file CbcGenCtlBlk.hpp.

#### 7.36.6.16 `bool CbcGenCtlBlk::defaultSettings_`

False if the user has made nontrivial modifications to the default control settings.

Initially true. Specifying DJFIX, TIGHTENFACTOR, or any cut or heuristic parameter will set this to false.

Definition at line 644 of file CbcGenCtlBlk.hpp.

#### 7.36.6.17 `std::string CbcGenCtlBlk::debugCreate_`

Control debug file creation.

At the conclusion of branch-and-cut, dump the full solution in a binary format to debug.file in the current directory. When set to "createAfterPre", the solution is dumped before integer presolve transforms are removed. When set to "create", the solution is dumped after integer presolve transforms are backed out.

Definition at line 654 of file CbcGenCtlBlk.hpp.

#### 7.36.6.18 `std::string CbcGenCtlBlk::debugFile_`

Last debug input file.

The file is expected to be in a binary format understood by activateRowCutDebugger.

Definition at line 662 of file CbcGenCtlBlk.hpp.

#### 7.36.6.19 `struct CbcGenCtlBlk::debugSolInfo_ struct CbcGenCtlBlk::debugSol_`

#### 7.36.6.20 `double CbcGenCtlBlk::totalTime_`

Total elapsed time for this run.

Definition at line 680 of file CbcGenCtlBlk.hpp.

#### 7.36.6.21 `CbcModel* CbcGenCtlBlk::model_`

The reference [CbcModel](#) object.

This is the [CbcModel](#) created when cbc-generic boots up. It holds the default solver with the current constraint system. [CbcCbcParam](#) parameters are applied here, and [CbcOsiParam](#) parameters are applied to the solver. Major modifications for branch-and-cut (integer preprocessing, installation of heuristics and cut generators) are performed on a clone. The solution is transferred back into this object.

Definition at line 697 of file CbcGenCtlBlk.hpp.

#### 7.36.6.22 `OsiSolverInterface* CbcGenCtlBlk::dfltSolver_`

The current default LP solver.

This is a pointer to a reference copy. If you want the solver associated with [model\\_](#), ask for it directly.

Definition at line 705 of file CbcGenCtlBlk.hpp.

7.36.6.23 `bool CbcGenCtlBlk::goodModel_`

True if we have a valid model loaded, false otherwise.

Definition at line 709 of file CbcGenCtlBlk.hpp.

7.36.6.24 `struct CbcGenCtlBlk::babState_struct CbcGenCtlBlk::bab_`7.36.6.25 `struct CbcGenCtlBlk::djFixCtl_struct CbcGenCtlBlk::djFix_`7.36.6.26 `BPCControl CbcGenCtlBlk::priorityAction_`

Control the assignment of branching priorities to integer variables.

Definition at line 747 of file CbcGenCtlBlk.hpp.

7.36.6.27 `struct CbcGenCtlBlk::chooseStrongCtl_struct CbcGenCtlBlk::chooseStrong_`

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

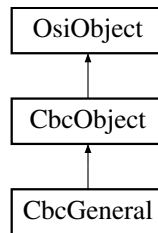
- [/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp](#)

## 7.37 CbcGeneral Class Reference

Define a catch all class.

```
#include <CbcGeneral.hpp>
```

Inheritance diagram for CbcGeneral:



## Public Member Functions

- [CbcGeneral](#) ()
- [CbcGeneral](#) ([CbcModel](#) \**model*)  
*Useful constructor Just needs to point to model.*
- [CbcGeneral](#) (const [CbcGeneral](#) &)
- virtual [CbcObject](#) \* *clone* () const =0  
*Clone.*
- [CbcGeneral](#) & *operator=* (const [CbcGeneral](#) &*rhs*)
- [~CbcGeneral](#) ()
- virtual double *infeasibility* (const [OsiBranchingInformation](#) \**info*, int &*preferredWay*) const  
*Infeasibility - large is 0.5.*
- virtual void *feasibleRegion* ()=0  
*This looks at solution and sets bounds to contain solution.*

- virtual [CbcBranchingObject](#) \* [createCbcBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way)  
*Creates a branching object.*
- virtual void [redoSequenceEtc](#) (CbcModel \*model, int numberColumns, const int \*originalColumns)=0  
*Redoes data when sequence numbers change.*

## Additional Inherited Members

### 7.37.1 Detailed Description

Define a catch all class.

This will create a list of subproblems

Definition at line 17 of file CbcGeneral.hpp.

### 7.37.2 Constructor & Destructor Documentation

#### 7.37.2.1 CbcGeneral::CbcGeneral ( )

#### 7.37.2.2 CbcGeneral::CbcGeneral ( CbcModel \* model )

Useful constructor Just needs to point to model.

#### 7.37.2.3 CbcGeneral::CbcGeneral ( const CbcGeneral & )

#### 7.37.2.4 CbcGeneral::~~CbcGeneral ( )

### 7.37.3 Member Function Documentation

#### 7.37.3.1 virtual CbcObject\* CbcGeneral::clone ( ) const [pure virtual]

Clone.

Implements [CbcObject](#).

#### 7.37.3.2 CbcGeneral& CbcGeneral::operator= ( const CbcGeneral & rhs )

#### 7.37.3.3 virtual double CbcGeneral::infeasibility ( const OsiBranchingInformation \* info, int & preferredWay ) const [virtual]

Infeasibility - large is 0.5.

Reimplemented from [CbcObject](#).

#### 7.37.3.4 virtual void CbcGeneral::feasibleRegion ( ) [pure virtual]

This looks at solution and sets bounds to contain solution.

Implements [CbcObject](#).

#### 7.37.3.5 virtual CbcBranchingObject\* CbcGeneral::createCbcBranch ( OsiSolverInterface \* solver, const OsiBranchingInformation \* info, int way ) [virtual]

Creates a branching object.

Reimplemented from [CbcObject](#).

7.37.3.6 `virtual void CbcGeneral::redoSequenceEtc ( CbcModel * model, int numberColumns, const int * originalColumns )`  
`[pure virtual]`

Redoes data when sequence numbers change.

Reimplemented from [CbcObject](#).

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

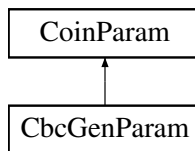
- [/home/ted/COIN/trunk/Cbc/src/CbcGeneral.hpp](#)

## 7.38 CbcGenParam Class Reference

Class for cbc-generic control parameters.

```
#include <CbcGenParam.hpp>
```

Inheritance diagram for CbcGenParam:



### Public Types

#### Subtypes

- enum [CbcGenParamCode](#) {  
[CBCGEN\\_FIRSTPARAM](#) = 0, [GENERALQUERY](#), [FULLGENERALQUERY](#), [HELP](#),  
[BAB](#), [CLEARCUTS](#), [CLIQUECUTS](#), [COMBINE](#),  
[COSTSTRATEGY](#), [CPP](#), [CUTDEPTH](#), [CUTSTRATEGY](#),  
[DEBUG](#), [DIRECTORY](#), [DJFIX](#), [DUMMY](#),  
[ERRORSALLOWED](#), [EXIT](#), [EXPORT](#), [FLOWCUTS](#),  
[FPUMP](#), [FPUMPITS](#), [GOMORYCUTS](#), [GREEDY](#),  
[HEURISTICSTRATEGY](#), [IMPORT](#), [INTPRINT](#), [KNAPSACKCUTS](#),  
[LOCALTREE](#), [LOGLEVEL](#), [MESSAGES](#), [MIPLIB](#),  
[MIXEDCUTS](#), [ODDHOLECUTS](#), [OUTDUPROWS](#), [OUTPUTFORMAT](#),  
[PREPROCESS](#), [PRINTMASK](#), [PRINTOPTIONS](#), [PRINTVERSION](#),  
[PRIORITYIN](#), [PROBINGCUTS](#), [REDSPLITCUTS](#), [ROUNDING](#),  
[SOLUTION](#), [SOLVECONTINUOUS](#), [SOLVER](#), [SOS](#),  
[STDIN](#), [STRENGTHEN](#), [TIGHTENFACTOR](#), [TWOIRMCUTS](#),  
[UNITTEST](#), [USERCBC](#), [USESOLUTION](#), [VERBOSE](#),  
[SHOWUNIMP](#), [CBCGEN\\_LASTPARAM](#) }  
*Enumeration for cbc-generic parameters.*

### Public Member Functions

#### Constructors and Destructors

*Be careful how you specify parameters for the constructors! There's great potential for confusion.*

- [CbcGenParam](#) ()

- Default constructor.*
- `CbcGenParam` (`CbcGenParamCode` code, `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.*
- `CbcGenParam` (`CbcGenParamCode` code, `std::string` name, `std::string` help, int lower, int upper, int dflt=0, bool display=true)
- Constructor for a parameter with an integer value.*
- `CbcGenParam` (`CbcGenParamCode` code, `std::string` name, `std::string` help, `std::string` firstValue, int dflt, bool display=true)
- Constructor for a parameter with keyword values.*
- `CbcGenParam` (`CbcGenParamCode` code, `std::string` name, `std::string` help, `std::string` dflt, bool display=true)
- Constructor for a string parameter.*
- `CbcGenParam` (`CbcGenParamCode` code, `std::string` name, `std::string` help, bool display=true)
- Constructor for an action parameter.*
- `CbcGenParam` (const `CbcGenParam` &orig)
- Copy constructor.*
- `CbcGenParam * clone` ()
- Clone.*
- `CbcGenParam & operator=` (const `CbcGenParam` &rhs)
- Assignment.*
- `~CbcGenParam` ()
- Destructor.*

### Methods to query and manipulate a parameter object

- `CbcGenParamCode paramCode` () const
- Get the parameter code.*
- void `setParamCode` (`CbcGenParamCode` code)
- Set the parameter code.*
- `CbcGenCtlBlk * obj` () const
- Get the underlying cbc-generic control object.*
- void `setObj` (`CbcGenCtlBlk *obj`)
- Set the underlying cbc-generic control object.*

#### 7.38.1 Detailed Description

Class for cbc-generic control parameters.

Adds parameter type codes and push/pull functions to the generic parameter object.

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

#### 7.38.2 Member Enumeration Documentation

##### 7.38.2.1 enum `CbcGenParam::CbcGenParamCode`

Enumeration for cbc-generic parameters.

These are parameters that control the operation of the cbc-generic main program by operating on a `CbcGenCtlBlk` object. `CBCGEN_FIRSTPARAM` and `CBCGEN_LASTPARAM` are markers to allow convenient separation of parameter groups.

Enumerator

**`CBCGEN_FIRSTPARAM`**



**GENERALQUERY**  
**FULLGENERALQUERY**  
**HELP**  
**BAB**  
**CLEARCUTS**  
**CLIQUECUTS**  
**COMBINE**  
**COSTSTRATEGY**  
**CPP**  
**CUTDEPTH**  
**CUTSTRATEGY**  
**DEBUG**  
**DIRECTORY**  
**DJFIX**  
**DUMMY**  
**ERRORSALLOWED**  
**EXIT**  
**EXPORT**  
**FLOWCUTS**  
**FPUMP**  
**FPUMPITS**  
**GOMORYCUTS**  
**GREEDY**  
**HEURISTICSTRATEGY**  
**IMPORT**  
**INTPRINT**  
**KNAPSACKCUTS**  
**LOCALTREE**  
**LOGLEVEL**  
**MESSAGES**  
**MIPLIB**  
**MIXEDCUTS**  
**ODDHOLECUTS**  
**OUTDUPROWS**  
**OUTPUTFORMAT**  
**PREPROCESS**  
**PRINTMASK**  
**PRINTOPTIONS**  
**PRINTVERSION**  
**PRIORITYIN**  
**PROBINGCUTS**  
**REDSPLITCUTS**

**ROUNDING**  
**SOLUTION**  
**SOLVECONTINUOUS**  
**SOLVER**  
**SOS**  
**STDIN**  
**STRENGTHEN**  
**TIGHTENFACTOR**  
**TWOMIRCUTS**  
**UNITTEST**  
**USERCBC**  
**USESOLUTION**  
**VERBOSE**  
**SHOWUNIMP**  
**CBCGEN\_LASTPARAM**

Definition at line 49 of file CbcGenParam.hpp.

### 7.38.3 Constructor & Destructor Documentation

#### 7.38.3.1 CbcGenParam::CbcGenParam ( )

Default constructor.

7.38.3.2 CbcGenParam::CbcGenParam ( CbcGenParamCode code, 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.

7.38.3.3 CbcGenParam::CbcGenParam ( CbcGenParamCode code, 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.

7.38.3.4 CbcGenParam::CbcGenParam ( CbcGenParamCode code, 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 keyword. Additional keywords can be added using `appendKwd()`. Keywords are numbered from zero. It's necessary to specify both the first keyword (`firstValue`) and the default keyword index (`dflt`) in order to distinguish this constructor from the string and action parameter constructors.

7.38.3.5 CbcGenParam::CbcGenParam ( CbcGenParamCode code, std::string name, std::string help, std::string dflt, bool display = true )

Constructor for a string parameter.

The default string value must be specified explicitly to distinguish a string constructor from an action parameter constructor.

**7.38.3.6 CbcGenParam::CbcGenParam ( CbcGenParamCode *code*, std::string *name*, std::string *help*, bool *display* = true )**

Constructor for an action parameter.

**7.38.3.7 CbcGenParam::CbcGenParam ( const CbcGenParam & *orig* )**

Copy constructor.

**7.38.3.8 CbcGenParam::~CbcGenParam ( )**

Destructor.

#### 7.38.4 Member Function Documentation

**7.38.4.1 CbcGenParam\* CbcGenParam::clone ( )**

Clone.

**7.38.4.2 CbcGenParam& CbcGenParam::operator= ( const CbcGenParam & *rhs* )**

Assignment.

**7.38.4.3 CbcGenParamCode CbcGenParam::paramCode ( ) const** [inline]

Get the parameter code.

Definition at line 148 of file CbcGenParam.hpp.

**7.38.4.4 void CbcGenParam::setParamCode ( CbcGenParamCode *code* )** [inline]

Set the parameter code.

Definition at line 154 of file CbcGenParam.hpp.

**7.38.4.5 CbcGenCtlBlk\* CbcGenParam::obj ( ) const** [inline]

Get the underlying cbc-generic control object.

Definition at line 160 of file CbcGenParam.hpp.

**7.38.4.6 void CbcGenParam::setObj ( CbcGenCtlBlk \* *obj* )** [inline]

Set the underlying cbc-generic control object.

Definition at line 166 of file CbcGenParam.hpp.

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

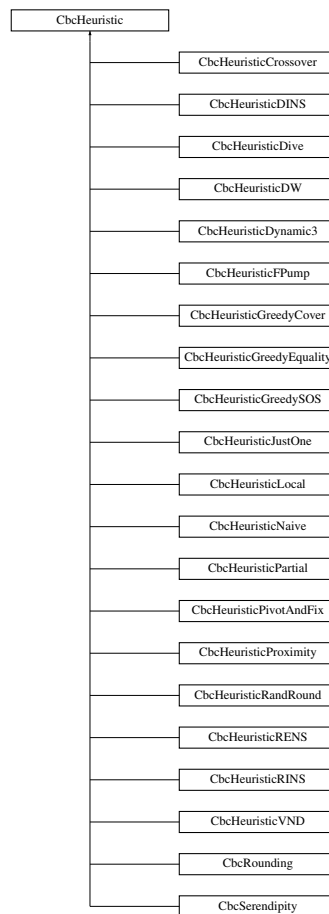
- [/home/ted/COIN/trunk/Cbc/src/CbcGenParam.hpp](#)

## 7.39 CbcHeuristic Class Reference

Heuristic base class.

```
#include <CbcHeuristic.hpp>
```

Inheritance diagram for CbcHeuristic:



### Public Member Functions

- [CbcHeuristic](#) ()
- [CbcHeuristic](#) ([CbcModel](#) &model)
- [CbcHeuristic](#) (const [CbcHeuristic](#) &)
- virtual [~CbcHeuristic](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const =0  
*Clone.*
- [CbcHeuristic](#) & [operator=](#) (const [CbcHeuristic](#) &rhs)  
*Assignment operator.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual void [resetModel](#) ([CbcModel](#) \*model)=0  
*Resets stuff if model changes.*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)=0  
*returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value This is called after cuts have been added - so can not add cuts*
- virtual int [solution2](#) (double &, double \*, [OsiCuts](#) &)

returns 0 if no solution, 1 if valid solution, -1 if just returning an estimate of best possible solution with better objective value than one passed in Sets solution values if good, sets objective value (only if nonzero code) This is called at same time as cut generators - so can add cuts Default is do nothing

- virtual void **validate** ()  
Validate model i.e. sets when\_ to 0 if necessary (may be NULL)
- void **setWhen** (int value)  
Sets "when" flag - 0 off, 1 at root, 2 other than root, 3 always.
- int **when** () const  
Gets "when" flag - 0 off, 1 at root, 2 other than root, 3 always.
- void **setNumberNodes** (int value)  
Sets number of nodes in subtree (default 200)
- int **numberNodes** () const  
Gets number of nodes in a subtree (default 200)
- void **setSwitches** (int value)  
Switches (does not apply equally to all heuristics) 1 bit - stop once allowable gap on objective reached 2 bit - always do given number of passes 4 bit - weaken cutoff by 5% every 50 passes? 8 bit - if has cutoff and suminf bobbling for 20 passes then first try halving distance to best possible then try keep halving distance to known cutoff 16 bit - needs new solution to run 1024 bit - stop all heuristics on max time.
- int **switches** () const  
Switches (does not apply equally to all heuristics) 1 bit - stop once allowable gap on objective reached 2 bit - always do given number of passes 4 bit - weaken cutoff by 5% every 50 passes? 8 bit - if has cutoff and suminf bobbling for 20 passes then first try halving distance to best possible then try keep halving distance to known cutoff 16 bit - needs new solution to run 1024 bit - stop all heuristics on max time.
- bool **exitNow** (double bestObjective) const  
Whether to exit at once on gap.
- void **setFeasibilityPumpOptions** (int value)  
Sets feasibility pump options (-1 is off)
- int **feasibilityPumpOptions** () const  
Gets feasibility pump options (-1 is off)
- void **setModelOnly** (CbcModel \*model)  
Just set model - do not do anything else.
- void **setFractionSmall** (double value)  
Sets fraction of new(rows+columns)/old(rows+columns) before doing small branch and bound (default 1.0)
- double **fractionSmall** () const  
Gets fraction of new(rows+columns)/old(rows+columns) before doing small branch and bound (default 1.0)
- int **numberSolutionsFound** () const  
Get how many solutions the heuristic thought it got.
- void **incrementNumberSolutionsFound** ()  
Increment how many solutions the heuristic thought it got.
- int **smallBranchAndBound** (OsiSolverInterface \*solver, int **numberNodes**, double \*newSolution, double &newSolutionValue, double cutoff, std::string name) const  
Do mini branch and bound - return 0 not finished - no solution 1 not finished - solution 2 finished - no solution 3 finished - solution (could add global cut if finished) -1 returned on size -2 time or user event.
- virtual void **generateCpp** (FILE \*)  
Create C++ lines to get to current state.
- void **generateCpp** (FILE \*fp, const char \*heuristic)  
Create C++ lines to get to current state - does work for base class.
- virtual bool **canDealWithOdd** () const  
Returns true if can deal with "odd" problems e.g. sos type 2.

- const char \* [heuristicName](#) () const  
*return name of heuristic*
- void [setHeuristicName](#) (const char \*name)  
*set name of heuristic*
- void [setSeed](#) (int value)  
*Set random number generator seed.*
- int [getSeed](#) () const  
*Get random number generator seed.*
- void [setDecayFactor](#) (double value)  
*Sets decay factor (for howOften) on failure.*
- void [setInputSolution](#) (const double \*[solution](#), double objValue)  
*Set input solution.*
- void [setWhereFrom](#) (int value)
- int [whereFrom](#) () const
- void [setShallowDepth](#) (int value)  
*Upto this depth we call the tree shallow and the heuristic can be called multiple times.*
- void [setHowOftenShallow](#) (int value)  
*How often to invoke the heuristics in the shallow part of the tree.*
- void [setMinDistanceToRun](#) (int value)  
*How "far" should this node be from every other where the heuristic was run in order to allow the heuristic to run in this node, too.*
- virtual bool [shouldHeurRun](#) (int [whereFrom](#))  
*Check whether the heuristic should run at all 0 - before cuts at root node (or from doHeuristics) 1 - during cuts at root 2 - after root node cuts 3 - after cuts at other nodes 4 - during cuts at other nodes 8 added if previous heuristic in loop found solution.*
- bool [shouldHeurRun\\_randomChoice](#) ()  
*Check whether the heuristic should run this time.*
- void [debugNodes](#) ()
- void [printDistanceToNodes](#) ()
- int [numRuns](#) () const  
*how many times the heuristic has actually run*
- int [numCouldRun](#) () const  
*How many times the heuristic could run.*
- OsiSolverInterface \* [cloneBut](#) (int type)  
*Clone, but ...*

#### Protected Attributes

- [CbcModel](#) \* [model\\_](#)  
*Model.*
- int [when\\_](#)  
*When flag - 0 off, 1 at root, 2 other than root, 3 always.*
- int [numberNodes\\_](#)  
*Number of nodes in any sub tree.*
- int [feasibilityPumpOptions\\_](#)  
*Feasibility pump options , -1 is off >=0 for feasibility pump itself -2 quick proximity search -3 longer proximity search.*
- double [fractionSmall\\_](#)  
*Fraction of new(rows+columns)/old(rows+columns) before doing small branch and bound.*

- CoinThreadRandom [randomNumberGenerator\\_](#)  
*Thread specific random number generator.*
- std::string [heuristicName\\_](#)  
*Name for printing.*
- int [howOften\\_](#)  
*How often to do (code can change)*
- double [decayFactor\\_](#)  
*How much to increase how often.*
- int [switches\\_](#)  
*Switches (does not apply equally to all heuristics) 1 bit - stop once allowable gap on objective reached 2 bit - always do given number of passes 4 bit - weaken cutoff by 5% every 50 passes? 8 bit - if has cutoff and suminf bobbling for 20 passes then first try halving distance to best possible then try keep halving distance to known cutoff 16 bit - needs new solution to run 1024 bit - stop all heuristics on max time.*
- int [whereFrom\\_](#)
- int [shallowDepth\\_](#)  
*Upto this depth we call the tree shallow and the heuristic can be called multiple times.*
- int [howOftenShallow\\_](#)  
*How often to invoke the heuristics in the shallow part of the tree.*
- int [numInvocationsInShallow\\_](#)  
*How many invocations happened within the same node when in a shallow part of the tree.*
- int [numInvocationsInDeep\\_](#)  
*How many invocations happened when in the deep part of the tree.*
- int [lastRunDeep\\_](#)  
*After how many deep invocations was the heuristic run last time.*
- int [numRuns\\_](#)  
*how many times the heuristic has actually run*
- int [minDistanceToRun\\_](#)  
*How "far" should this node be from every other where the heuristic was run in order to allow the heuristic to run in this node, too.*
- CbcHeuristicNodeList [runNodes\\_](#)  
*The description of the nodes where this heuristic has been applied.*
- int [numCouldRun\\_](#)  
*How many times the heuristic could run.*
- int [numberSolutionsFound\\_](#)  
*How many solutions the heuristic thought it got.*
- int [numberNodesDone\\_](#)  
*How many nodes the heuristic did this go.*
- double \* [inputSolution\\_](#)

#### 7.39.1 Detailed Description

Heuristic base class.

Definition at line 77 of file CbcHeuristic.hpp.

### 7.39.2 Constructor & Destructor Documentation

7.39.2.1 `CbcHeuristic::CbcHeuristic ( )`

7.39.2.2 `CbcHeuristic::CbcHeuristic ( CbcModel & model )`

7.39.2.3 `CbcHeuristic::CbcHeuristic ( const CbcHeuristic & )`

7.39.2.4 `virtual CbcHeuristic::~~CbcHeuristic ( ) [virtual]`

### 7.39.3 Member Function Documentation

7.39.3.1 `virtual CbcHeuristic* CbcHeuristic::clone ( ) const [pure virtual]`

Clone.

Implemented in [CbcHeuristicJustOne](#), [CbcSerendipity](#), [CbcHeuristicPartial](#), [CbcRounding](#), [CbcHeuristicDynamic3](#), [CbcHeuristicCrossover](#), [CbcHeuristicGreedySOS](#), [CbcHeuristicNaive](#), [CbcHeuristicGreedyEquality](#), [CbcHeuristicProximity](#), [CbcHeuristicDW](#), [CbcHeuristicDive](#), [CbcHeuristicRINS](#), [CbcHeuristicVND](#), [CbcHeuristicFPump](#), [CbcHeuristicRENS](#), [CbcHeuristicDINS](#), [CbcHeuristicGreedyCover](#), [CbcHeuristicLocal](#), [CbcHeuristicPivotAndFix](#), [CbcHeuristicRandRound](#), [CbcHeuristicDiveCoefficient](#), [CbcHeuristicDiveFractional](#), [CbcHeuristicDiveGuided](#), [CbcHeuristicDiveLineSearch](#), [CbcHeuristicDivePseudoCost](#), and [CbcHeuristicDiveVectorLength](#).

7.39.3.2 `CbcHeuristic& CbcHeuristic::operator= ( const CbcHeuristic & rhs )`

Assignment operator.

7.39.3.3 `virtual void CbcHeuristic::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented in [CbcHeuristicJustOne](#), [CbcSerendipity](#), [CbcHeuristicPartial](#), [CbcRounding](#), [CbcHeuristicDynamic3](#), [CbcHeuristicCrossover](#), [CbcHeuristicGreedySOS](#), [CbcHeuristicNaive](#), [CbcHeuristicGreedyEquality](#), [CbcHeuristicProximity](#), [CbcHeuristicGreedyCover](#), [CbcHeuristicDW](#), [CbcHeuristicDive](#), [CbcHeuristicRINS](#), [CbcHeuristicVND](#), [CbcHeuristicDINS](#), [CbcHeuristicRENS](#), [CbcHeuristicLocal](#), [CbcHeuristicPivotAndFix](#), [CbcHeuristicRandRound](#), [CbcHeuristicFPump](#), and [CbcHeuristicGreedyCover](#).

7.39.3.4 `virtual void CbcHeuristic::resetModel ( CbcModel * model ) [pure virtual]`

Resets stuff if model changes.

Implemented in [CbcHeuristicJustOne](#), [CbcSerendipity](#), [CbcHeuristicPartial](#), [CbcRounding](#), [CbcHeuristicDynamic3](#), [CbcHeuristicCrossover](#), [CbcHeuristicGreedySOS](#), [CbcHeuristicNaive](#), [CbcHeuristicGreedyEquality](#), [CbcHeuristicProximity](#), [CbcHeuristicGreedyCover](#), [CbcHeuristicDW](#), [CbcHeuristicDive](#), [CbcHeuristicRINS](#), [CbcHeuristicVND](#), [CbcHeuristicDINS](#), [CbcHeuristicRENS](#), [CbcHeuristicLocal](#), [CbcHeuristicPivotAndFix](#), [CbcHeuristicRandRound](#), and [CbcHeuristicFPump](#).

7.39.3.5 `virtual int CbcHeuristic::solution ( double & objectiveValue, double * newSolution ) [pure virtual]`

returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value This is called after cuts have been added - so can not add cuts

Implemented in [CbcHeuristicJustOne](#), [CbcSerendipity](#), [CbcHeuristicPartial](#), [CbcRounding](#), [CbcHeuristicDynamic3](#), [CbcHeuristicCrossover](#), [CbcHeuristicGreedySOS](#), [CbcHeuristicNaive](#), [CbcHeuristicGreedyEquality](#), [CbcHeuristicProximity](#), [CbcHeuristicFPump](#), [CbcHeuristicDive](#), [CbcHeuristicDW](#), [CbcHeuristicLocal](#), [CbcHeuristicRINS](#), [CbcHeuristicVND](#), [CbcHeuristicDINS](#), [CbcHeuristicRENS](#), [CbcHeuristicGreedyCover](#), [CbcHeuristicPivotAndFix](#), and [CbcHeuristicRandRound](#).



**7.39.3.6** `virtual int CbcHeuristic::solution2 ( double & , double * , OsiCuts & ) [inline],[virtual]`

returns 0 if no solution, 1 if valid solution, -1 if just returning an estimate of best possible solution with better objective value than one passed in Sets solution values if good, sets objective value (only if nonzero code) This is called at same time as cut generators - so can add cuts Default is do nothing

Definition at line 121 of file CbcHeuristic.hpp.

**7.39.3.7** `virtual void CbcHeuristic::validate ( ) [inline],[virtual]`

Validate model i.e. sets when\_ to 0 if necessary (may be NULL)

Reimplemented in [CbcHeuristicJustOne](#), [CbcHeuristicPartial](#), [CbcRounding](#), [CbcHeuristicGreedySOS](#), [CbcHeuristic-GreedyEquality](#), [CbcHeuristicDive](#), and [CbcHeuristicGreedyCover](#).

Definition at line 128 of file CbcHeuristic.hpp.

**7.39.3.8** `void CbcHeuristic::setWhen ( int value ) [inline]`

Sets "when" flag - 0 off, 1 at root, 2 other than root, 3 always.

If 10 added then don't worry if validate says there are funny objects as user knows it will be fine

Definition at line 134 of file CbcHeuristic.hpp.

**7.39.3.9** `int CbcHeuristic::when ( ) const [inline]`

Gets "when" flag - 0 off, 1 at root, 2 other than root, 3 always.

Definition at line 138 of file CbcHeuristic.hpp.

**7.39.3.10** `void CbcHeuristic::setNumberNodes ( int value ) [inline]`

Sets number of nodes in subtree (default 200)

Definition at line 143 of file CbcHeuristic.hpp.

**7.39.3.11** `int CbcHeuristic::numberNodes ( ) const [inline]`

Gets number of nodes in a subtree (default 200)

Definition at line 147 of file CbcHeuristic.hpp.

**7.39.3.12** `void CbcHeuristic::setSwitches ( int value ) [inline]`

Switches (does not apply equally to all heuristics) 1 bit - stop once allowable gap on objective reached 2 bit - always do given number of passes 4 bit - weaken cutoff by 5% every 50 passes? 8 bit - if has cutoff and suminf bobbling for 20 passes then first try halving distance to best possible then try keep halving distance to known cutoff 16 bit - needs new solution to run 1024 bit - stop all heuristics on max time.

Definition at line 160 of file CbcHeuristic.hpp.

**7.39.3.13** `int CbcHeuristic::switches ( ) const [inline]`

Switches (does not apply equally to all heuristics) 1 bit - stop once allowable gap on objective reached 2 bit - always do given number of passes 4 bit - weaken cutoff by 5% every 50 passes? 8 bit - if has cutoff and suminf bobbling for 20 passes then first try halving distance to best possible then try keep halving distance to known cutoff 16 bit - needs new solution to run 1024 bit - stop all heuristics on max time.

Definition at line 173 of file CbcHeuristic.hpp.

**7.39.3.14** `bool CbcHeuristic::exitNow ( double bestObjective ) const`

Whether to exit at once on gap.

**7.39.3.15** `void CbcHeuristic::setFeasibilityPumpOptions ( int value ) [inline]`

Sets feasibility pump options (-1 is off)

Definition at line 179 of file CbcHeuristic.hpp.

**7.39.3.16** `int CbcHeuristic::feasibilityPumpOptions ( ) const [inline]`

Gets feasibility pump options (-1 is off)

Definition at line 183 of file CbcHeuristic.hpp.

**7.39.3.17** `void CbcHeuristic::setModelOnly ( CbcModel * model ) [inline]`

Just set model - do not do anything else.

Definition at line 187 of file CbcHeuristic.hpp.

**7.39.3.18** `void CbcHeuristic::setFractionSmall ( double value ) [inline]`

Sets fraction of new(rows+columns)/old(rows+columns) before doing small branch and bound (default 1.0)

Definition at line 193 of file CbcHeuristic.hpp.

**7.39.3.19** `double CbcHeuristic::fractionSmall ( ) const [inline]`

Gets fraction of new(rows+columns)/old(rows+columns) before doing small branch and bound (default 1.0)

Definition at line 197 of file CbcHeuristic.hpp.

**7.39.3.20** `int CbcHeuristic::numberSolutionsFound ( ) const [inline]`

Get how many solutions the heuristic thought it got.

Definition at line 201 of file CbcHeuristic.hpp.

**7.39.3.21** `void CbcHeuristic::incrementNumberSolutionsFound ( ) [inline]`

Increment how many solutions the heuristic thought it got.

Definition at line 205 of file CbcHeuristic.hpp.

**7.39.3.22** `int CbcHeuristic::smallBranchAndBound ( OsiSolverInterface * solver, int numberNodes, double * newSolution, double & newSolutionValue, double cutoff, std::string name ) const`

Do mini branch and bound - return 0 not finished - no solution 1 not finished - solution 2 finished - no solution 3 finished - solution (could add global cut if finished) -1 returned on size -2 time or user event.

**7.39.3.23** `virtual void CbcHeuristic::generateCpp ( FILE * ) [inline],[virtual]`

Create C++ lines to get to current state.

Reimplemented in [CbcHeuristicJustOne](#), [CbcSerendipity](#), [CbcHeuristicPartial](#), [CbcRounding](#), [CbcHeuristicCrossover](#), [CbcHeuristicGreedySOS](#), [CbcHeuristicNaive](#), [CbcHeuristicGreedyEquality](#), [CbcHeuristicProximity](#), [CbcHeuristicDW](#), [CbcHeuristicDive](#), [CbcHeuristicRINS](#), [CbcHeuristicVND](#), [CbcHeuristicDINS](#), [CbcHeuristicLocal](#), [CbcHeuristicPivotAndFix](#), [CbcHeuristicRandRound](#), [CbcHeuristicDiveCoefficient](#), [CbcHeuristicDiveFractional](#), [CbcHeuristicDiveGuided](#), [CbcHeuristicDiveLineSearch](#), [CbcHeuristicDivePseudoCost](#), [CbcHeuristicDiveVectorLength](#), [CbcHeuristicFPump](#), and

[CbcHeuristicGreedyCover](#).

Definition at line 222 of file CbcHeuristic.hpp.

**7.39.3.24** void CbcHeuristic::generateCpp ( FILE \* *fp*, const char \* *heuristic* )

Create C++ lines to get to current state - does work for base class.

**7.39.3.25** virtual bool CbcHeuristic::canDealWithOdd ( ) const [inline], [virtual]

Returns true if can deal with "odd" problems e.g. sos type 2.

Reimplemented in [CbcHeuristicDynamic3](#).

Definition at line 226 of file CbcHeuristic.hpp.

**7.39.3.26** const char\* CbcHeuristic::heuristicName ( ) const [inline]

return name of heuristic

Definition at line 230 of file CbcHeuristic.hpp.

**7.39.3.27** void CbcHeuristic::setHeuristicName ( const char \* *name* ) [inline]

set name of heuristic

Definition at line 234 of file CbcHeuristic.hpp.

**7.39.3.28** void CbcHeuristic::setSeed ( int *value* )

Set random number generator seed.

**7.39.3.29** int CbcHeuristic::getSeed ( ) const

Get random number generator seed.

**7.39.3.30** void CbcHeuristic::setDecayFactor ( double *value* ) [inline]

Sets decay factor (for howOften) on failure.

Definition at line 242 of file CbcHeuristic.hpp.

**7.39.3.31** void CbcHeuristic::setInputSolution ( const double \* *solution*, double *objValue* )

Set input solution.

**7.39.3.32** void CbcHeuristic::setWhereFrom ( int *value* ) [inline]

Definition at line 255 of file CbcHeuristic.hpp.

**7.39.3.33** int CbcHeuristic::whereFrom ( ) const [inline]

Definition at line 258 of file CbcHeuristic.hpp.

**7.39.3.34** void CbcHeuristic::setShallowDepth ( int *value* ) [inline]

Upto this depth we call the tree shallow and the heuristic can be called multiple times.

That is, the test whether the current node is far from the others where the heuristic was invoked will not be done, only the frequency will be tested. After that depth the heuristic will can be invoked only once per node, right before branching. That's when it'll be tested whether the heuristic should run at all.

Definition at line 267 of file CbcHeuristic.hpp.

**7.39.3.35** `void CbcHeuristic::setHowOftenShallow ( int value ) [inline]`

How often to invoke the heuristics in the shallow part of the tree.

Definition at line 271 of file CbcHeuristic.hpp.

**7.39.3.36** `void CbcHeuristic::setMinDistanceToRun ( int value ) [inline]`

How "far" should this node be from every other where the heuristic was run in order to allow the heuristic to run in this node, too.

Currently this is tested, but we may switch to `avgDistanceToRun_` in the future.

Definition at line 277 of file CbcHeuristic.hpp.

**7.39.3.37** `virtual bool CbcHeuristic::shouldHeurRun ( int whereFrom ) [virtual]`

Check whether the heuristic should run at all 0 - before cuts at root node (or from `doHeuristics`) 1 - during cuts at root 2 - after root node cuts 3 - after cuts at other nodes 4 - during cuts at other nodes 8 added if previous heuristic in loop found solution.

Reimplemented in [CbcHeuristicPartial](#).

**7.39.3.38** `bool CbcHeuristic::shouldHeurRun_randomChoice ( )`

Check whether the heuristic should run this time.

**7.39.3.39** `void CbcHeuristic::debugNodes ( )`

**7.39.3.40** `void CbcHeuristic::printDistanceToNodes ( )`

**7.39.3.41** `int CbcHeuristic::numRuns ( ) const [inline]`

how many times the heuristic has actually run

Definition at line 295 of file CbcHeuristic.hpp.

**7.39.3.42** `int CbcHeuristic::numCouldRun ( ) const [inline]`

How many times the heuristic could run.

Definition at line 300 of file CbcHeuristic.hpp.

**7.39.3.43** `OsiSolverInterface* CbcHeuristic::cloneBut ( int type )`

Clone, but ...

If type is

- 0 clone the solver for the model,
- 1 clone the continuous solver for the model
- Add 2 to say without integer variables which are at low priority
- Add 4 to say quite likely infeasible so give up easily (clp only).

#### 7.39.4 Member Data Documentation

**7.39.4.1 CbcModel\* CbcHeuristic::model\_ [protected]**

Model.

Definition at line 315 of file CbcHeuristic.hpp.

**7.39.4.2 int CbcHeuristic::when\_ [protected]**

When flag - 0 off, 1 at root, 2 other than root, 3 always.

Definition at line 317 of file CbcHeuristic.hpp.

**7.39.4.3 int CbcHeuristic::numberNodes\_ [protected]**

Number of nodes in any sub tree.

Definition at line 319 of file CbcHeuristic.hpp.

**7.39.4.4 int CbcHeuristic::feasibilityPumpOptions\_ [protected]**

Feasibility pump options , -1 is off  $\geq 0$  for feasibility pump itself -2 quick proximity search -3 longer proximity search.

Definition at line 325 of file CbcHeuristic.hpp.

**7.39.4.5 double CbcHeuristic::fractionSmall\_ [mutable], [protected]**

Fraction of new(rows+columns)/old(rows+columns) before doing small branch and bound.

Definition at line 327 of file CbcHeuristic.hpp.

**7.39.4.6 CoinThreadRandom CbcHeuristic::randomNumberGenerator\_ [protected]**

Thread specific random number generator.

Definition at line 329 of file CbcHeuristic.hpp.

**7.39.4.7 std::string CbcHeuristic::heuristicName\_ [protected]**

Name for printing.

Definition at line 331 of file CbcHeuristic.hpp.

**7.39.4.8 int CbcHeuristic::howOften\_ [mutable], [protected]**

How often to do (code can change)

Definition at line 334 of file CbcHeuristic.hpp.

**7.39.4.9 double CbcHeuristic::decayFactor\_ [protected]**

How much to increase how often.

Definition at line 336 of file CbcHeuristic.hpp.

**7.39.4.10 int CbcHeuristic::switches\_ [mutable], [protected]**

Switches (does not apply equally to all heuristics) 1 bit - stop once allowable gap on objective reached 2 bit - always do given number of passes 4 bit - weaken cutoff by 5% every 50 passes? 8 bit - if has cutoff and suminf bobbling for 20 passes then first try halving distance to best possible then try keep halving distance to known cutoff 16 bit - needs new solution to run 1024 bit - stop all heuristics on max time.

Definition at line 347 of file CbcHeuristic.hpp.

#### 7.39.4.11 `int CbcHeuristic::whereFrom_` [protected]

Definition at line 356 of file CbcHeuristic.hpp.

#### 7.39.4.12 `int CbcHeuristic::shallowDepth_` [protected]

Upto this depth we call the tree shallow and the heuristic can be called multiple times.

That is, the test whether the current node is far from the others where the heuristic was invoked will not be done, only the frequency will be tested. After that depth the heuristic will can be invoked only once per node, right before branching. That's when it'll be tested whether the heuristic should run at all.

Definition at line 363 of file CbcHeuristic.hpp.

#### 7.39.4.13 `int CbcHeuristic::howOftenShallow_` [protected]

How often to invoke the heuristics in the shallow part of the tree.

Definition at line 365 of file CbcHeuristic.hpp.

#### 7.39.4.14 `int CbcHeuristic::numInvocationsInShallow_` [protected]

How many invocations happened within the same node when in a shallow part of the tree.

Definition at line 368 of file CbcHeuristic.hpp.

#### 7.39.4.15 `int CbcHeuristic::numInvocationsInDeep_` [protected]

How many invocations happened when in the deep part of the tree.

For every node we count only one invocation.

Definition at line 371 of file CbcHeuristic.hpp.

#### 7.39.4.16 `int CbcHeuristic::lastRunDeep_` [protected]

After how many deep invocations was the heuristic run last time.

Definition at line 373 of file CbcHeuristic.hpp.

#### 7.39.4.17 `int CbcHeuristic::numRuns_` [protected]

how many times the heuristic has actually run

Definition at line 375 of file CbcHeuristic.hpp.

#### 7.39.4.18 `int CbcHeuristic::minDistanceToRun_` [protected]

How "far" should this node be from every other where the heuristic was run in order to allow the heuristic to run in this node, too.

Currently this is tested, but we may switch to `avgDistanceToRun_` in the future.

Definition at line 379 of file CbcHeuristic.hpp.

#### 7.39.4.19 `CbcHeuristicNodeList CbcHeuristic::runNodes_` [protected]

The description of the nodes where this heuristic has been applied.

Definition at line 382 of file CbcHeuristic.hpp.

7.39.4.20 `int CbcHeuristic::numCouldRun_` [protected]

How many times the heuristic could run.

Definition at line 385 of file CbcHeuristic.hpp.

7.39.4.21 `int CbcHeuristic::numberSolutionsFound_` [protected]

How many solutions the heuristic thought it got.

Definition at line 388 of file CbcHeuristic.hpp.

7.39.4.22 `int CbcHeuristic::numberNodesDone_` [mutable], [protected]

How many nodes the heuristic did this go.

Definition at line 391 of file CbcHeuristic.hpp.

7.39.4.23 `double* CbcHeuristic::inputSolution_` [protected]

Definition at line 394 of file CbcHeuristic.hpp.

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

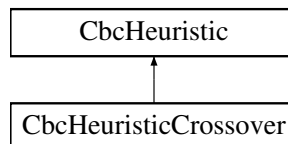
- </home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp>

## 7.40 CbcHeuristicCrossover Class Reference

Crossover Search class.

```
#include <CbcHeuristicLocal.hpp>
```

Inheritance diagram for CbcHeuristicCrossover:



### Public Member Functions

- [CbcHeuristicCrossover](#) ()
- [CbcHeuristicCrossover](#) ([CbcModel](#) &model)
- [CbcHeuristicCrossover](#) (const [CbcHeuristicCrossover](#) &)
- [~CbcHeuristicCrossover](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicCrossover](#) & [operator=](#) (const [CbcHeuristicCrossover](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*

- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- void [setNumberSolutions](#) (int value)  
*Sets number of solutions to use.*

#### Protected Attributes

- std::vector< double > [attempts\\_](#)  
*Attempts.*
- double [random\\_](#) [10]  
*Random numbers to stop same search happening.*
- int [numberSolutions\\_](#)  
*Number of solutions so we only do after new solution.*
- int [useNumber\\_](#)  
*Number of solutions to use.*

#### 7.40.1 Detailed Description

Crossover Search class.

Definition at line 211 of file CbcHeuristicLocal.hpp.

#### 7.40.2 Constructor & Destructor Documentation

7.40.2.1 [CbcHeuristicCrossover::CbcHeuristicCrossover \( \)](#)

7.40.2.2 [CbcHeuristicCrossover::CbcHeuristicCrossover \( \[CbcModel\]\(#\) & model \)](#)

7.40.2.3 [CbcHeuristicCrossover::CbcHeuristicCrossover \( const \[CbcHeuristicCrossover\]\(#\) & \)](#)

7.40.2.4 [CbcHeuristicCrossover::~~CbcHeuristicCrossover \( \)](#)

#### 7.40.3 Member Function Documentation

7.40.3.1 [virtual \[CbcHeuristic\]\(#\)\\* \[CbcHeuristicCrossover::clone\]\(#\) \( \) const](#) [virtual]

Clone.

Implements [CbcHeuristic](#).

7.40.3.2 [CbcHeuristicCrossover& \[CbcHeuristicCrossover::operator=\]\(#\) \( const \[CbcHeuristicCrossover\]\(#\) & rhs \)](#)

Assignment operator.

7.40.3.3 [virtual void \[CbcHeuristicCrossover::generateCpp\]\(#\) \( FILE \\* fp \)](#) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).



7.40.3.4 `virtual void CbcHeuristicCrossover::resetModel ( CbcModel * model ) [virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.40.3.5 `virtual void CbcHeuristicCrossover::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.40.3.6 `virtual int CbcHeuristicCrossover::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution.

Fix variables if agree in useNumber\_ solutions when\_ 0 off, 1 only at new solutions, 2 also every now and then add 10 to make only if agree at lower bound

Implements [CbcHeuristic](#).

7.40.3.7 `void CbcHeuristicCrossover::setNumberSolutions ( int value ) [inline]`

Sets number of solutions to use.

Definition at line 253 of file CbcHeuristicLocal.hpp.

#### 7.40.4 Member Data Documentation

7.40.4.1 `std::vector<double> CbcHeuristicCrossover::attempts_ [protected]`

Attempts.

Definition at line 261 of file CbcHeuristicLocal.hpp.

7.40.4.2 `double CbcHeuristicCrossover::random_[10] [protected]`

Random numbers to stop same search happening.

Definition at line 263 of file CbcHeuristicLocal.hpp.

7.40.4.3 `int CbcHeuristicCrossover::numberSolutions_ [protected]`

Number of solutions so we only do after new solution.

Definition at line 265 of file CbcHeuristicLocal.hpp.

7.40.4.4 `int CbcHeuristicCrossover::useNumber_ [protected]`

Number of solutions to use.

Definition at line 267 of file CbcHeuristicLocal.hpp.

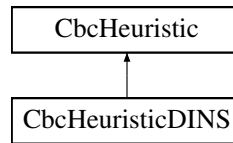
The documentation for this class was generated from the following file:

- `/home/ted/COIN/trunk/Cbc/src/CbcHeuristicLocal.hpp`

## 7.41 CbcHeuristicDINS Class Reference

```
#include <CbcHeuristicDINS.hpp>
```

Inheritance diagram for CbcHeuristicDINS:



### Public Member Functions

- [CbcHeuristicDINS](#) ()
- [CbcHeuristicDINS](#) ([CbcModel](#) &model)
- [CbcHeuristicDINS](#) (const [CbcHeuristicDINS](#) &)
- [~CbcHeuristicDINS](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicDINS](#) & [operator=](#) (const [CbcHeuristicDINS](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- int [solutionFix](#) (double &objectiveValue, double \*newSolution, const int \*keep)  
*This version fixes stuff and does IP.*
- void [setHowOften](#) (int value)  
*Sets how often to do it.*
- void [setMaximumKeep](#) (int value)  
*Sets maximum number of solutions kept.*
- void [setConstraint](#) (int value)  
*Sets tightness of extra constraint.*

### Protected Attributes

- int [numberSolutions\\_](#)  
*Number of solutions so we can do something at solution.*
- int [howOften\\_](#)  
*How often to do (code can change)*
- int [numberSuccesses\\_](#)  
*Number of successes.*
- int [numberTries\\_](#)  
*Number of tries.*
- int [maximumKeepSolutions\\_](#)  
*Maximum number of solutions to keep.*

- int [numberKeptSolutions\\_](#)  
*Number of solutions kept.*
- int [numberIntegers\\_](#)  
*Number of integer variables.*
- int [localSpace\\_](#)  
*Local parameter.*
- int \*\* [values\\_](#)  
*Values of integer variables.*

#### 7.41.1 Detailed Description

Definition at line 14 of file CbcHeuristicDINS.hpp.

#### 7.41.2 Constructor & Destructor Documentation

7.41.2.1 `CbcHeuristicDINS::CbcHeuristicDINS ( )`

7.41.2.2 `CbcHeuristicDINS::CbcHeuristicDINS ( CbcModel & model )`

7.41.2.3 `CbcHeuristicDINS::CbcHeuristicDINS ( const CbcHeuristicDINS & )`

7.41.2.4 `CbcHeuristicDINS::~CbcHeuristicDINS ( )`

#### 7.41.3 Member Function Documentation

7.41.3.1 `virtual CbcHeuristic* CbcHeuristicDINS::clone ( ) const` [virtual]

Clone.

Implements [CbcHeuristic](#).

7.41.3.2 `CbcHeuristicDINS& CbcHeuristicDINS::operator= ( const CbcHeuristicDINS & rhs )`

Assignment operator.

7.41.3.3 `virtual void CbcHeuristicDINS::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.41.3.4 `virtual void CbcHeuristicDINS::resetModel ( CbcModel * model )` [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.41.3.5 `virtual void CbcHeuristicDINS::setModel ( CbcModel * model )` [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.41.3.6 `virtual int CbcHeuristicDINS::solution ( double & objectiveValue, double * newSolution )` [virtual]

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) This does Relaxation Induced Neighborhood Search  
Implements [CbcHeuristic](#).

**7.41.3.7** `int CbcHeuristicDINS::solutionFix ( double & objectiveValue, double * newSolution, const int * keep )`

This version fixes stuff and does IP.

**7.41.3.8** `void CbcHeuristicDINS::setHowOften ( int value ) [inline]`

Sets how often to do it.

Definition at line 60 of file CbcHeuristicDINS.hpp.

**7.41.3.9** `void CbcHeuristicDINS::setMaximumKeep ( int value ) [inline]`

Sets maximum number of solutions kept.

Definition at line 64 of file CbcHeuristicDINS.hpp.

**7.41.3.10** `void CbcHeuristicDINS::setConstraint ( int value ) [inline]`

Sets tightness of extra constraint.

Definition at line 68 of file CbcHeuristicDINS.hpp.

#### 7.41.4 Member Data Documentation

**7.41.4.1** `int CbcHeuristicDINS::numberSolutions_ [protected]`

Number of solutions so we can do something at solution.

Definition at line 76 of file CbcHeuristicDINS.hpp.

**7.41.4.2** `int CbcHeuristicDINS::howOften_ [protected]`

How often to do (code can change)

Definition at line 78 of file CbcHeuristicDINS.hpp.

**7.41.4.3** `int CbcHeuristicDINS::numberSuccesses_ [protected]`

Number of successes.

Definition at line 80 of file CbcHeuristicDINS.hpp.

**7.41.4.4** `int CbcHeuristicDINS::numberTries_ [protected]`

Number of tries.

Definition at line 82 of file CbcHeuristicDINS.hpp.

**7.41.4.5** `int CbcHeuristicDINS::maximumKeepSolutions_ [protected]`

Maximum number of solutions to keep.

Definition at line 84 of file CbcHeuristicDINS.hpp.

**7.41.4.6** `int CbcHeuristicDINS::numberKeptSolutions_ [protected]`

Number of solutions kept.

Definition at line 86 of file CbcHeuristicDINS.hpp.

7.41.4.7 `int CbcHeuristicDINS::numberIntegers_ [protected]`

Number of integer variables.

Definition at line 88 of file CbcHeuristicDINS.hpp.

7.41.4.8 `int CbcHeuristicDINS::localSpace_ [protected]`

Local parameter.

Definition at line 90 of file CbcHeuristicDINS.hpp.

7.41.4.9 `int** CbcHeuristicDINS::values_ [protected]`

Values of integer variables.

Definition at line 92 of file CbcHeuristicDINS.hpp.

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

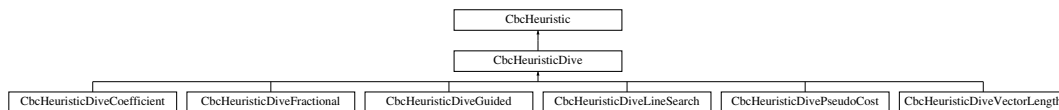
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDINS.hpp](#)

## 7.42 CbcHeuristicDive Class Reference

Dive class.

```
#include <CbcHeuristicDive.hpp>
```

Inheritance diagram for CbcHeuristicDive:



### Public Member Functions

- [CbcHeuristicDive \(\)](#)
- [CbcHeuristicDive \(CbcModel &model\)](#)
- [CbcHeuristicDive \(const CbcHeuristicDive &\)](#)
- [~CbcHeuristicDive \(\)](#)
- virtual [CbcHeuristicDive \\* clone \(\)](#) const =0  
*Clone.*
- [CbcHeuristicDive & operator= \(const CbcHeuristicDive &rhs\)](#)  
*Assignment operator.*
- virtual void [generateCpp \(FILE \\*\)](#)  
*Create C++ lines to get to current state.*
- void [generateCpp \(FILE \\*fp, const char \\*heuristic\)](#)  
*Create C++ lines to get to current state - does work for base class.*
- virtual void [resetModel \(CbcModel \\*model\)](#)  
*Resets stuff if model changes.*
- virtual void [setModel \(CbcModel \\*model\)](#)

- update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)
  - returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts This does Fractional Diving*
- int [solution](#) (double &objectiveValue, int &numberNodes, int &numberCuts, OsiRowCut \*\*cuts, CbcSubProblem \*\*&nodes, double \*newSolution)
  - inner part of dive*
- int [fathom](#) (CbcModel \*model, int &numberNodes, CbcSubProblem \*\*&nodes)
  - returns 0 if no solution, 1 if valid solution with better objective value than one passed in also returns list of nodes This does Fractional Diving*
- virtual void [validate](#) ()
  - Validate model i.e. sets when\_ to 0 if necessary (may be NULL)*
- void [selectBinaryVariables](#) ()
  - Select candidate binary variables for fixing.*
- void [setPercentageToFix](#) (double value)
  - Set percentage of integer variables to fix at bounds.*
- void [setMaxIterations](#) (int value)
  - Set maximum number of iterations.*
- void [setMaxSimplexIterations](#) (int value)
  - Set maximum number of simplex iterations.*
- int [maxSimplexIterations](#) () const
  - Get maximum number of simplex iterations.*
- void [setMaxSimplexIterationsAtRoot](#) (int value)
  - Set maximum number of simplex iterations at root node.*
- void [setMaxTime](#) (double value)
  - Set maximum time allowed.*
- virtual bool [canHeuristicRun](#) ()
  - Tests if the heuristic can run.*
- virtual bool [selectVariableToBranch](#) (OsiSolverInterface \*solver, const double \*newSolution, int &bestColumn, int &bestRound)=0
  - Selects the next variable to branch on Returns true if all the fractional variables can be trivially rounded.*
- virtual void [initializeData](#) ()
  - Initializes any data which is going to be used repeatedly in selectVariableToBranch.*
- int [reducedCostFix](#) (OsiSolverInterface \*solver)
  - Perform reduced cost fixing on integer variables.*
- virtual int [fixOtherVariables](#) (OsiSolverInterface \*solver, const double \*solution, [PseudoReducedCost](#) \*candidate, const double \*random)
  - Fix other variables at bounds.*

#### Protected Attributes

- CoinPackedMatrix [matrix\\_](#)
- CoinPackedMatrix [matrixByRow\\_](#)
- unsigned short \* [downLocks\\_](#)
- unsigned short \* [upLocks\\_](#)
- double \* [downArray\\_](#)
  - Extra down array (number Integers long)*
- double \* [upArray\\_](#)

*Extra up array (number Integers long)*

- `std::vector< int > binVarIndex_`
- `std::vector< int > vbRowIndex_`
- `double percentageToFix_`
- `int maxIterations_`
- `int maxSimplexIterations_`
- `int maxSimplexIterationsAtRoot_`
- `double maxTime_`

### 7.42.1 Detailed Description

Dive class.

Definition at line 21 of file CbcHeuristicDive.hpp.

### 7.42.2 Constructor & Destructor Documentation

7.42.2.1 `CbcHeuristicDive::CbcHeuristicDive ( )`

7.42.2.2 `CbcHeuristicDive::CbcHeuristicDive ( CbcModel & model )`

7.42.2.3 `CbcHeuristicDive::CbcHeuristicDive ( const CbcHeuristicDive & )`

7.42.2.4 `CbcHeuristicDive::~~CbcHeuristicDive ( )`

### 7.42.3 Member Function Documentation

7.42.3.1 `virtual CbcHeuristicDive* CbcHeuristicDive::clone ( ) const` [pure virtual]

Clone.

Implements [CbcHeuristic](#).

Implemented in [CbcHeuristicDiveCoefficient](#), [CbcHeuristicDiveFractional](#), [CbcHeuristicDiveGuided](#), [CbcHeuristicDiveLineSearch](#), [CbcHeuristicDivePseudoCost](#), and [CbcHeuristicDiveVectorLength](#).

7.42.3.2 `CbcHeuristicDive& CbcHeuristicDive::operator= ( const CbcHeuristicDive & rhs )`

Assignment operator.

7.42.3.3 `virtual void CbcHeuristicDive::generateCpp ( FILE * )` [inline],[virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

Reimplemented in [CbcHeuristicDiveCoefficient](#), [CbcHeuristicDiveFractional](#), [CbcHeuristicDiveGuided](#), [CbcHeuristicDiveLineSearch](#), [CbcHeuristicDivePseudoCost](#), and [CbcHeuristicDiveVectorLength](#).

Definition at line 43 of file CbcHeuristicDive.hpp.

7.42.3.4 `void CbcHeuristicDive::generateCpp ( FILE * fp, const char * heuristic )`

Create C++ lines to get to current state - does work for base class.

7.42.3.5 `virtual void CbcHeuristicDive::resetModel ( CbcModel * model ) [virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.42.3.6 `virtual void CbcHeuristicDive::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.42.3.7 `virtual int CbcHeuristicDive::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts This does Fractional Diving

Implements [CbcHeuristic](#).

7.42.3.8 `int CbcHeuristicDive::solution ( double & objectiveValue, int & numberNodes, int & numberCuts, OsiRowCut ** cuts, CbcSubProblem **& nodes, double * newSolution )`

inner part of dive

7.42.3.9 `int CbcHeuristicDive::fathom ( CbcModel * model, int & numberNodes, CbcSubProblem **& nodes )`

returns 0 if no solution, 1 if valid solution with better objective value than one passed in also returns list of nodes This does Fractional Diving

7.42.3.10 `virtual void CbcHeuristicDive::validate ( ) [virtual]`

Validate model i.e. sets when\_ to 0 if necessary (may be NULL)

Reimplemented from [CbcHeuristic](#).

7.42.3.11 `void CbcHeuristicDive::selectBinaryVariables ( )`

Select candidate binary variables for fixing.

7.42.3.12 `void CbcHeuristicDive::setPercentageToFix ( double value ) [inline]`

Set percentage of integer variables to fix at bounds.

Definition at line 82 of file CbcHeuristicDive.hpp.

7.42.3.13 `void CbcHeuristicDive::setMaxIterations ( int value ) [inline]`

Set maximum number of iterations.

Definition at line 87 of file CbcHeuristicDive.hpp.

7.42.3.14 `void CbcHeuristicDive::setMaxSimplexIterations ( int value ) [inline]`

Set maximum number of simplex iterations.

Definition at line 92 of file CbcHeuristicDive.hpp.

7.42.3.15 `int CbcHeuristicDive::maxSimplexIterations ( ) const [inline]`

Get maximum number of simplex iterations.



Definition at line 96 of file CbcHeuristicDive.hpp.

**7.42.3.16** `void CbcHeuristicDive::setMaxSimplexIterationsAtRoot ( int value )` `[inline]`

Set maximum number of simplex iterations at root node.

Definition at line 101 of file CbcHeuristicDive.hpp.

**7.42.3.17** `void CbcHeuristicDive::setMaxTime ( double value )` `[inline]`

Set maximum time allowed.

Definition at line 106 of file CbcHeuristicDive.hpp.

**7.42.3.18** `virtual bool CbcHeuristicDive::canHeuristicRun ( )` `[virtual]`

Tests if the heuristic can run.

Reimplemented in [CbcHeuristicDiveGuided](#).

**7.42.3.19** `virtual bool CbcHeuristicDive::selectVariableToBranch ( OsiSolverInterface * solver, const double * newSolution, int & bestColumn, int & bestRound )` `[pure virtual]`

Selects the next variable to branch on Returns true if all the fractional variables can be trivially rounded.

Returns false, if there is at least one fractional variable that is not trivially roundable. In this case, the bestColumn returned will not be trivially roundable.

Implemented in [CbcHeuristicDiveGuided](#), [CbcHeuristicDiveCoefficient](#), [CbcHeuristicDiveFractional](#), [CbcHeuristicDiveLineSearch](#), [CbcHeuristicDivePseudoCost](#), and [CbcHeuristicDiveVectorLength](#).

**7.42.3.20** `virtual void CbcHeuristicDive::initializeData ( )` `[inline], [virtual]`

Initializes any data which is going to be used repeatedly in selectVariableToBranch.

Reimplemented in [CbcHeuristicDivePseudoCost](#).

Definition at line 125 of file CbcHeuristicDive.hpp.

**7.42.3.21** `int CbcHeuristicDive::reducedCostFix ( OsiSolverInterface * solver )`

Perform reduced cost fixing on integer variables.

**7.42.3.22** `virtual int CbcHeuristicDive::fixOtherVariables ( OsiSolverInterface * solver, const double * solution, PseudoReducedCost * candidate, const double * random )` `[virtual]`

Fix other variables at bounds.

Reimplemented in [CbcHeuristicDivePseudoCost](#).

## 7.42.4 Member Data Documentation

**7.42.4.1** `CoinPackedMatrix CbcHeuristicDive::matrix_` `[protected]`

Definition at line 139 of file CbcHeuristicDive.hpp.

**7.42.4.2** `CoinPackedMatrix CbcHeuristicDive::matrixByRow_` `[protected]`

Definition at line 142 of file CbcHeuristicDive.hpp.

**7.42.4.3** `unsigned short* CbcHeuristicDive::downLocks_` [protected]

Definition at line 145 of file CbcHeuristicDive.hpp.

**7.42.4.4** `unsigned short* CbcHeuristicDive::upLocks_` [protected]

Definition at line 148 of file CbcHeuristicDive.hpp.

**7.42.4.5** `double* CbcHeuristicDive::downArray_` [protected]

Extra down array (number Integers long)

Definition at line 151 of file CbcHeuristicDive.hpp.

**7.42.4.6** `double* CbcHeuristicDive::upArray_` [protected]

Extra up array (number Integers long)

Definition at line 154 of file CbcHeuristicDive.hpp.

**7.42.4.7** `std::vector<int> CbcHeuristicDive::binVarIndex_` [protected]

Definition at line 158 of file CbcHeuristicDive.hpp.

**7.42.4.8** `std::vector<int> CbcHeuristicDive::vbRowIndex_` [protected]

Definition at line 161 of file CbcHeuristicDive.hpp.

**7.42.4.9** `double CbcHeuristicDive::percentageToFix_` [protected]

Definition at line 164 of file CbcHeuristicDive.hpp.

**7.42.4.10** `int CbcHeuristicDive::maxIterations_` [protected]

Definition at line 167 of file CbcHeuristicDive.hpp.

**7.42.4.11** `int CbcHeuristicDive::maxSimplexIterations_` [protected]

Definition at line 170 of file CbcHeuristicDive.hpp.

**7.42.4.12** `int CbcHeuristicDive::maxSimplexIterationsAtRoot_` [protected]

Definition at line 173 of file CbcHeuristicDive.hpp.

**7.42.4.13** `double CbcHeuristicDive::maxTime_` [protected]

Definition at line 176 of file CbcHeuristicDive.hpp.

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

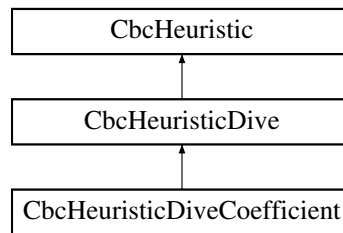
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDive.hpp](#)

## 7.43 CbcHeuristicDiveCoefficient Class Reference

DiveCoefficient class.

```
#include <CbcHeuristicDiveCoefficient.hpp>
```

Inheritance diagram for CbcHeuristicDiveCoefficient:



### Public Member Functions

- [CbcHeuristicDiveCoefficient](#) ()
- [CbcHeuristicDiveCoefficient](#) ([CbcModel](#) &model)
- [CbcHeuristicDiveCoefficient](#) (const [CbcHeuristicDiveCoefficient](#) &)
- [~CbcHeuristicDiveCoefficient](#) ()
- virtual  
[CbcHeuristicDiveCoefficient](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicDiveCoefficient](#) & [operator=](#) (const [CbcHeuristicDiveCoefficient](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual bool [selectVariableToBranch](#) ([OsiSolverInterface](#) \*solver, const double \*newSolution, int &bestColumn, int &bestRound)  
*Selects the next variable to branch on.*

### Additional Inherited Members

#### 7.43.1 Detailed Description

DiveCoefficient class.

Definition at line 14 of file [CbcHeuristicDiveCoefficient.hpp](#).

#### 7.43.2 Constructor & Destructor Documentation

7.43.2.1 [CbcHeuristicDiveCoefficient::CbcHeuristicDiveCoefficient](#) ( )

7.43.2.2 [CbcHeuristicDiveCoefficient::CbcHeuristicDiveCoefficient](#) ( [CbcModel](#) & model )

7.43.2.3 [CbcHeuristicDiveCoefficient::CbcHeuristicDiveCoefficient](#) ( const [CbcHeuristicDiveCoefficient](#) & )

7.43.2.4 [CbcHeuristicDiveCoefficient::~~CbcHeuristicDiveCoefficient](#) ( )

#### 7.43.3 Member Function Documentation

7.43.3.1 virtual [CbcHeuristicDiveCoefficient](#)\* [CbcHeuristicDiveCoefficient::clone](#) ( ) const [virtual]

Clone.

Implements [CbcHeuristicDive](#).

### 7.43.3.2 CbcHeuristicDiveCoefficient& CbcHeuristicDiveCoefficient::operator= ( const CbcHeuristicDiveCoefficient & rhs )

Assignment operator.

### 7.43.3.3 virtual void CbcHeuristicDiveCoefficient::generateCpp ( FILE \* fp ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristicDive](#).

### 7.43.3.4 virtual bool CbcHeuristicDiveCoefficient::selectVariableToBranch ( OsiSolverInterface \* solver, const double \* newSolution, int & bestColumn, int & bestRound ) [virtual]

Selects the next variable to branch on.

Returns true if all the fractional variables can be trivially rounded. Returns false, if there is at least one fractional variable that is not trivially roundable. In this case, the bestColumn returned will not be trivially roundable.

Implements [CbcHeuristicDive](#).

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

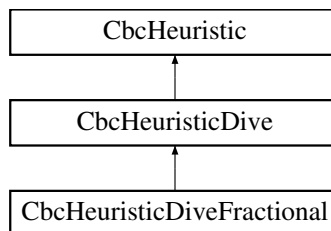
- /home/ted/COIN/trunk/Cbc/src/[CbcHeuristicDiveCoefficient.hpp](#)

## 7.44 CbcHeuristicDiveFractional Class Reference

DiveFractional class.

```
#include <CbcHeuristicDiveFractional.hpp>
```

Inheritance diagram for CbcHeuristicDiveFractional:



### Public Member Functions

- [CbcHeuristicDiveFractional](#) ( )
- [CbcHeuristicDiveFractional](#) (CbcModel &model)
- [CbcHeuristicDiveFractional](#) (const [CbcHeuristicDiveFractional](#) &)
- [~CbcHeuristicDiveFractional](#) ( )
- virtual  
[CbcHeuristicDiveFractional](#) \* clone ( ) const  
*Clone.*
- [CbcHeuristicDiveFractional](#) & operator= (const [CbcHeuristicDiveFractional](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

- virtual bool [selectVariableToBranch](#) (OsiSolverInterface \*solver, const double \*newSolution, int &bestColumn, int &bestRound)

*Selects the next variable to branch on.*

#### Additional Inherited Members

##### 7.44.1 Detailed Description

DiveFractional class.

Definition at line 14 of file CbcHeuristicDiveFractional.hpp.

##### 7.44.2 Constructor & Destructor Documentation

7.44.2.1 CbcHeuristicDiveFractional::CbcHeuristicDiveFractional ( )

7.44.2.2 CbcHeuristicDiveFractional::CbcHeuristicDiveFractional ( CbcModel & model )

7.44.2.3 CbcHeuristicDiveFractional::CbcHeuristicDiveFractional ( const CbcHeuristicDiveFractional & )

7.44.2.4 CbcHeuristicDiveFractional::~~CbcHeuristicDiveFractional ( )

##### 7.44.3 Member Function Documentation

7.44.3.1 virtual CbcHeuristicDiveFractional\* CbcHeuristicDiveFractional::clone ( ) const [virtual]

Clone.

Implements [CbcHeuristicDive](#).

7.44.3.2 CbcHeuristicDiveFractional& CbcHeuristicDiveFractional::operator= ( const CbcHeuristicDiveFractional & rhs )

Assignment operator.

7.44.3.3 virtual void CbcHeuristicDiveFractional::generateCpp ( FILE \* fp ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristicDive](#).

7.44.3.4 virtual bool CbcHeuristicDiveFractional::selectVariableToBranch ( OsiSolverInterface \* solver, const double \* newSolution, int & bestColumn, int & bestRound ) [virtual]

Selects the next variable to branch on.

Returns true if all the fractional variables can be trivially rounded. Returns false, if there is at least one fractional variable that is not trivially roundable. In this case, the bestColumn returned will not be trivially roundable.

Implements [CbcHeuristicDive](#).

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

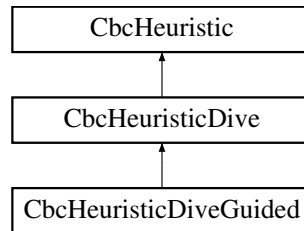
- /home/ted/COIN/trunk/Cbc/src/[CbcHeuristicDiveFractional.hpp](#)

## 7.45 CbcHeuristicDiveGuided Class Reference

DiveGuided class.

```
#include <CbcHeuristicDiveGuided.hpp>
```

Inheritance diagram for CbcHeuristicDiveGuided:



### Public Member Functions

- [CbcHeuristicDiveGuided](#) ()
- [CbcHeuristicDiveGuided](#) ([CbcModel](#) &model)
- [CbcHeuristicDiveGuided](#) (const [CbcHeuristicDiveGuided](#) &)
- [~CbcHeuristicDiveGuided](#) ()
- virtual [CbcHeuristicDiveGuided \\* clone](#) () const  
*Clone.*
- [CbcHeuristicDiveGuided](#) & [operator=](#) (const [CbcHeuristicDiveGuided](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual bool [canHeuristicRun](#) ()  
*Tests if the heuristic can run.*
- virtual bool [selectVariableToBranch](#) ([OsiSolverInterface](#) \*solver, const double \*newSolution, int &bestColumn, int &bestRound)  
*Selects the next variable to branch on.*

### Additional Inherited Members

#### 7.45.1 Detailed Description

DiveGuided class.

Definition at line 14 of file [CbcHeuristicDiveGuided.hpp](#).

#### 7.45.2 Constructor & Destructor Documentation

7.45.2.1 [CbcHeuristicDiveGuided::CbcHeuristicDiveGuided](#) ( )

7.45.2.2 [CbcHeuristicDiveGuided::CbcHeuristicDiveGuided](#) ( [CbcModel](#) & *model* )

7.45.2.3 [CbcHeuristicDiveGuided::CbcHeuristicDiveGuided](#) ( const [CbcHeuristicDiveGuided](#) & )

## 7.45.2.4 CbcHeuristicDiveGuided::~~CbcHeuristicDiveGuided ( )

## 7.45.3 Member Function Documentation

## 7.45.3.1 virtual CbcHeuristicDiveGuided\* CbcHeuristicDiveGuided::clone ( ) const [virtual]

Clone.

Implements [CbcHeuristicDive](#).

## 7.45.3.2 CbcHeuristicDiveGuided&amp; CbcHeuristicDiveGuided::operator= ( const CbcHeuristicDiveGuided &amp; rhs )

Assignment operator.

## 7.45.3.3 virtual void CbcHeuristicDiveGuided::generateCpp ( FILE \* fp ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristicDive](#).

## 7.45.3.4 virtual bool CbcHeuristicDiveGuided::canHeuristicRun ( ) [virtual]

Tests if the heuristic can run.

Reimplemented from [CbcHeuristicDive](#).

## 7.45.3.5 virtual bool CbcHeuristicDiveGuided::selectVariableToBranch ( OsiSolverInterface \* solver, const double \* newSolution, int &amp; bestColumn, int &amp; bestRound ) [virtual]

Selects the next variable to branch on.

Returns true if all the fractional variables can be trivially rounded. Returns false, if there is at least one fractional variable that is not trivially roundable. In this case, the bestColumn returned will not be trivially roundable.

Implements [CbcHeuristicDive](#).

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

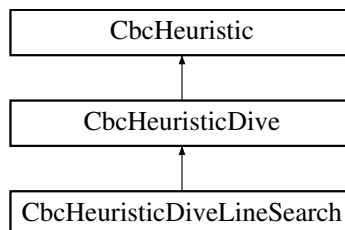
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveGuided.hpp](#)

## 7.46 CbcHeuristicDiveLineSearch Class Reference

DiveLineSearch class.

```
#include <CbcHeuristicDiveLineSearch.hpp>
```

Inheritance diagram for CbcHeuristicDiveLineSearch:



## Public Member Functions

- [CbcHeuristicDiveLineSearch](#) ()
- [CbcHeuristicDiveLineSearch](#) ([CbcModel](#) &model)
- [CbcHeuristicDiveLineSearch](#) (const [CbcHeuristicDiveLineSearch](#) &)
- [~CbcHeuristicDiveLineSearch](#) ()
- virtual  
[CbcHeuristicDiveLineSearch](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicDiveLineSearch](#) & [operator=](#) (const [CbcHeuristicDiveLineSearch](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual bool [selectVariableToBranch](#) (OsiSolverInterface \*solver, const double \*newSolution, int &bestColumn, int &bestRound)  
*Selects the next variable to branch on.*

## Additional Inherited Members

## 7.46.1 Detailed Description

DiveLineSearch class.

Definition at line 14 of file [CbcHeuristicDiveLineSearch.hpp](#).

## 7.46.2 Constructor &amp; Destructor Documentation

7.46.2.1 [CbcHeuristicDiveLineSearch::CbcHeuristicDiveLineSearch](#) ( )

7.46.2.2 [CbcHeuristicDiveLineSearch::CbcHeuristicDiveLineSearch](#) ( [CbcModel](#) & *model* )

7.46.2.3 [CbcHeuristicDiveLineSearch::CbcHeuristicDiveLineSearch](#) ( const [CbcHeuristicDiveLineSearch](#) & )

7.46.2.4 [CbcHeuristicDiveLineSearch::~~CbcHeuristicDiveLineSearch](#) ( )

## 7.46.3 Member Function Documentation

7.46.3.1 virtual [CbcHeuristicDiveLineSearch](#)\* [CbcHeuristicDiveLineSearch::clone](#) ( ) const [virtual]

Clone.

Implements [CbcHeuristicDive](#).

7.46.3.2 [CbcHeuristicDiveLineSearch](#)& [CbcHeuristicDiveLineSearch::operator=](#) ( const [CbcHeuristicDiveLineSearch](#) & *rhs* )

Assignment operator.

7.46.3.3 virtual void [CbcHeuristicDiveLineSearch::generateCpp](#) ( FILE \* *fp* ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristicDive](#).



7.46.3.4 `virtual bool CbcHeuristicDiveLineSearch::selectVariableToBranch ( OsiSolverInterface * solver, const double * newSolution, int & bestColumn, int & bestRound ) [virtual]`

Selects the next variable to branch on.

Returns true if all the fractional variables can be trivially rounded. Returns false, if there is at least one fractional variable that is not trivially roundable. In this case, the bestColumn returned will not be trivially roundable.

Implements [CbcHeuristicDive](#).

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

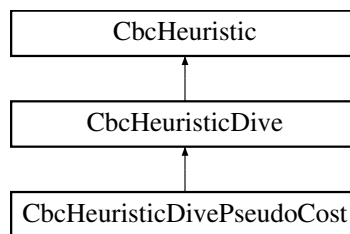
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveLineSearch.hpp](#)

## 7.47 CbcHeuristicDivePseudoCost Class Reference

DivePseudoCost class.

```
#include <CbcHeuristicDivePseudoCost.hpp>
```

Inheritance diagram for CbcHeuristicDivePseudoCost:



### Public Member Functions

- [CbcHeuristicDivePseudoCost \(\)](#)
- [CbcHeuristicDivePseudoCost \(CbcModel &model\)](#)
- [CbcHeuristicDivePseudoCost \(const CbcHeuristicDivePseudoCost &\)](#)
- [~CbcHeuristicDivePseudoCost \(\)](#)
- virtual  
[CbcHeuristicDivePseudoCost \\* clone \(\) const](#)  
*Clone.*
- [CbcHeuristicDivePseudoCost & operator= \(const CbcHeuristicDivePseudoCost &rhs\)](#)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual bool [selectVariableToBranch](#) (OsiSolverInterface \*solver, const double \*newSolution, int &bestColumn, int &bestRound)  
*Selects the next variable to branch on.*
- virtual void [initializeData](#) ()  
*Initializes any data which is going to be used repeatedly in selectVariableToBranch.*
- virtual int [fixOtherVariables](#) (OsiSolverInterface \*solver, const double \*solution, [PseudoReducedCost](#) \*candidate, const double \*random)  
*Fix other variables at bounds.*

## Additional Inherited Members

### 7.47.1 Detailed Description

DivePseudoCost class.

Definition at line 14 of file CbcHeuristicDivePseudoCost.hpp.

### 7.47.2 Constructor & Destructor Documentation

7.47.2.1 `CbcHeuristicDivePseudoCost::CbcHeuristicDivePseudoCost ( )`

7.47.2.2 `CbcHeuristicDivePseudoCost::CbcHeuristicDivePseudoCost ( CbcModel & model )`

7.47.2.3 `CbcHeuristicDivePseudoCost::CbcHeuristicDivePseudoCost ( const CbcHeuristicDivePseudoCost & )`

7.47.2.4 `CbcHeuristicDivePseudoCost::~~CbcHeuristicDivePseudoCost ( )`

### 7.47.3 Member Function Documentation

7.47.3.1 `virtual CbcHeuristicDivePseudoCost* CbcHeuristicDivePseudoCost::clone ( ) const` [virtual]

Clone.

Implements [CbcHeuristicDive](#).

7.47.3.2 `CbcHeuristicDivePseudoCost& CbcHeuristicDivePseudoCost::operator= ( const CbcHeuristicDivePseudoCost & rhs )`

Assignment operator.

7.47.3.3 `virtual void CbcHeuristicDivePseudoCost::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristicDive](#).

7.47.3.4 `virtual bool CbcHeuristicDivePseudoCost::selectVariableToBranch ( OsiSolverInterface * solver, const double * newSolution, int & bestColumn, int & bestRound )` [virtual]

Selects the next variable to branch on.

Returns true if all the fractional variables can be trivially rounded. Returns false, if there is at least one fractional variable that is not trivially roundable. In this case, the bestColumn returned will not be trivially roundable.

Implements [CbcHeuristicDive](#).

7.47.3.5 `virtual void CbcHeuristicDivePseudoCost::initializeData ( )` [virtual]

Initializes any data which is going to be used repeatedly in selectVariableToBranch.

Reimplemented from [CbcHeuristicDive](#).

7.47.3.6 `virtual int CbcHeuristicDivePseudoCost::fixOtherVariables ( OsiSolverInterface * solver, const double * solution, PseudoReducedCost * candidate, const double * random )` [virtual]

Fix other variables at bounds.

Reimplemented from [CbcHeuristicDive](#).

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

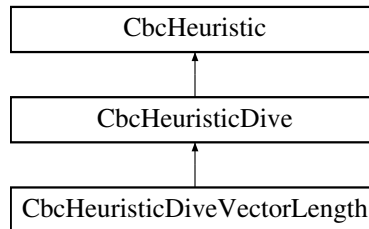
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDivePseudoCost.hpp](#)

## 7.48 CbcHeuristicDiveVectorLength Class Reference

DiveVectorLength class.

```
#include <CbcHeuristicDiveVectorLength.hpp>
```

Inheritance diagram for CbcHeuristicDiveVectorLength:



### Public Member Functions

- [CbcHeuristicDiveVectorLength](#) ()
- [CbcHeuristicDiveVectorLength](#) (CbcModel &model)
- [CbcHeuristicDiveVectorLength](#) (const [CbcHeuristicDiveVectorLength](#) &)
- [~CbcHeuristicDiveVectorLength](#) ()
- virtual  
[CbcHeuristicDiveVectorLength \\* clone](#) () const  
*Clone.*
- [CbcHeuristicDiveVectorLength & operator=](#) (const [CbcHeuristicDiveVectorLength](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual bool [selectVariableToBranch](#) (OsiSolverInterface \*solver, const double \*newSolution, int &bestColumn, int &bestRound)  
*Selects the next variable to branch on.*

### Additional Inherited Members

#### 7.48.1 Detailed Description

DiveVectorLength class.

Definition at line 14 of file CbcHeuristicDiveVectorLength.hpp.

#### 7.48.2 Constructor & Destructor Documentation

##### 7.48.2.1 CbcHeuristicDiveVectorLength::CbcHeuristicDiveVectorLength ( )

##### 7.48.2.2 CbcHeuristicDiveVectorLength::CbcHeuristicDiveVectorLength ( CbcModel & model )

7.48.2.3 `CbcHeuristicDiveVectorLength::CbcHeuristicDiveVectorLength ( const CbcHeuristicDiveVectorLength & )`

7.48.2.4 `CbcHeuristicDiveVectorLength::~~CbcHeuristicDiveVectorLength ( )`

### 7.48.3 Member Function Documentation

7.48.3.1 `virtual CbcHeuristicDiveVectorLength* CbcHeuristicDiveVectorLength::clone ( ) const` `[virtual]`

Clone.

Implements [CbcHeuristicDive](#).

7.48.3.2 `CbcHeuristicDiveVectorLength& CbcHeuristicDiveVectorLength::operator= ( const CbcHeuristicDiveVectorLength & rhs )`

Assignment operator.

7.48.3.3 `virtual void CbcHeuristicDiveVectorLength::generateCpp ( FILE * fp )` `[virtual]`

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristicDive](#).

7.48.3.4 `virtual bool CbcHeuristicDiveVectorLength::selectVariableToBranch ( OsiSolverInterface * solver, const double * newSolution, int & bestColumn, int & bestRound )` `[virtual]`

Selects the next variable to branch on.

Returns true if all the fractional variables can be trivially rounded. Returns false, if there is at least one fractional variable that is not trivially roundable. In this case, the bestColumn returned will not be trivially roundable.

Implements [CbcHeuristicDive](#).

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

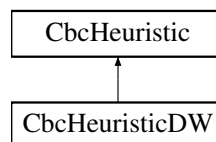
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveVectorLength.hpp](#)

## 7.49 CbcHeuristicDW Class Reference

This is unlike the other heuristics in that it is very very compute intensive.

```
#include <CbcHeuristicDW.hpp>
```

Inheritance diagram for CbcHeuristicDW:



### Public Member Functions

- [CbcHeuristicDW](#) ( )
- [CbcHeuristicDW](#) ( [CbcModel](#) &model, int keepContinuous=0)
- [CbcHeuristicDW](#) ( [CbcModel](#) &model, int callBack([CbcHeuristicDW](#) \*currentHeuristic, [CbcModel](#) \*thisModel, int whereFrom), int keepContinuous=0)

- [CbcHeuristicDW](#) (const [CbcHeuristicDW](#) &)
- [~CbcHeuristicDW](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicDW](#) & [operator=](#) (const [CbcHeuristicDW](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &[objectiveValue](#), double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- int [numberBlocks](#) () const  
*Return number of blocks <=0 - no usable structure.*
- void [passInSolution](#) (const double \*[solution](#))  
*Pass in a solution.*
- void [passInContinuousSolution](#) (const double \*[solution](#))  
*Pass in continuous solution.*
- void [setProposalActions](#) (int fullDWEverySoOften)  
*DW Proposal actions fullDWEverySoOften - 0 - off k - every k times solution gets better.*
- double [objectiveValueWhen](#) (int whichDW) const  
*Objective value when whichDw created.*
- int [numberColumnsDW](#) (int whichDW) const  
*Number of columns in DW.*
- [OsiSolverInterface](#) \* [solver](#) () const  
*Solver.*
- [OsiSolverInterface](#) \* [DWModel](#) (int whichDW) const  
*DW model (user must delete)*
- double [bestObjective](#) () const  
*Best objective value.*
- const double \* [bestSolution](#) () const  
*Best solution found so far.*
- const double \* [continuousSolution](#) () const  
*Continuous solution.*
- const double \* [fixedDj](#) () const  
*Reduced costs of fixed solution.*
- const double \* [objectiveDW](#) () const  
*Objective at which DW updated.*
- int [numberDWTimes](#) () const  
*Number of times we have added to DW model.*
- const int \* [numberColumnsDW](#) () const  
*Number of columns in DW.*
- void [setNumberPasses](#) (int value)  
*Set number of passes.*
- void [setNumberBadPasses](#) (int value)

- Set number of passes without better solution.*
- void [setNumberNeeded](#) (int value)
  - Set number free integers needed (Base value)*
- int [getNumberNeeded](#) () const
  - Get number free integers needed (Base value)*
- void [setCurrentNumberNeeded](#) (int value)
  - Set number free integers needed (Current value)*
- int [getCurrentNumberNeeded](#) () const
  - Get number free integers needed (Current value)*
- void [setNumberNodes](#) (int value)
  - Set number nodes (could be done in callback) (Base value)*
- int [getNumberNodes](#) () const
  - Get number nodes (could be done in callback) (Base value)*
- void [setCurrentNumberNodes](#) (int value)
  - Set number nodes (could be done in callback) (Current value)*
- int [getCurrentNumberNodes](#) () const
  - Get number nodes (could be done in callback) (Current value)*
- void [setTargetObjective](#) (double value)
  - Set target objective.*
- void [setHowOften](#) (int value)
  - Sets how often to do it.*
- const int \* [whichRowBlock](#) () const
  - Block for every row.*
- const int \* [whichColumnBlock](#) () const
  - Block for every column.*
- double \* [initialLower](#) () const
  - Initial Lower bounds.*
- double \* [initialUpper](#) () const
  - Initial Upper bounds.*
- int \* [intArrays](#) () const
  - Local integer arrays (each numberBlocks\_ long)*
- double \* [doubleArrays](#) () const
  - Local double arrays (each numberBlocks\_ long)*
- int [phase](#) () const
  - Phase of solution.*
- int [pass](#) () const
  - Pass number.*
- const int \* [columnsInBlock](#) () const
  - Which columns are in block.*
- const int \* [startColumnBlock](#) () const
  - Starts for columnsInBlock.*
- const int \* [intsInBlock](#) () const
  - Number of integer variables in each block.*
- double [objectiveValue](#) (const double \*[solution](#))
  - Objective value (could also check validity)*

## Protected Types

- typedef int(\* [heuristicCallBack](#) )(CbcHeuristicDW \*, CbcModel \*, int)

## Protected Attributes

- double [targetObjective\\_](#)  
*Target objective.*
- double [bestObjective\\_](#)  
*Best objective value.*
- double [lastObjective\\_](#)  
*Objective value last time.*
- [heuristicCallBack](#) [functionPointer\\_](#)  
*Call back whereFrom - 0 - after blocks found but before data setup 1 - after blocks sorted but before used 2 - just before normal branch and bound 3 - after DW has been updated 4 - if better solution found 5 - every time a block might be used next few for adjustment of nNeeded etc 6 - complete search done - no solution 7 - stopped on nodes - no improvement 8 - improving (same as 4 but after nNeeded changed Pointers to local data given by following pointers.*
- int \* [intArray\\_](#)  
*Local integer arrays (each numberBlocks\_ long)*
- double \* [doubleArray\\_](#)  
*Local double arrays (each numberBlocks\_ long)*
- OsiSolverInterface \* [solver\\_](#)  
*Base solver.*
- OsiSolverInterface \* [dwSolver\\_](#)  
*DW solver.*
- double \* [bestSolution\\_](#)  
*Best solution found so far.*
- double \* [continuousSolution\\_](#)  
*Continuous solution.*
- double \* [fixedDj\\_](#)  
*Reduced costs of fixed solution.*
- double \* [saveLower\\_](#)  
*Original lower bounds.*
- double \* [saveUpper\\_](#)  
*Original Upper bounds.*
- double \* [random\\_](#)  
*random numbers for master rows*
- double \* [weights\\_](#)  
*Weights for each proposal.*
- double \* [objectiveDW\\_](#)  
*Objective at which DW updated.*
- int \* [numberColumnsDW\\_](#)  
*Number of columns in each DW.*
- int \* [whichRowBlock\\_](#)  
*Block for every row.*
- int \* [whichColumnBlock\\_](#)  
*Block for every column.*
- int \* [dwBlock\\_](#)

- *Block number for each proposal.*
- int \* [backwardRow\\_](#)  
*Points back to master rows.*
- int \* [rowsInBlock\\_](#)  
*Which rows are in block.*
- int \* [columnsInBlock\\_](#)  
*Which columns are in block.*
- int \* [startRowBlock\\_](#)  
*Starts for rowsInBlock.*
- int \* [startColumnBlock\\_](#)  
*Starts for columnsInBlock.*
- int \* [intsInBlock\\_](#)  
*Number of integer variables in each block.*
- unsigned int \* [fingerPrint\\_](#)  
*Bits set for 1 integers in each block.*
- unsigned short \* [affinity\\_](#)  
*Affinity each block has for other (will be triangular?)*
- int [fullDWEverySoOften\\_](#)  
*DW Proposal actions fullDWEverySoOften - 0 - off k - every k times solution gets better.*
- int [numberPasses\\_](#)  
*Number of passes.*
- int [howOften\\_](#)  
*How often to do (code can change)*
- int [maximumDW\\_](#)  
*Current maximum number of DW proposals.*
- int [numberDW\\_](#)  
*Number of DW proposals.*
- int [numberDWTimes\\_](#)  
*Number of times we have added to DW model.*
- int [sizeFingerPrint\\_](#)  
*Number of unsigned ints needed for each block of fingerPrint.*
- int [numberMasterColumns\\_](#)  
*Number of columns in master.*
- int [numberMasterRows\\_](#)  
*Number of rows in master.*
- int [numberBlocks\\_](#)  
*Number of blocks.*
- int [keepContinuous\\_](#)  
*Action on decomposition - 1 keep continuous, 0 don't.*
- int [phase\\_](#)  
*Phase of solution.*
- int [pass\\_](#)  
*Pass number.*
- int [nNeededBase\\_](#)  
*Base number of integers needed.*
- int [nNodesBase\\_](#)  
*Base number of nodes needed.*



- int [nNeeded\\_](#)  
*Base number of integers needed.*
- int [nNodes\\_](#)  
*Base number of nodes needed.*
- int [numberBadPasses\\_](#)  
*Number of passes without better solution.*
- int [solveState\\_](#)

#### 7.49.1 Detailed Description

This is unlike the other heuristics in that it is very very compute intensive.

It tries to find a DW structure and use that

Definition at line 17 of file CbcHeuristicDW.hpp.

#### 7.49.2 Member Typedef Documentation

7.49.2.1 `typedef int(* CbcHeuristicDW::heuristicCallBack)(CbcHeuristicDW *,CbcModel *, int)` `[protected]`

Definition at line 194 of file CbcHeuristicDW.hpp.

#### 7.49.3 Constructor & Destructor Documentation

7.49.3.1 `CbcHeuristicDW::CbcHeuristicDW ( )`

7.49.3.2 `CbcHeuristicDW::CbcHeuristicDW ( CbcModel & model, int keepContinuous = 0 )`

7.49.3.3 `CbcHeuristicDW::CbcHeuristicDW ( CbcModel & model, int callBackCbcHeuristicDW *currentHeuristic, CbcModel *thisModel, int whereFrom, int keepContinuous = 0 )`

7.49.3.4 `CbcHeuristicDW::CbcHeuristicDW ( const CbcHeuristicDW & )`

7.49.3.5 `CbcHeuristicDW::~~CbcHeuristicDW ( )`

#### 7.49.4 Member Function Documentation

7.49.4.1 `virtual CbcHeuristic* CbcHeuristicDW::clone ( ) const` `[virtual]`

Clone.

Implements [CbcHeuristic](#).

7.49.4.2 `CbcHeuristicDW& CbcHeuristicDW::operator= ( const CbcHeuristicDW & rhs )`

Assignment operator.

7.49.4.3 `virtual void CbcHeuristicDW::generateCpp ( FILE * fp )` `[virtual]`

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.49.4.4 `virtual void CbcHeuristicDW::resetModel ( CbcModel * model ) [virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.49.4.5 `virtual void CbcHeuristicDW::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.49.4.6 `virtual int CbcHeuristicDW::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) This does Relaxation Induced Neighborhood Search

Implements [CbcHeuristic](#).

7.49.4.7 `int CbcHeuristicDW::numberBlocks ( ) const [inline]`

Return number of blocks <=0 - no usable structure.

Definition at line 65 of file CbcHeuristicDW.hpp.

7.49.4.8 `void CbcHeuristicDW::passInSolution ( const double * solution )`

Pass in a solution.

7.49.4.9 `void CbcHeuristicDW::passInContinuousSolution ( const double * solution )`

Pass in continuous solution.

7.49.4.10 `void CbcHeuristicDW::setProposalActions ( int fullDWEverySoOften )`

DW Proposal actions fullDWEverySoOften - 0 - off k - every k times solution gets better.

7.49.4.11 `double CbcHeuristicDW::objectiveValueWhen ( int whichDW ) const`

Objective value when whichDw created.

7.49.4.12 `int CbcHeuristicDW::numberColumnsDW ( int whichDW ) const`

Number of columns in DW.

7.49.4.13 `OsiSolverInterface* CbcHeuristicDW::solver ( ) const [inline]`

Solver.

Definition at line 82 of file CbcHeuristicDW.hpp.

7.49.4.14 `OsiSolverInterface* CbcHeuristicDW::DWModel ( int whichDW ) const`

DW model (user must delete)

7.49.4.15 `double CbcHeuristicDW::bestObjective ( ) const [inline]`

Best objective value.

Definition at line 87 of file CbcHeuristicDW.hpp.

**7.49.4.16** `const double* CbcHeuristicDW::bestSolution ( ) const [inline]`

Best solution found so far.

Definition at line 90 of file CbcHeuristicDW.hpp.

**7.49.4.17** `const double* CbcHeuristicDW::continuousSolution ( ) const [inline]`

Continuous solution.

Definition at line 93 of file CbcHeuristicDW.hpp.

**7.49.4.18** `const double* CbcHeuristicDW::fixedDj ( ) const [inline]`

Reduced costs of fixed solution.

Definition at line 96 of file CbcHeuristicDW.hpp.

**7.49.4.19** `const double* CbcHeuristicDW::objectiveDW ( ) const [inline]`

Objective at which DW updated.

Definition at line 99 of file CbcHeuristicDW.hpp.

**7.49.4.20** `int CbcHeuristicDW::numberDWTimes ( ) const [inline]`

Number of times we have added to DW model.

Definition at line 102 of file CbcHeuristicDW.hpp.

**7.49.4.21** `const int* CbcHeuristicDW::numberColumnsDW ( ) const [inline]`

Number of columns in DW.

Definition at line 105 of file CbcHeuristicDW.hpp.

**7.49.4.22** `void CbcHeuristicDW::setNumberPasses ( int value ) [inline]`

Set number of passes.

Definition at line 108 of file CbcHeuristicDW.hpp.

**7.49.4.23** `void CbcHeuristicDW::setNumberBadPasses ( int value ) [inline]`

Set number of passes without better solution.

Definition at line 111 of file CbcHeuristicDW.hpp.

**7.49.4.24** `void CbcHeuristicDW::setNumberNeeded ( int value ) [inline]`

Set number free integers needed (Base value)

Definition at line 114 of file CbcHeuristicDW.hpp.

**7.49.4.25** `int CbcHeuristicDW::getNumberNeeded ( ) const [inline]`

Get number free integers needed (Base value)

Definition at line 117 of file CbcHeuristicDW.hpp.

7.49.4.26 `void CbcHeuristicDW::setCurrentNumberNeeded ( int value ) [inline]`

Set number free integers needed (Current value)

Definition at line 120 of file CbcHeuristicDW.hpp.

7.49.4.27 `int CbcHeuristicDW::getCurrentNumberNeeded ( ) const [inline]`

Get number free integers needed (Current value)

Definition at line 123 of file CbcHeuristicDW.hpp.

7.49.4.28 `void CbcHeuristicDW::setNumberNodes ( int value ) [inline]`

Set number nodes (could be done in callback) (Base value)

Definition at line 126 of file CbcHeuristicDW.hpp.

7.49.4.29 `int CbcHeuristicDW::getNumberNodes ( ) const [inline]`

Get number nodes (could be done in callback) (Base value)

Definition at line 129 of file CbcHeuristicDW.hpp.

7.49.4.30 `void CbcHeuristicDW::setCurrentNumberNodes ( int value ) [inline]`

Set number nodes (could be done in callback) (Current value)

Definition at line 132 of file CbcHeuristicDW.hpp.

7.49.4.31 `int CbcHeuristicDW::getCurrentNumberNodes ( ) const [inline]`

Get number nodes (could be done in callback) (Current value)

Definition at line 135 of file CbcHeuristicDW.hpp.

7.49.4.32 `void CbcHeuristicDW::setTargetObjective ( double value ) [inline]`

Set target objective.

Definition at line 138 of file CbcHeuristicDW.hpp.

7.49.4.33 `void CbcHeuristicDW::setHowOften ( int value ) [inline]`

Sets how often to do it.

Definition at line 141 of file CbcHeuristicDW.hpp.

7.49.4.34 `const int* CbcHeuristicDW::whichRowBlock ( ) const [inline]`

Block for every row.

Definition at line 145 of file CbcHeuristicDW.hpp.

7.49.4.35 `const int* CbcHeuristicDW::whichColumnBlock ( ) const [inline]`

Block for every column.

Definition at line 148 of file CbcHeuristicDW.hpp.

**7.49.4.36** `double* CbcHeuristicDW::initialLower ( ) const [inline]`

Initial Lower bounds.

Definition at line 151 of file CbcHeuristicDW.hpp.

**7.49.4.37** `double* CbcHeuristicDW::initialUpper ( ) const [inline]`

Initial Upper bounds.

Definition at line 154 of file CbcHeuristicDW.hpp.

**7.49.4.38** `int* CbcHeuristicDW::intArrays ( ) const [inline]`

Local integer arrays (each numberBlocks\_ long)

Definition at line 157 of file CbcHeuristicDW.hpp.

**7.49.4.39** `double* CbcHeuristicDW::doubleArrays ( ) const [inline]`

Local double arrays (each numberBlocks\_ long)

Definition at line 160 of file CbcHeuristicDW.hpp.

**7.49.4.40** `int CbcHeuristicDW::phase ( ) const [inline]`

Phase of solution.

Definition at line 163 of file CbcHeuristicDW.hpp.

**7.49.4.41** `int CbcHeuristicDW::pass ( ) const [inline]`

Pass number.

Definition at line 166 of file CbcHeuristicDW.hpp.

**7.49.4.42** `const int* CbcHeuristicDW::columnsInBlock ( ) const [inline]`

Which columns are in block.

Definition at line 169 of file CbcHeuristicDW.hpp.

**7.49.4.43** `const int* CbcHeuristicDW::startColumnBlock ( ) const [inline]`

Starts for columnsInBlock.

Definition at line 172 of file CbcHeuristicDW.hpp.

**7.49.4.44** `const int* CbcHeuristicDW::intsInBlock ( ) const [inline]`

Number of integer variables in each block.

Definition at line 175 of file CbcHeuristicDW.hpp.

**7.49.4.45** `double CbcHeuristicDW::objectiveValue ( const double * solution )`

Objective value (could also check validity)

## 7.49.5 Member Data Documentation

**7.49.5.1 double CbcHeuristicDW::targetObjective\_ [protected]**

Target objective.

Definition at line 197 of file CbcHeuristicDW.hpp.

**7.49.5.2 double CbcHeuristicDW::bestObjective\_ [protected]**

Best objective value.

Definition at line 199 of file CbcHeuristicDW.hpp.

**7.49.5.3 double CbcHeuristicDW::lastObjective\_ [protected]**

Objective value last time.

Definition at line 201 of file CbcHeuristicDW.hpp.

**7.49.5.4 heuristicCallback CbcHeuristicDW::functionPointer\_ [protected]**

Call back whereFrom - 0 - after blocks found but before data setup 1 - after blocks sorted but before used 2 - just before normal branch and bound 3 - after DW has been updated 4 - if better solution found 5 - every time a block might be used next few for adjustment of nNeeded etc 6 - complete search done - no solution 7 - stopped on nodes - no improvement 8 - improving (same as 4 but after nNeeded changed Pointers to local data given by following pointers.

Definition at line 216 of file CbcHeuristicDW.hpp.

**7.49.5.5 int\* CbcHeuristicDW::intArray\_ [protected]**

Local integer arrays (each numberBlocks\_ long)

Definition at line 218 of file CbcHeuristicDW.hpp.

**7.49.5.6 double\* CbcHeuristicDW::doubleArray\_ [protected]**

Local double arrays (each numberBlocks\_ long)

Definition at line 220 of file CbcHeuristicDW.hpp.

**7.49.5.7 OsiSolverInterface\* CbcHeuristicDW::solver\_ [protected]**

Base solver.

Definition at line 222 of file CbcHeuristicDW.hpp.

**7.49.5.8 OsiSolverInterface\* CbcHeuristicDW::dwSolver\_ [protected]**

DW solver.

Definition at line 224 of file CbcHeuristicDW.hpp.

**7.49.5.9 double\* CbcHeuristicDW::bestSolution\_ [protected]**

Best solution found so far.

Definition at line 226 of file CbcHeuristicDW.hpp.

**7.49.5.10 double\* CbcHeuristicDW::continuousSolution\_ [protected]**

Continuous solution.

Definition at line 228 of file CbcHeuristicDW.hpp.

**7.49.5.11** `double* CbcHeuristicDW::fixedDj_` `[protected]`

Reduced costs of fixed solution.

Definition at line 230 of file CbcHeuristicDW.hpp.

**7.49.5.12** `double* CbcHeuristicDW::saveLower_` `[protected]`

Original lower bounds.

Definition at line 232 of file CbcHeuristicDW.hpp.

**7.49.5.13** `double* CbcHeuristicDW::saveUpper_` `[protected]`

Original Upper bounds.

Definition at line 234 of file CbcHeuristicDW.hpp.

**7.49.5.14** `double* CbcHeuristicDW::random_` `[protected]`

random numbers for master rows

Definition at line 236 of file CbcHeuristicDW.hpp.

**7.49.5.15** `double* CbcHeuristicDW::weights_` `[protected]`

Weights for each proposal.

Definition at line 238 of file CbcHeuristicDW.hpp.

**7.49.5.16** `double* CbcHeuristicDW::objectiveDW_` `[protected]`

Objective at which DW updated.

Definition at line 240 of file CbcHeuristicDW.hpp.

**7.49.5.17** `int* CbcHeuristicDW::numberColumnsDW_` `[protected]`

Number of columns in each DW.

Definition at line 242 of file CbcHeuristicDW.hpp.

**7.49.5.18** `int* CbcHeuristicDW::whichRowBlock_` `[protected]`

Block for every row.

Definition at line 244 of file CbcHeuristicDW.hpp.

**7.49.5.19** `int* CbcHeuristicDW::whichColumnBlock_` `[protected]`

Block for every column.

Definition at line 246 of file CbcHeuristicDW.hpp.

**7.49.5.20** `int* CbcHeuristicDW::dwBlock_` `[protected]`

Block number for each proposal.

Definition at line 248 of file CbcHeuristicDW.hpp.

**7.49.5.21** `int* CbcHeuristicDW::backwardRow_` [protected]

Points back to master rows.

Definition at line 250 of file CbcHeuristicDW.hpp.

**7.49.5.22** `int* CbcHeuristicDW::rowsInBlock_` [protected]

Which rows are in block.

Definition at line 252 of file CbcHeuristicDW.hpp.

**7.49.5.23** `int* CbcHeuristicDW::columnsInBlock_` [protected]

Which columns are in block.

Definition at line 254 of file CbcHeuristicDW.hpp.

**7.49.5.24** `int* CbcHeuristicDW::startRowBlock_` [protected]

Starts for rowsInBlock.

Definition at line 256 of file CbcHeuristicDW.hpp.

**7.49.5.25** `int* CbcHeuristicDW::startColumnBlock_` [protected]

Starts for columnsInBlock.

Definition at line 258 of file CbcHeuristicDW.hpp.

**7.49.5.26** `int* CbcHeuristicDW::intsInBlock_` [protected]

Number of integer variables in each block.

Definition at line 260 of file CbcHeuristicDW.hpp.

**7.49.5.27** `unsigned int* CbcHeuristicDW::fingerPrint_` [protected]

Bits set for 1 integers in each block.

Definition at line 262 of file CbcHeuristicDW.hpp.

**7.49.5.28** `unsigned short* CbcHeuristicDW::affinity_` [protected]

Affinity each block has for other (will be triangular?)

Definition at line 264 of file CbcHeuristicDW.hpp.

**7.49.5.29** `int CbcHeuristicDW::fullDWEverySoOften_` [protected]

DW Proposal actions fullDWEverySoOften - 0 - off k - every k times solution gets better.

Definition at line 270 of file CbcHeuristicDW.hpp.

**7.49.5.30** `int CbcHeuristicDW::numberPasses_` [protected]

Number of passes.

Definition at line 272 of file CbcHeuristicDW.hpp.



**7.49.5.31** int CbcHeuristicDW::howOften\_ [protected]

How often to do (code can change)

Definition at line 274 of file CbcHeuristicDW.hpp.

**7.49.5.32** int CbcHeuristicDW::maximumDW\_ [protected]

Current maximum number of DW proposals.

Definition at line 276 of file CbcHeuristicDW.hpp.

**7.49.5.33** int CbcHeuristicDW::numberDW\_ [protected]

Number of DW proposals.

Definition at line 278 of file CbcHeuristicDW.hpp.

**7.49.5.34** int CbcHeuristicDW::numberDWTimes\_ [protected]

Number of times we have added to DW model.

Definition at line 280 of file CbcHeuristicDW.hpp.

**7.49.5.35** int CbcHeuristicDW::sizeFingerPrint\_ [protected]

Number of unsigned ints needed for each block of fingerPrint.

Definition at line 282 of file CbcHeuristicDW.hpp.

**7.49.5.36** int CbcHeuristicDW::numberMasterColumns\_ [protected]

Number of columns in master.

Definition at line 284 of file CbcHeuristicDW.hpp.

**7.49.5.37** int CbcHeuristicDW::numberMasterRows\_ [protected]

Number of rows in master.

Definition at line 286 of file CbcHeuristicDW.hpp.

**7.49.5.38** int CbcHeuristicDW::numberBlocks\_ [protected]

Number of blocks.

Definition at line 288 of file CbcHeuristicDW.hpp.

**7.49.5.39** int CbcHeuristicDW::keepContinuous\_ [protected]

Action on decomposition - 1 keep continuous, 0 don't.

Definition at line 290 of file CbcHeuristicDW.hpp.

**7.49.5.40** int CbcHeuristicDW::phase\_ [protected]

Phase of solution.

Definition at line 292 of file CbcHeuristicDW.hpp.

**7.49.5.41** `int CbcHeuristicDW::pass_` [protected]

Pass number.

Definition at line 294 of file CbcHeuristicDW.hpp.

**7.49.5.42** `int CbcHeuristicDW::nNeededBase_` [protected]

Base number of integers needed.

Definition at line 296 of file CbcHeuristicDW.hpp.

**7.49.5.43** `int CbcHeuristicDW::nNodesBase_` [protected]

Base number of nodes needed.

Definition at line 298 of file CbcHeuristicDW.hpp.

**7.49.5.44** `int CbcHeuristicDW::nNeeded_` [protected]

Base number of integers needed.

Definition at line 300 of file CbcHeuristicDW.hpp.

**7.49.5.45** `int CbcHeuristicDW::nNodes_` [protected]

Base number of nodes needed.

Definition at line 302 of file CbcHeuristicDW.hpp.

**7.49.5.46** `int CbcHeuristicDW::numberBadPasses_` [protected]

Number of passes without better solution.

Definition at line 304 of file CbcHeuristicDW.hpp.

**7.49.5.47** `int CbcHeuristicDW::solveState_` [protected]

Definition at line 306 of file CbcHeuristicDW.hpp.

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

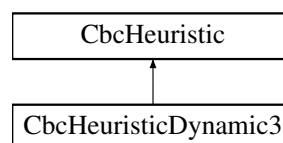
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDW.hpp](#)

## 7.50 CbcHeuristicDynamic3 Class Reference

heuristic - just picks up any good solution

```
#include <CbcLinked.hpp>
```

Inheritance diagram for CbcHeuristicDynamic3:



## Public Member Functions

- [CbcHeuristicDynamic3](#) ()
- [CbcHeuristicDynamic3](#) ([CbcModel](#) &model)
- [CbcHeuristicDynamic3](#) (const [CbcHeuristicDynamic3](#) &)
- [~CbcHeuristicDynamic3](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual bool [canDealWithOdd](#) () const  
*Returns true if can deal with "odd" problems e.g. sos type 2.*

## Additional Inherited Members

## 7.50.1 Detailed Description

heuristic - just picks up any good solution

Definition at line 379 of file CbcLinked.hpp.

## 7.50.2 Constructor &amp; Destructor Documentation

7.50.2.1 [CbcHeuristicDynamic3::CbcHeuristicDynamic3](#) ( )

7.50.2.2 [CbcHeuristicDynamic3::CbcHeuristicDynamic3](#) ( [CbcModel](#) & *model* )

7.50.2.3 [CbcHeuristicDynamic3::CbcHeuristicDynamic3](#) ( const [CbcHeuristicDynamic3](#) & )

7.50.2.4 [CbcHeuristicDynamic3::~~CbcHeuristicDynamic3](#) ( )

## 7.50.3 Member Function Documentation

7.50.3.1 virtual [CbcHeuristic](#)\* [CbcHeuristicDynamic3::clone](#) ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

7.50.3.2 virtual void [CbcHeuristicDynamic3::setModel](#) ( [CbcModel](#) \* *model* ) [virtual]

update model

Reimplemented from [CbcHeuristic](#).

7.50.3.3 virtual int [CbcHeuristicDynamic3::solution](#) ( double & *objectiveValue*, double \* *newSolution* ) [virtual]

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) We leave all variables which are at one at this node of the tree to that value and will initially set all others to zero. We then sort all variables in order of their cost divided by the number of entries in rows which are not yet covered. We randomize that value a bit so that ties will be broken in different ways on different runs of the heuristic. We then choose the best one and set it to one and repeat the exercise.

Implements [CbcHeuristic](#).

**7.50.3.4** `virtual void CbcHeuristicDynamic3::resetModel ( CbcModel * model ) [virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

**7.50.3.5** `virtual bool CbcHeuristicDynamic3::canDealWithOdd ( ) const [inline],[virtual]`

Returns true if can deal with "odd" problems e.g. sos type 2.

Reimplemented from [CbcHeuristic](#).

Definition at line 417 of file CbcLinked.hpp.

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

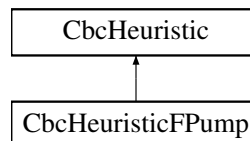
- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.51 CbcHeuristicFPump Class Reference

Feasibility Pump class.

```
#include <CbcHeuristicFPump.hpp>
```

Inheritance diagram for CbcHeuristicFPump:



### Public Member Functions

- [CbcHeuristicFPump](#) ()
- [CbcHeuristicFPump](#) (CbcModel &model, double downValue=0.5, bool roundExpensive=false)
- [CbcHeuristicFPump](#) (const [CbcHeuristicFPump](#) &)
- [~CbcHeuristicFPump](#) ()
- [CbcHeuristicFPump](#) & [operator=](#) (const [CbcHeuristicFPump](#) &rhs)  
*Assignment operator.*
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) (CbcModel \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) (CbcModel \*model)

- update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)
  - returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts.*
- void [setMaximumTime](#) (double value)
  - Set maximum Time (default off) - also sets starttime to current.*
- double [maximumTime](#) () const
  - Get maximum Time (default 0.0 == time limit off)*
- void [setFakeCutoff](#) (double value)
  - Set fake cutoff (default COIN\_DBL\_MAX == off)*
- double [fakeCutoff](#) () const
  - Get fake cutoff (default 0.0 == off)*
- void [setAbsoluteIncrement](#) (double value)
  - Set absolute increment (default 0.0 == off)*
- double [absoluteIncrement](#) () const
  - Get absolute increment (default 0.0 == off)*
- void [setRelativeIncrement](#) (double value)
  - Set relative increment (default 0.0 == off)*
- double [relativeIncrement](#) () const
  - Get relative increment (default 0.0 == off)*
- void [setDefaultRounding](#) (double value)
  - Set default rounding (default 0.5)*
- double [defaultRounding](#) () const
  - Get default rounding (default 0.5)*
- void [setInitialWeight](#) (double value)
  - Set initial weight (default 0.0 == off)*
- double [initialWeight](#) () const
  - Get initial weight (default 0.0 == off)*
- void [setWeightFactor](#) (double value)
  - Set weight factor (default 0.1)*
- double [weightFactor](#) () const
  - Get weight factor (default 0.1)*
- void [setArtificialCost](#) (double value)
  - Set threshold cost for using original cost - even on continuous (default infinity)*
- double [artificialCost](#) () const
  - Get threshold cost for using original cost - even on continuous (default infinity)*
- double [iterationRatio](#) () const
  - Get iteration to size ratio.*
- void [setIterationRatio](#) (double value)
  - Set iteration to size ratio.*
- void [setMaximumPasses](#) (int value)
  - Set maximum passes (default 100)*
- int [maximumPasses](#) () const
  - Get maximum passes (default 100)*
- void [setMaximumRetries](#) (int value)
  - Set maximum retries (default 1)*
- int [maximumRetries](#) () const

- Get maximum retries (default 1)*

  - void `setAccumulate` (int value)
  - Set use of multiple solutions and solves 0 - do not reuse solves, do not accumulate integer solutions for local search 1 - do not reuse solves, accumulate integer solutions for local search 2 - reuse solves, do not accumulate integer solutions for local search 3 - reuse solves, accumulate integer solutions for local search If we add 4 then use second form of problem (with extra rows and variables for general integers) At some point (date?), I added.*
- int `accumulate` () const
- Get accumulation option.*
- void `setFixOnReducedCosts` (int value)
- Set whether to fix variables on known solution 0 - do not fix 1 - fix integers on reduced costs 2 - fix integers on reduced costs but only on entry.*
- int `fixOnReducedCosts` () const
- Get reduced cost option.*
- void `setReducedCostMultiplier` (double value)
- Set reduced cost multiplier 1.0 as normal < 1.0 (x) - pretend gap is x\* actual gap - just for fixing.*
- double `reducedCostMultiplier` () const
- Get reduced cost multiplier.*

#### Protected Attributes

- double `startTime_`
- Start time.*
- double `maximumTime_`
- Maximum Cpu seconds.*
- double `fakeCutoff_`
- Fake cutoff value.*
- double `absoluteIncrement_`
- If positive carry on after solution expecting gain of at least this.*
- double `relativeIncrement_`
- If positive carry on after solution expecting gain of at least this times objective.*
- double `defaultRounding_`
- Default is round up if > this.*
- double `initialWeight_`
- Initial weight for true objective.*
- double `weightFactor_`
- Factor for decreasing weight.*
- double `artificialCost_`
- Threshold cost for using original cost - even on continuous.*
- double `iterationRatio_`
- If iterationRatio > 0 use instead of maximumPasses\_ test is iterations > ratio\*(2\*nrow+ncol)*
- double `reducedCostMultiplier_`
- Reduced cost multiplier 1.0 as normal < 1.0 (x) - pretend gap is x\* actual gap - just for fixing.*
- int `maximumPasses_`
- Maximum number of passes.*
- int `maximumRetries_`
- Maximum number of retries if we find a solution.*
- int `accumulate_`

*Set use of multiple solutions and solves 0 - do not reuse solves, do not accumulate integer solutions for local search 1 - do not reuse solves, accumulate integer solutions for local search 2 - reuse solves, do not accumulate integer solutions for local search 3 - reuse solves, accumulate integer solutions for local search If we add 4 then use second form of problem (with extra rows and variables for general integers) If we do not accumulate solutions then no mini branch and bounds will be done reuse - refers to initial solve after adding in new "cut" If we add 8 then can run after initial cuts (if no solution)*

- int [fixOnReducedCosts\\_](#)

*Set whether to fix variables on known solution 0 - do not fix 1 - fix integers on reduced costs 2 - fix integers on reduced costs but only on entry.*

- bool [roundExpensive\\_](#)

*If true round to expensive.*

### 7.51.1 Detailed Description

Feasibility Pump class.

Definition at line 15 of file CbcHeuristicFPump.hpp.

### 7.51.2 Constructor & Destructor Documentation

#### 7.51.2.1 CbcHeuristicFPump::CbcHeuristicFPump ( )

#### 7.51.2.2 CbcHeuristicFPump::CbcHeuristicFPump ( CbcModel & model, double downValue = 0.5, bool roundExpensive = false )

#### 7.51.2.3 CbcHeuristicFPump::CbcHeuristicFPump ( const CbcHeuristicFPump & )

#### 7.51.2.4 CbcHeuristicFPump::~~CbcHeuristicFPump ( )

### 7.51.3 Member Function Documentation

#### 7.51.3.1 CbcHeuristicFPump& CbcHeuristicFPump::operator= ( const CbcHeuristicFPump & rhs )

Assignment operator.

#### 7.51.3.2 virtual CbcHeuristic\* CbcHeuristicFPump::clone ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

#### 7.51.3.3 virtual void CbcHeuristicFPump::generateCpp ( FILE \* fp ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

#### 7.51.3.4 virtual void CbcHeuristicFPump::resetModel ( CbcModel \* model ) [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

#### 7.51.3.5 virtual void CbcHeuristicFPump::setModel ( CbcModel \* model ) [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

**7.51.3.6** `virtual int CbcHeuristicFPump::solution ( double & objectiveValue, double * newSolution )` `[virtual]`

returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts.

It may make sense for user to call this outside Branch and Cut to get solution. Or normally is just at root node.

new meanings for when\_ - on first try then set back to 1 11 - at end fix all integers at same bound throughout 12 - also fix all integers staying at same internal integral value throughout 13 - also fix all continuous variables staying at same bound throughout 14 - also fix all continuous variables staying at same internal value throughout 15 - as 13 but no internal integers And beyond that, it's apparently possible for the range to be between 21 and 25, in which case it's reduced on entry to [solution\(\)](#) to be between 11 and 15 and allSlack is set to true. Then, if we're not processing general integers, we'll use an all-slack basis to solve ... what? Don't see that yet.

Implements [CbcHeuristic](#).

**7.51.3.7** `void CbcHeuristicFPump::setMaximumTime ( double value )`

Set maximum Time (default off) - also sets starttime to current.

**7.51.3.8** `double CbcHeuristicFPump::maximumTime ( ) const` `[inline]`

Get maximum Time (default 0.0 == time limit off)

Definition at line 71 of file CbcHeuristicFPump.hpp.

**7.51.3.9** `void CbcHeuristicFPump::setFakeCutoff ( double value )` `[inline]`

Set fake cutoff (default COIN\_DBL\_MAX == off)

Definition at line 75 of file CbcHeuristicFPump.hpp.

**7.51.3.10** `double CbcHeuristicFPump::fakeCutoff ( ) const` `[inline]`

Get fake cutoff (default 0.0 == off)

Definition at line 79 of file CbcHeuristicFPump.hpp.

**7.51.3.11** `void CbcHeuristicFPump::setAbsoluteIncrement ( double value )` `[inline]`

Set absolute increment (default 0.0 == off)

Definition at line 83 of file CbcHeuristicFPump.hpp.

**7.51.3.12** `double CbcHeuristicFPump::absoluteIncrement ( ) const` `[inline]`

Get absolute increment (default 0.0 == off)

Definition at line 87 of file CbcHeuristicFPump.hpp.

**7.51.3.13** `void CbcHeuristicFPump::setRelativeIncrement ( double value )` `[inline]`

Set relative increment (default 0.0 == off)

Definition at line 91 of file CbcHeuristicFPump.hpp.

**7.51.3.14** `double CbcHeuristicFPump::relativeIncrement ( ) const` `[inline]`

Get relative increment (default 0.0 == off)

Definition at line 95 of file CbcHeuristicFPump.hpp.



7.51.3.15 void CbcHeuristicFPump::setDefaultRounding ( double *value* ) [inline]

Set default rounding (default 0.5)

Definition at line 99 of file CbcHeuristicFPump.hpp.

7.51.3.16 double CbcHeuristicFPump::defaultRounding ( ) const [inline]

Get default rounding (default 0.5)

Definition at line 103 of file CbcHeuristicFPump.hpp.

7.51.3.17 void CbcHeuristicFPump::setInitialWeight ( double *value* ) [inline]

Set initial weight (default 0.0 == off)

Definition at line 107 of file CbcHeuristicFPump.hpp.

7.51.3.18 double CbcHeuristicFPump::initialWeight ( ) const [inline]

Get initial weight (default 0.0 == off)

Definition at line 111 of file CbcHeuristicFPump.hpp.

7.51.3.19 void CbcHeuristicFPump::setWeightFactor ( double *value* ) [inline]

Set weight factor (default 0.1)

Definition at line 115 of file CbcHeuristicFPump.hpp.

7.51.3.20 double CbcHeuristicFPump::weightFactor ( ) const [inline]

Get weight factor (default 0.1)

Definition at line 119 of file CbcHeuristicFPump.hpp.

7.51.3.21 void CbcHeuristicFPump::setArtificialCost ( double *value* ) [inline]

Set threshold cost for using original cost - even on continuous (default infinity)

Definition at line 123 of file CbcHeuristicFPump.hpp.

7.51.3.22 double CbcHeuristicFPump::artificialCost ( ) const [inline]

Get threshold cost for using original cost - even on continuous (default infinity)

Definition at line 127 of file CbcHeuristicFPump.hpp.

7.51.3.23 double CbcHeuristicFPump::iterationRatio ( ) const [inline]

Get iteration to size ratio.

Definition at line 131 of file CbcHeuristicFPump.hpp.

7.51.3.24 void CbcHeuristicFPump::setIterationRatio ( double *value* ) [inline]

Set iteration to size ratio.

Definition at line 135 of file CbcHeuristicFPump.hpp.

**7.51.3.25** `void CbcHeuristicFPump::setMaximumPasses ( int value ) [inline]`

Set maximum passes (default 100)

Definition at line 139 of file CbcHeuristicFPump.hpp.

**7.51.3.26** `int CbcHeuristicFPump::maximumPasses ( ) const [inline]`

Get maximum passes (default 100)

Definition at line 143 of file CbcHeuristicFPump.hpp.

**7.51.3.27** `void CbcHeuristicFPump::setMaximumRetries ( int value ) [inline]`

Set maximum retries (default 1)

Definition at line 147 of file CbcHeuristicFPump.hpp.

**7.51.3.28** `int CbcHeuristicFPump::maximumRetries ( ) const [inline]`

Get maximum retries (default 1)

Definition at line 151 of file CbcHeuristicFPump.hpp.

**7.51.3.29** `void CbcHeuristicFPump::setAccumulate ( int value ) [inline]`

Set use of multiple solutions and solves 0 - do not reuse solves, do not accumulate integer solutions for local search 1 - do not reuse solves, accumulate integer solutions for local search 2 - reuse solves, do not accumulate integer solutions for local search 3 - reuse solves, accumulate integer solutions for local search If we add 4 then use second form of problem (with extra rows and variables for general integers) At some point (date?), I added.

And then there are a few bit fields: 4 - something about general integers So my (lh) guess for 4 was at least in the ballpark, but I'll have to rethink 8 entirely (and it may well not mean the same thing as it did when I added that comment. 8 - determines whether we process general integers

And on 090831, John added

If we add 4 then use second form of problem (with extra rows and variables for general integers) If we add 8 then can run after initial cuts (if no solution)

Definition at line 175 of file CbcHeuristicFPump.hpp.

**7.51.3.30** `int CbcHeuristicFPump::accumulate ( ) const [inline]`

Get accumulation option.

Definition at line 179 of file CbcHeuristicFPump.hpp.

**7.51.3.31** `void CbcHeuristicFPump::setFixOnReducedCosts ( int value ) [inline]`

Set whether to fix variables on known solution 0 - do not fix 1 - fix integers on reduced costs 2 - fix integers on reduced costs but only on entry.

Definition at line 187 of file CbcHeuristicFPump.hpp.

**7.51.3.32** `int CbcHeuristicFPump::fixOnReducedCosts ( ) const [inline]`

Get reduced cost option.

Definition at line 191 of file CbcHeuristicFPump.hpp.

7.51.3.33 `void CbcHeuristicFPump::setReducedCostMultiplier ( double value ) [inline]`

Set reduced cost multiplier 1.0 as normal <1.0 (x) - pretend gap is x\* actual gap - just for fixing.

Definition at line 198 of file CbcHeuristicFPump.hpp.

7.51.3.34 `double CbcHeuristicFPump::reducedCostMultiplier ( ) const [inline]`

Get reduced cost multiplier.

Definition at line 202 of file CbcHeuristicFPump.hpp.

#### 7.51.4 Member Data Documentation

7.51.4.1 `double CbcHeuristicFPump::startTime_ [protected]`

Start time.

Definition at line 209 of file CbcHeuristicFPump.hpp.

7.51.4.2 `double CbcHeuristicFPump::maximumTime_ [protected]`

Maximum Cpu seconds.

Definition at line 211 of file CbcHeuristicFPump.hpp.

7.51.4.3 `double CbcHeuristicFPump::fakeCutoff_ [protected]`

Fake cutoff value.

If set then better of real cutoff and this used to add a constraint

Definition at line 215 of file CbcHeuristicFPump.hpp.

7.51.4.4 `double CbcHeuristicFPump::absoluteIncrement_ [protected]`

If positive carry on after solution expecting gain of at least this.

Definition at line 217 of file CbcHeuristicFPump.hpp.

7.51.4.5 `double CbcHeuristicFPump::relativeIncrement_ [protected]`

If positive carry on after solution expecting gain of at least this times objective.

Definition at line 219 of file CbcHeuristicFPump.hpp.

7.51.4.6 `double CbcHeuristicFPump::defaultRounding_ [protected]`

Default is round up if > this.

Definition at line 221 of file CbcHeuristicFPump.hpp.

7.51.4.7 `double CbcHeuristicFPump::initialWeight_ [protected]`

Initial weight for true objective.

Definition at line 223 of file CbcHeuristicFPump.hpp.

7.51.4.8 `double CbcHeuristicFPump::weightFactor_ [protected]`

Factor for decreasing weight.

Definition at line 225 of file CbcHeuristicFPump.hpp.

**7.51.4.9** `double CbcHeuristicFPump::artificialCost_` [protected]

Threshold cost for using original cost - even on continuous.

Definition at line 227 of file CbcHeuristicFPump.hpp.

**7.51.4.10** `double CbcHeuristicFPump::iterationRatio_` [protected]

If iterationRatio >0 use instead of maximumPasses\_ test is iterations > ratio\*(2\*nrow+ncol)

Definition at line 230 of file CbcHeuristicFPump.hpp.

**7.51.4.11** `double CbcHeuristicFPump::reducedCostMultiplier_` [protected]

Reduced cost multiplier 1.0 as normal <1.0 (x) - pretend gap is x\* actual gap - just for fixing.

Definition at line 235 of file CbcHeuristicFPump.hpp.

**7.51.4.12** `int CbcHeuristicFPump::maximumPasses_` [protected]

Maximum number of passes.

Definition at line 237 of file CbcHeuristicFPump.hpp.

**7.51.4.13** `int CbcHeuristicFPump::maximumRetries_` [protected]

Maximum number of retries if we find a solution.

If negative we clean out used array

Definition at line 241 of file CbcHeuristicFPump.hpp.

**7.51.4.14** `int CbcHeuristicFPump::accumulate_` [protected]

Set use of multiple solutions and solves 0 - do not reuse solves, do not accumulate integer solutions for local search 1 - do not reuse solves, accumulate integer solutions for local search 2 - reuse solves, do not accumulate integer solutions for local search 3 - reuse solves, accumulate integer solutions for local search If we add 4 then use second form of problem (with extra rows and variables for general integers) If we do not accumulate solutions then no mini branch and bounds will be done reuse - refers to initial solve after adding in new "cut" If we add 8 then can run after initial cuts (if no solution)

Definition at line 252 of file CbcHeuristicFPump.hpp.

**7.51.4.15** `int CbcHeuristicFPump::fixOnReducedCosts_` [protected]

Set whether to fix variables on known solution 0 - do not fix 1 - fix integers on reduced costs 2 - fix integers on reduced costs but only on entry.

Definition at line 258 of file CbcHeuristicFPump.hpp.

**7.51.4.16** `bool CbcHeuristicFPump::roundExpensive_` [protected]

If true round to expensive.

Definition at line 260 of file CbcHeuristicFPump.hpp.

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

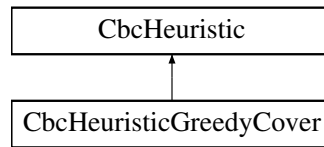
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicFPump.hpp](#)

## 7.52 CbcHeuristicGreedyCover Class Reference

Greedy heuristic classes.

```
#include <CbcHeuristicGreedy.hpp>
```

Inheritance diagram for CbcHeuristicGreedyCover:



### Public Member Functions

- [CbcHeuristicGreedyCover](#) ()
- [CbcHeuristicGreedyCover](#) ([CbcModel](#) &model)
- [CbcHeuristicGreedyCover](#) (const [CbcHeuristicGreedyCover](#) &)
- [~CbcHeuristicGreedyCover](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicGreedyCover](#) & [operator=](#) (const [CbcHeuristicGreedyCover](#) &rhs)  
*Assignment operator.*
- virtual void [generateCxx](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- virtual void [validate](#) ()  
*Validate model i.e. sets when\_ to 0 if necessary (may be NULL)*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- int [algorithm](#) () const
- void [setAlgorithm](#) (int value)
- int [numberTimes](#) () const
- void [setNumberTimes](#) (int value)

### Protected Member Functions

- void [gutsOfConstructor](#) ([CbcModel](#) \*model)  
*Guts of constructor from a [CbcModel](#).*

### Protected Attributes

- CoinPackedMatrix [matrix\\_](#)
- int [originalNumberRows\\_](#)
- int [algorithm\\_](#)
- int [numberTimes\\_](#)  
*Do this many times.*

### 7.52.1 Detailed Description

Greedy heuristic classes.

Definition at line 13 of file CbcHeuristicGreedy.hpp.

### 7.52.2 Constructor & Destructor Documentation

7.52.2.1 `CbcHeuristicGreedyCover::CbcHeuristicGreedyCover ( )`

7.52.2.2 `CbcHeuristicGreedyCover::CbcHeuristicGreedyCover ( CbcModel & model )`

7.52.2.3 `CbcHeuristicGreedyCover::CbcHeuristicGreedyCover ( const CbcHeuristicGreedyCover & )`

7.52.2.4 `CbcHeuristicGreedyCover::~~CbcHeuristicGreedyCover ( )`

### 7.52.3 Member Function Documentation

7.52.3.1 `virtual CbcHeuristic* CbcHeuristicGreedyCover::clone ( ) const` [virtual]

Clone.

Implements [CbcHeuristic](#).

7.52.3.2 `CbcHeuristicGreedyCover& CbcHeuristicGreedyCover::operator= ( const CbcHeuristicGreedyCover & rhs )`

Assignment operator.

7.52.3.3 `virtual void CbcHeuristicGreedyCover::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.52.3.4 `virtual void CbcHeuristicGreedyCover::setModel ( CbcModel * model )` [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.52.3.5 `virtual int CbcHeuristicGreedyCover::solution ( double & objectiveValue, double * newSolution )` [virtual]

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) We leave all variables which are at one at this node of the tree to that value and will initially set all others to zero. We then sort all variables in order of their cost divided by the number of entries in rows which are not yet covered. We randomize that value a bit so that ties will be broken in different ways on different runs of the heuristic. We then choose the best one and set it to one and repeat the exercise.

Implements [CbcHeuristic](#).

7.52.3.6 `virtual void CbcHeuristicGreedyCover::validate ( )` [virtual]

Validate model i.e. sets when\_ to 0 if necessary (may be NULL)

Reimplemented from [CbcHeuristic](#).

7.52.3.7 `virtual void CbcHeuristicGreedyCover::resetModel ( CbcModel * model )` [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.52.3.8 `int CbcHeuristicGreedyCover::algorithm ( ) const` [inline]

Definition at line 63 of file CbcHeuristicGreedy.hpp.

7.52.3.9 `void CbcHeuristicGreedyCover::setAlgorithm ( int value )` [inline]

Definition at line 66 of file CbcHeuristicGreedy.hpp.

7.52.3.10 `int CbcHeuristicGreedyCover::numberOfTimes ( ) const` [inline]

Definition at line 70 of file CbcHeuristicGreedy.hpp.

7.52.3.11 `void CbcHeuristicGreedyCover::setNumberOfTimes ( int value )` [inline]

Definition at line 73 of file CbcHeuristicGreedy.hpp.

7.52.3.12 `void CbcHeuristicGreedyCover::gutsOfConstructor ( CbcModel * model )` [protected]

Guts of constructor from a [CbcModel](#).

## 7.52.4 Member Data Documentation

7.52.4.1 `CoinPackedMatrix CbcHeuristicGreedyCover::matrix_` [protected]

Definition at line 83 of file CbcHeuristicGreedy.hpp.

7.52.4.2 `int CbcHeuristicGreedyCover::originalNumberOfRows_` [protected]

Definition at line 85 of file CbcHeuristicGreedy.hpp.

7.52.4.3 `int CbcHeuristicGreedyCover::algorithm_` [protected]

Definition at line 91 of file CbcHeuristicGreedy.hpp.

7.52.4.4 `int CbcHeuristicGreedyCover::numberOfTimes_` [protected]

Do this many times.

Definition at line 93 of file CbcHeuristicGreedy.hpp.

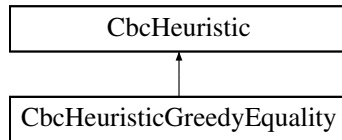
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicGreedy.hpp](#)

## 7.53 CbcHeuristicGreedyEquality Class Reference

```
#include <CbcHeuristicGreedy.hpp>
```

Inheritance diagram for CbcHeuristicGreedyEquality:



### Public Member Functions

- [CbcHeuristicGreedyEquality](#) ()
- [CbcHeuristicGreedyEquality](#) ([CbcModel](#) &model)
- [CbcHeuristicGreedyEquality](#) (const [CbcHeuristicGreedyEquality](#) &)
- [~CbcHeuristicGreedyEquality](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicGreedyEquality](#) & [operator=](#) (const [CbcHeuristicGreedyEquality](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- virtual void [validate](#) ()  
*Validate model i.e. sets when\_ to 0 if necessary (may be NULL)*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- int [algorithm](#) () const
- void [setAlgorithm](#) (int value)
- void [setFraction](#) (double value)
- double [fraction](#) () const
- int [numberTimes](#) () const
- void [setNumberTimes](#) (int value)

### Protected Member Functions

- void [gutsOfConstructor](#) ([CbcModel](#) \*model)  
*Guts of constructor from a [CbcModel](#).*

### Protected Attributes

- CoinPackedMatrix [matrix\\_](#)
- double [fraction\\_](#)
- int [originalNumberRows\\_](#)
- int [algorithm\\_](#)
- int [numberTimes\\_](#)  
*Do this many times.*



## 7.53.1 Detailed Description

Definition at line 98 of file CbcHeuristicGreedy.hpp.

## 7.53.2 Constructor &amp; Destructor Documentation

7.53.2.1 CbcHeuristicGreedyEquality::CbcHeuristicGreedyEquality ( )

7.53.2.2 CbcHeuristicGreedyEquality::CbcHeuristicGreedyEquality ( CbcModel & *model* )

7.53.2.3 CbcHeuristicGreedyEquality::CbcHeuristicGreedyEquality ( const CbcHeuristicGreedyEquality & )

7.53.2.4 CbcHeuristicGreedyEquality::~CbcHeuristicGreedyEquality ( )

## 7.53.3 Member Function Documentation

7.53.3.1 virtual CbcHeuristic\* CbcHeuristicGreedyEquality::clone ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

7.53.3.2 CbcHeuristicGreedyEquality& CbcHeuristicGreedyEquality::operator= ( const CbcHeuristicGreedyEquality & *rhs* )

Assignment operator.

7.53.3.3 virtual void CbcHeuristicGreedyEquality::generateCpp ( FILE \* *fp* ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.53.3.4 virtual void CbcHeuristicGreedyEquality::setModel ( CbcModel \* *model* ) [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.53.3.5 virtual int CbcHeuristicGreedyEquality::solution ( double & *objectiveValue*, double \* *newSolution* ) [virtual]

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) We leave all variables which are at one at this node of the tree to that value and will initially set all others to zero. We then sort all variables in order of their cost divided by the number of entries in rows which are not yet covered. We randomize that value a bit so that ties will be broken in different ways on different runs of the heuristic. We then choose the best one and set it to one and repeat the exercise.

Implements [CbcHeuristic](#).

7.53.3.6 virtual void CbcHeuristicGreedyEquality::validate ( ) [virtual]

Validate model i.e. sets when\_ to 0 if necessary (may be NULL)

Reimplemented from [CbcHeuristic](#).

7.53.3.7 virtual void CbcHeuristicGreedyEquality::resetModel ( CbcModel \* *model* ) [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

**7.53.3.8** `int CbcHeuristicGreedyEquality::algorithm ( ) const [inline]`

Definition at line 148 of file CbcHeuristicGreedy.hpp.

**7.53.3.9** `void CbcHeuristicGreedyEquality::setAlgorithm ( int value ) [inline]`

Definition at line 151 of file CbcHeuristicGreedy.hpp.

**7.53.3.10** `void CbcHeuristicGreedyEquality::setFraction ( double value ) [inline]`

Definition at line 155 of file CbcHeuristicGreedy.hpp.

**7.53.3.11** `double CbcHeuristicGreedyEquality::fraction ( ) const [inline]`

Definition at line 158 of file CbcHeuristicGreedy.hpp.

**7.53.3.12** `int CbcHeuristicGreedyEquality::numberOfTimes ( ) const [inline]`

Definition at line 162 of file CbcHeuristicGreedy.hpp.

**7.53.3.13** `void CbcHeuristicGreedyEquality::setNumberOfTimes ( int value ) [inline]`

Definition at line 165 of file CbcHeuristicGreedy.hpp.

**7.53.3.14** `void CbcHeuristicGreedyEquality::gutsOfConstructor ( CbcModel * model ) [protected]`

Guts of constructor from a [CbcModel](#).

#### 7.53.4 Member Data Documentation

**7.53.4.1** `CoinPackedMatrix CbcHeuristicGreedyEquality::matrix_ [protected]`

Definition at line 174 of file CbcHeuristicGreedy.hpp.

**7.53.4.2** `double CbcHeuristicGreedyEquality::fraction_ [protected]`

Definition at line 176 of file CbcHeuristicGreedy.hpp.

**7.53.4.3** `int CbcHeuristicGreedyEquality::originalNumberOfRows_ [protected]`

Definition at line 178 of file CbcHeuristicGreedy.hpp.

**7.53.4.4** `int CbcHeuristicGreedyEquality::algorithm_ [protected]`

Definition at line 184 of file CbcHeuristicGreedy.hpp.

**7.53.4.5** `int CbcHeuristicGreedyEquality::numberOfTimes_ [protected]`

Do this many times.

Definition at line 186 of file CbcHeuristicGreedy.hpp.

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

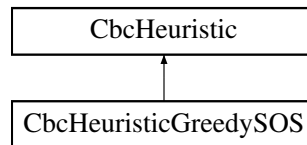
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicGreedy.hpp](#)

## 7.54 CbcHeuristicGreedySOS Class Reference

Greedy heuristic for SOS and L rows (and positive elements)

```
#include <CbcHeuristicGreedy.hpp>
```

Inheritance diagram for CbcHeuristicGreedySOS:



### Public Member Functions

- [CbcHeuristicGreedySOS](#) ()
- [CbcHeuristicGreedySOS](#) ([CbcModel](#) &model)
- [CbcHeuristicGreedySOS](#) (const [CbcHeuristicGreedySOS](#) &)
- [~CbcHeuristicGreedySOS](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicGreedySOS](#) & [operator=](#) (const [CbcHeuristicGreedySOS](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- virtual void [validate](#) ()  
*Validate model i.e. sets when\_ to 0 if necessary (may be NULL)*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- int [algorithm](#) () const
- void [setAlgorithm](#) (int value)
- int [numberTimes](#) () const
- void [setNumberTimes](#) (int value)

### Protected Member Functions

- void [gutsOfConstructor](#) ([CbcModel](#) \*model)  
*Guts of constructor from a [CbcModel](#).*

### Protected Attributes

- double \* [originalRhs\\_](#)
- CoinPackedMatrix [matrix\\_](#)
- int [originalNumberRows\\_](#)

- int [algorithm\\_](#)
- int [numberTimes\\_](#)  
*Do this many times.*

#### 7.54.1 Detailed Description

Greedy heuristic for SOS and L rows (and positive elements)

Definition at line 193 of file CbcHeuristicGreedy.hpp.

#### 7.54.2 Constructor & Destructor Documentation

7.54.2.1 `CbcHeuristicGreedySOS::CbcHeuristicGreedySOS ( )`

7.54.2.2 `CbcHeuristicGreedySOS::CbcHeuristicGreedySOS ( CbcModel & model )`

7.54.2.3 `CbcHeuristicGreedySOS::CbcHeuristicGreedySOS ( const CbcHeuristicGreedySOS & )`

7.54.2.4 `CbcHeuristicGreedySOS::~~CbcHeuristicGreedySOS ( )`

#### 7.54.3 Member Function Documentation

7.54.3.1 `virtual CbcHeuristic* CbcHeuristicGreedySOS::clone ( ) const` `[virtual]`

Clone.

Implements [CbcHeuristic](#).

7.54.3.2 `CbcHeuristicGreedySOS& CbcHeuristicGreedySOS::operator= ( const CbcHeuristicGreedySOS & rhs )`

Assignment operator.

7.54.3.3 `virtual void CbcHeuristicGreedySOS::generateCpp ( FILE * fp )` `[virtual]`

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.54.3.4 `virtual void CbcHeuristicGreedySOS::setModel ( CbcModel * model )` `[virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.54.3.5 `virtual int CbcHeuristicGreedySOS::solution ( double & objectiveValue, double * newSolution )` `[virtual]`

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) We leave all variables which are at one at this node of the tree to that value and will initially set all others to zero. We then sort all variables in order of their cost divided by the number of entries in rows which are not yet covered. We randomize that value a bit so that ties will be broken in different ways on different runs of the heuristic. We then choose the best one and set it to one and repeat the exercise.

Implements [CbcHeuristic](#).

7.54.3.6 `virtual void CbcHeuristicGreedySOS::validate ( )` `[virtual]`

Validate model i.e. sets when\_ to 0 if necessary (may be NULL)

Reimplemented from [CbcHeuristic](#).

7.54.3.7 `virtual void CbcHeuristicGreedySOS::resetModel ( CbcModel * model )` [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.54.3.8 `int CbcHeuristicGreedySOS::algorithm ( ) const` [inline]

Definition at line 245 of file `CbcHeuristicGreedy.hpp`.

7.54.3.9 `void CbcHeuristicGreedySOS::setAlgorithm ( int value )` [inline]

Definition at line 248 of file `CbcHeuristicGreedy.hpp`.

7.54.3.10 `int CbcHeuristicGreedySOS::numberOfTimes ( ) const` [inline]

Definition at line 252 of file `CbcHeuristicGreedy.hpp`.

7.54.3.11 `void CbcHeuristicGreedySOS::setNumberOfTimes ( int value )` [inline]

Definition at line 255 of file `CbcHeuristicGreedy.hpp`.

7.54.3.12 `void CbcHeuristicGreedySOS::gutsOfConstructor ( CbcModel * model )` [protected]

Guts of constructor from a [CbcModel](#).

#### 7.54.4 Member Data Documentation

7.54.4.1 `double* CbcHeuristicGreedySOS::originalRhs_` [protected]

Definition at line 265 of file `CbcHeuristicGreedy.hpp`.

7.54.4.2 `CoinPackedMatrix CbcHeuristicGreedySOS::matrix_` [protected]

Definition at line 267 of file `CbcHeuristicGreedy.hpp`.

7.54.4.3 `int CbcHeuristicGreedySOS::originalNumberRows_` [protected]

Definition at line 269 of file `CbcHeuristicGreedy.hpp`.

7.54.4.4 `int CbcHeuristicGreedySOS::algorithm_` [protected]

Definition at line 272 of file `CbcHeuristicGreedy.hpp`.

7.54.4.5 `int CbcHeuristicGreedySOS::numberOfTimes_` [protected]

Do this many times.

Definition at line 274 of file `CbcHeuristicGreedy.hpp`.

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

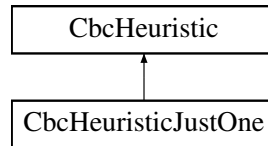
- `/home/ted/COIN/trunk/Cbc/src/CbcHeuristicGreedy.hpp`

## 7.55 CbcHeuristicJustOne Class Reference

Just One class - this chooses one at random.

```
#include <CbcHeuristic.hpp>
```

Inheritance diagram for CbcHeuristicJustOne:



### Public Member Functions

- [CbcHeuristicJustOne](#) ()
- [CbcHeuristicJustOne](#) ([CbcModel](#) &model)
- [CbcHeuristicJustOne](#) (const [CbcHeuristicJustOne](#) &)
- [~CbcHeuristicJustOne](#) ()
- virtual [CbcHeuristicJustOne](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicJustOne](#) & [operator=](#) (const [CbcHeuristicJustOne](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts This does Fractional Diving*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual bool [selectVariableToBranch](#) ([OsiSolverInterface](#) \*, const double \*, int &, int &)  
*Selects the next variable to branch on.*
- virtual void [validate](#) ()  
*Validate model i.e. sets when\_ to 0 if necessary (may be NULL)*
- void [addHeuristic](#) (const [CbcHeuristic](#) \*heuristic, double probability)  
*Adds an heuristic with probability.*
- void [normalizeProbabilities](#) ()  
*Normalize probabilities.*

### Protected Attributes

- double \* [probabilities\\_](#)
- [CbcHeuristic](#) \*\* [heuristic\\_](#)
- int [numberHeuristics\\_](#)

## 7.55.1 Detailed Description

Just One class - this chooses one at random.

Definition at line 601 of file CbcHeuristic.hpp.

## 7.55.2 Constructor &amp; Destructor Documentation

7.55.2.1 `CbcHeuristicJustOne::CbcHeuristicJustOne ( )`

7.55.2.2 `CbcHeuristicJustOne::CbcHeuristicJustOne ( CbcModel & model )`

7.55.2.3 `CbcHeuristicJustOne::CbcHeuristicJustOne ( const CbcHeuristicJustOne & )`

7.55.2.4 `CbcHeuristicJustOne::~~CbcHeuristicJustOne ( )`

## 7.55.3 Member Function Documentation

7.55.3.1 `virtual CbcHeuristicJustOne* CbcHeuristicJustOne::clone ( ) const` `[virtual]`

Clone.

Implements [CbcHeuristic](#).

7.55.3.2 `CbcHeuristicJustOne& CbcHeuristicJustOne::operator= ( const CbcHeuristicJustOne & rhs )`

Assignment operator.

7.55.3.3 `virtual void CbcHeuristicJustOne::generateCpp ( FILE * fp )` `[virtual]`

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.55.3.4 `virtual int CbcHeuristicJustOne::solution ( double & objectiveValue, double * newSolution )` `[virtual]`

returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts This does Fractional Diving

Implements [CbcHeuristic](#).

7.55.3.5 `virtual void CbcHeuristicJustOne::resetModel ( CbcModel * model )` `[virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.55.3.6 `virtual void CbcHeuristicJustOne::setModel ( CbcModel * model )` `[virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.55.3.7 `virtual bool CbcHeuristicJustOne::selectVariableToBranch ( OsiSolverInterface *, const double *, int &, int & )`  
`[inline], [virtual]`

Selects the next variable to branch on.

Returns true if all the fractional variables can be trivially rounded. Returns false, if there is at least one fractional variable

that is not trivially roundable. In this case, the bestColumn returned will not be trivially roundable. This is dummy as never called

Definition at line 645 of file CbcHeuristic.hpp.

**7.55.3.8** `virtual void CbcHeuristicJustOne::validate ( ) [virtual]`

Validate model i.e. sets when\_ to 0 if necessary (may be NULL)

Reimplemented from [CbcHeuristic](#).

**7.55.3.9** `void CbcHeuristicJustOne::addHeuristic ( const CbcHeuristic * heuristic, double probability )`

Adds an heuristic with probability.

**7.55.3.10** `void CbcHeuristicJustOne::normalizeProbabilities ( )`

Normalize probabilities.

#### 7.55.4 Member Data Documentation

**7.55.4.1** `double* CbcHeuristicJustOne::probabilities_ [protected]`

Definition at line 661 of file CbcHeuristic.hpp.

**7.55.4.2** `CbcHeuristic** CbcHeuristicJustOne::heuristic_ [protected]`

Definition at line 664 of file CbcHeuristic.hpp.

**7.55.4.3** `int CbcHeuristicJustOne::numberHeuristics_ [protected]`

Definition at line 667 of file CbcHeuristic.hpp.

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

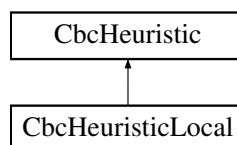
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp](#)

## 7.56 CbcHeuristicLocal Class Reference

LocalSearch class.

```
#include <CbcHeuristicLocal.hpp>
```

Inheritance diagram for CbcHeuristicLocal:



#### Public Member Functions

- [CbcHeuristicLocal \( \)](#)
- [CbcHeuristicLocal \(CbcModel &model\)](#)



- [CbcHeuristicLocal](#) (const [CbcHeuristicLocal](#) &)
- [~CbcHeuristicLocal](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicLocal](#) & [operator=](#) (const [CbcHeuristicLocal](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- int [solutionFix](#) (double &objectiveValue, double \*newSolution, const int \*keep)  
*This version fixes stuff and does IP.*
- void [setSearchType](#) (int value)  
*Sets type of search.*
- int \* [used](#) () const  
*Used array so we can set.*

#### Protected Attributes

- CoinPackedMatrix [matrix\\_](#)
- int [numberSolutions\\_](#)
- int [swap\\_](#)
- int \* [used\\_](#)  
*Whether a variable has been in a solution (also when)*

#### 7.56.1 Detailed Description

LocalSearch class.

Definition at line 13 of file CbcHeuristicLocal.hpp.

#### 7.56.2 Constructor & Destructor Documentation

7.56.2.1 [CbcHeuristicLocal::CbcHeuristicLocal](#) ( )

7.56.2.2 [CbcHeuristicLocal::CbcHeuristicLocal](#) ( [CbcModel](#) & *model* )

7.56.2.3 [CbcHeuristicLocal::CbcHeuristicLocal](#) ( const [CbcHeuristicLocal](#) & )

7.56.2.4 [CbcHeuristicLocal::~~CbcHeuristicLocal](#) ( )

#### 7.56.3 Member Function Documentation

7.56.3.1 virtual [CbcHeuristic](#)\* [CbcHeuristicLocal::clone](#) ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

### 7.56.3.2 `CbcHeuristicLocal& CbcHeuristicLocal::operator= ( const CbcHeuristicLocal & rhs )`

Assignment operator.

### 7.56.3.3 `virtual void CbcHeuristicLocal::generateCpp ( FILE * fp ) [virtual]`

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

### 7.56.3.4 `virtual void CbcHeuristicLocal::resetModel ( CbcModel * model ) [virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

### 7.56.3.5 `virtual void CbcHeuristicLocal::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

### 7.56.3.6 `virtual int CbcHeuristicLocal::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts First tries setting a variable to better value. If feasible then tries setting others. If not feasible then tries swaps

This first version does not do LP's and does swaps of two integer variables. Later versions could do Lps.

Implements [CbcHeuristic](#).

### 7.56.3.7 `int CbcHeuristicLocal::solutionFix ( double & objectiveValue, double * newSolution, const int * keep )`

This version fixes stuff and does IP.

### 7.56.3.8 `void CbcHeuristicLocal::setSearchType ( int value ) [inline]`

Sets type of search.

Definition at line 65 of file `CbcHeuristicLocal.hpp`.

### 7.56.3.9 `int* CbcHeuristicLocal::used ( ) const [inline]`

Used array so we can set.

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

## 7.56.4 Member Data Documentation

### 7.56.4.1 `CoinPackedMatrix CbcHeuristicLocal::matrix_ [protected]`

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

### 7.56.4.2 `int CbcHeuristicLocal::numberSolutions_ [protected]`

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

7.56.4.3 `int CbcHeuristicLocal::swap_ [protected]`

Definition at line 82 of file CbcHeuristicLocal.hpp.

7.56.4.4 `int* CbcHeuristicLocal::used_ [protected]`

Whether a variable has been in a solution (also when)

Definition at line 84 of file CbcHeuristicLocal.hpp.

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

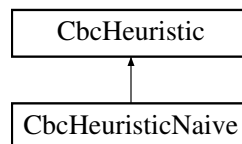
- </home/ted/COIN/trunk/Cbc/src/CbcHeuristicLocal.hpp>

## 7.57 CbcHeuristicNaive Class Reference

Naive class a) Fix all ints as close to zero as possible b) Fix all ints with nonzero costs and < large to zero c) Put bounds round continuous and UIs and maximize.

```
#include <CbcHeuristicLocal.hpp>
```

Inheritance diagram for CbcHeuristicNaive:



### Public Member Functions

- [CbcHeuristicNaive \(\)](#)
- [CbcHeuristicNaive \(CbcModel &model\)](#)
- [CbcHeuristicNaive \(const CbcHeuristicNaive &\)](#)
- [~CbcHeuristicNaive \(\)](#)
- virtual [CbcHeuristic \\* clone \(\)](#) const  
*Clone.*
- [CbcHeuristicNaive & operator= \(const CbcHeuristicNaive &rhs\)](#)  
*Assignment operator.*
- virtual void [generateCpp \(FILE \\*fp\)](#)  
*Create C++ lines to get to current state.*
- virtual void [resetModel \(CbcModel \\*model\)](#)  
*Resets stuff if model changes.*
- virtual void [setModel \(CbcModel \\*model\)](#)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution \(double &objectiveValue, double \\*newSolution\)](#)  
*returns 0 if no solution, 1 if valid solution.*
- void [setLargeValue \(double value\)](#)  
*Sets large cost value.*
- double [largeValue \(\)](#) const  
*Gets large cost value.*

## Protected Attributes

- double [large\\_](#)  
*Data Large value.*

### 7.57.1 Detailed Description

Naive class a) Fix all ints as close to zero as possible b) Fix all ints with nonzero costs and < large to zero c) Put bounds round continuous and UIs and maximize.

Definition at line 154 of file CbcHeuristicLocal.hpp.

### 7.57.2 Constructor & Destructor Documentation

7.57.2.1 `CbcHeuristicNaive::CbcHeuristicNaive ( )`

7.57.2.2 `CbcHeuristicNaive::CbcHeuristicNaive ( CbcModel & model )`

7.57.2.3 `CbcHeuristicNaive::CbcHeuristicNaive ( const CbcHeuristicNaive & )`

7.57.2.4 `CbcHeuristicNaive::~~CbcHeuristicNaive ( )`

### 7.57.3 Member Function Documentation

7.57.3.1 `virtual CbcHeuristic* CbcHeuristicNaive::clone ( ) const` [virtual]

Clone.

Implements [CbcHeuristic](#).

7.57.3.2 `CbcHeuristicNaive& CbcHeuristicNaive::operator= ( const CbcHeuristicNaive & rhs )`

Assignment operator.

7.57.3.3 `virtual void CbcHeuristicNaive::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.57.3.4 `virtual void CbcHeuristicNaive::resetModel ( CbcModel * model )` [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.57.3.5 `virtual void CbcHeuristicNaive::setModel ( CbcModel * model )` [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.57.3.6 `virtual int CbcHeuristicNaive::solution ( double & objectiveValue, double * newSolution )` [virtual]

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good)

Implements [CbcHeuristic](#).

**7.57.3.7** `void CbcHeuristicNaive::setLargeValue ( double value ) [inline]`

Sets large cost value.

Definition at line 194 of file CbcHeuristicLocal.hpp.

**7.57.3.8** `double CbcHeuristicNaive::largeValue ( ) const [inline]`

Gets large cost value.

Definition at line 198 of file CbcHeuristicLocal.hpp.

#### 7.57.4 Member Data Documentation

**7.57.4.1** `double CbcHeuristicNaive::large_ [protected]`

Data Large value.

Definition at line 205 of file CbcHeuristicLocal.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicLocal.hpp](#)

## 7.58 CbcHeuristicNode Class Reference

A class describing the branching decisions that were made to get to the node where a heuristic was invoked from.

```
#include <CbcHeuristic.hpp>
```

### Public Member Functions

- [CbcHeuristicNode](#) ([CbcModel](#) &model)
- [CbcHeuristicNode](#) (const [CbcHeuristicNode](#) &rhs)
- [~CbcHeuristicNode](#) ()
- double [distance](#) (const [CbcHeuristicNode](#) \*node) const
- double [minDistance](#) (const [CbcHeuristicNodeList](#) &nodeList) const
- bool [minDistanceIsSmall](#) (const [CbcHeuristicNodeList](#) &nodeList, const double threshold) const
- double [avgDistance](#) (const [CbcHeuristicNodeList](#) &nodeList) const

#### 7.58.1 Detailed Description

A class describing the branching decisions that were made to get to the node where a heuristic was invoked from.

Definition at line 28 of file CbcHeuristic.hpp.

#### 7.58.2 Constructor & Destructor Documentation

**7.58.2.1** `CbcHeuristicNode::CbcHeuristicNode ( CbcModel & model )`

**7.58.2.2** `CbcHeuristicNode::CbcHeuristicNode ( const CbcHeuristicNode & rhs )`

7.58.2.3 `CbcHeuristicNode::~~CbcHeuristicNode ( )`

### 7.58.3 Member Function Documentation

7.58.3.1 `double CbcHeuristicNode::distance ( const CbcHeuristicNode * node ) const`

7.58.3.2 `double CbcHeuristicNode::minDistance ( const CbcHeuristicNodeList & nodeList ) const`

7.58.3.3 `bool CbcHeuristicNode::minDistancelsSmall ( const CbcHeuristicNodeList & nodeList, const double threshold ) const`

7.58.3.4 `double CbcHeuristicNode::avgDistance ( const CbcHeuristicNodeList & nodeList ) const`

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

- </home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp>

## 7.59 CbcHeuristicNodeList Class Reference

```
#include <CbcHeuristic.hpp>
```

### Public Member Functions

- [CbcHeuristicNodeList \( \)](#)
- [CbcHeuristicNodeList \(const CbcHeuristicNodeList &rhs\)](#)
- [CbcHeuristicNodeList & operator= \(const CbcHeuristicNodeList &rhs\)](#)
- [~CbcHeuristicNodeList \( \)](#)
- `void append (CbcHeuristicNode *&node)`
- `void append (const CbcHeuristicNodeList &nodes)`
- `const CbcHeuristicNode * node (int i) const`
- `int size ( ) const`

### 7.59.1 Detailed Description

Definition at line 52 of file CbcHeuristic.hpp.

### 7.59.2 Constructor & Destructor Documentation

7.59.2.1 `CbcHeuristicNodeList::CbcHeuristicNodeList ( ) [inline]`

Definition at line 59 of file CbcHeuristic.hpp.

7.59.2.2 `CbcHeuristicNodeList::CbcHeuristicNodeList ( const CbcHeuristicNodeList & rhs )`

7.59.2.3 `CbcHeuristicNodeList::~~CbcHeuristicNodeList ( )`

### 7.59.3 Member Function Documentation

7.59.3.1 `CbcHeuristicNodeList& CbcHeuristicNodeList::operator= ( const CbcHeuristicNodeList & rhs )`

7.59.3.2 `void CbcHeuristicNodeList::append ( CbcHeuristicNode *& node )`

7.59.3.3 void CbcHeuristicNodeList::append ( const CbcHeuristicNodeList & nodes )

7.59.3.4 const CbcHeuristicNode\* CbcHeuristicNodeList::node ( int i ) const [inline]

Definition at line 66 of file CbcHeuristic.hpp.

7.59.3.5 int CbcHeuristicNodeList::size ( ) const [inline]

Definition at line 69 of file CbcHeuristic.hpp.

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

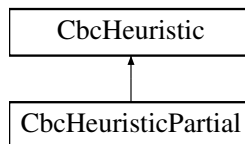
- </home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp>

## 7.60 CbcHeuristicPartial Class Reference

Partial solution class If user knows a partial solution this tries to get an integer solution it uses hotstart information.

```
#include <CbcHeuristic.hpp>
```

Inheritance diagram for CbcHeuristicPartial:



### Public Member Functions

- [CbcHeuristicPartial](#) ()
- [CbcHeuristicPartial](#) (CbcModel &model, int fixPriority=10000, int numberNodes=200)  
*Constructor with model - assumed before cuts Fixes all variables with priority <= given and does given number of nodes.*
- [CbcHeuristicPartial](#) (const [CbcHeuristicPartial](#) &)
- [~CbcHeuristicPartial](#) ()
- [CbcHeuristicPartial](#) & operator= (const [CbcHeuristicPartial](#) &rhs)  
*Assignment operator.*
- virtual [CbcHeuristic](#) \* clone () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) (CbcModel \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) (CbcModel \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts*
- virtual void [validate](#) ()  
*Validate model i.e. sets when\_ to 0 if necessary (may be NULL)*
- void [setFixPriority](#) (int value)

*Set priority level.*

- virtual bool [shouldHeurRun](#) (int [whereFrom](#))

*Check whether the heuristic should run at all.*

## Protected Attributes

- int [fixPriority\\_](#)

## 7.60.1 Detailed Description

Partial solution class If user knows a partial solution this tries to get an integer solution it uses hotstart information.

Definition at line 489 of file CbcHeuristic.hpp.

## 7.60.2 Constructor & Destructor Documentation

### 7.60.2.1 CbcHeuristicPartial::CbcHeuristicPartial ( )

### 7.60.2.2 CbcHeuristicPartial::CbcHeuristicPartial ( CbcModel & model, int fixPriority = 10000, int numberNodes = 200 )

Constructor with model - assumed before cuts Fixes all variables with priority <= given and does given number of nodes.

### 7.60.2.3 CbcHeuristicPartial::CbcHeuristicPartial ( const CbcHeuristicPartial & )

### 7.60.2.4 CbcHeuristicPartial::~~CbcHeuristicPartial ( )

## 7.60.3 Member Function Documentation

### 7.60.3.1 CbcHeuristicPartial& CbcHeuristicPartial::operator= ( const CbcHeuristicPartial & rhs )

Assignment operator.

### 7.60.3.2 virtual CbcHeuristic\* CbcHeuristicPartial::clone ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

### 7.60.3.3 virtual void CbcHeuristicPartial::generateCpp ( FILE \* fp ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

### 7.60.3.4 virtual void CbcHeuristicPartial::resetModel ( CbcModel \* model ) [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

### 7.60.3.5 virtual void CbcHeuristicPartial::setModel ( CbcModel \* model ) [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).



7.60.3.6 `virtual int CbcHeuristicPartial::solution ( double & objectiveValue, double * newSolution )` [virtual]

returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts

Implements [CbcHeuristic](#).

7.60.3.7 `virtual void CbcHeuristicPartial::validate ( )` [virtual]

Validate model i.e. sets when\_ to 0 if necessary (may be NULL)

Reimplemented from [CbcHeuristic](#).

7.60.3.8 `void CbcHeuristicPartial::setFixPriority ( int value )` [inline]

Set priority level.

Definition at line 534 of file CbcHeuristic.hpp.

7.60.3.9 `virtual bool CbcHeuristicPartial::shouldHeurRun ( int whereFrom )` [virtual]

Check whether the heuristic should run at all.

Reimplemented from [CbcHeuristic](#).

#### 7.60.4 Member Data Documentation

7.60.4.1 `int CbcHeuristicPartial::fixPriority_` [protected]

Definition at line 545 of file CbcHeuristic.hpp.

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

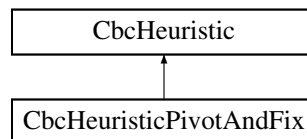
- /home/ted/COIN/trunk/Cbc/src/[CbcHeuristic.hpp](#)

## 7.61 CbcHeuristicPivotAndFix Class Reference

LocalSearch class.

```
#include <CbcHeuristicPivotAndFix.hpp>
```

Inheritance diagram for CbcHeuristicPivotAndFix:



#### Public Member Functions

- [CbcHeuristicPivotAndFix](#) ()
- [CbcHeuristicPivotAndFix](#) ([CbcModel](#) &model)
- [CbcHeuristicPivotAndFix](#) (const [CbcHeuristicPivotAndFix](#) &)
- [~CbcHeuristicPivotAndFix](#) ()

- virtual [CbcHeuristic](#) \* [clone](#) ( ) const  
*Clone.*
- [CbcHeuristicPivotAndFix](#) & [operator=](#) (const [CbcHeuristicPivotAndFix](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*

## Additional Inherited Members

### 7.61.1 Detailed Description

LocalSearch class.

Definition at line 13 of file [CbcHeuristicPivotAndFix.hpp](#).

### 7.61.2 Constructor & Destructor Documentation

7.61.2.1 [CbcHeuristicPivotAndFix::CbcHeuristicPivotAndFix](#) ( )

7.61.2.2 [CbcHeuristicPivotAndFix::CbcHeuristicPivotAndFix](#) ( [CbcModel](#) & *model* )

7.61.2.3 [CbcHeuristicPivotAndFix::CbcHeuristicPivotAndFix](#) ( const [CbcHeuristicPivotAndFix](#) & )

7.61.2.4 [CbcHeuristicPivotAndFix::~~CbcHeuristicPivotAndFix](#) ( )

### 7.61.3 Member Function Documentation

7.61.3.1 virtual [CbcHeuristic](#)\* [CbcHeuristicPivotAndFix::clone](#) ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

7.61.3.2 [CbcHeuristicPivotAndFix](#)& [CbcHeuristicPivotAndFix::operator=](#) ( const [CbcHeuristicPivotAndFix](#) & *rhs* )

Assignment operator.

7.61.3.3 virtual void [CbcHeuristicPivotAndFix::generateCpp](#) ( FILE \* *fp* ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.61.3.4 virtual void [CbcHeuristicPivotAndFix::resetModel](#) ( [CbcModel](#) \* *model* ) [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.61.3.5 `virtual void CbcHeuristicPivotAndFix::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.61.3.6 `virtual int CbcHeuristicPivotAndFix::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) needs comments

Implements [CbcHeuristic](#).

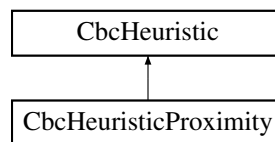
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicPivotAndFix.hpp](#)

## 7.62 CbcHeuristicProximity Class Reference

```
#include <CbcHeuristicLocal.hpp>
```

Inheritance diagram for CbcHeuristicProximity:



### Public Member Functions

- [CbcHeuristicProximity](#) ()
- [CbcHeuristicProximity](#) (CbcModel &model)
- [CbcHeuristicProximity](#) (const [CbcHeuristicProximity](#) &)
- [~CbcHeuristicProximity](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicProximity](#) & [operator=](#) (const [CbcHeuristicProximity](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) (CbcModel \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) (CbcModel \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- void [setIncrement](#) (double value)  
*Set extra increment.*
- int \* [used](#) () const  
*Used array so we can set.*

## Protected Attributes

- double [increment\\_](#)  
*Increment to use if no change.*
- [CbcHeuristicFPump](#) \* [feasibilityPump\\_](#)  
*Copy of Feasibility pump.*
- int [numberSolutions\\_](#)  
*Number of solutions so we only do after new solution.*
- int \* [used\\_](#)  
*Whether a variable has been in a solution (also when)*

### 7.62.1 Detailed Description

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

### 7.62.2 Constructor & Destructor Documentation

7.62.2.1 `CbcHeuristicProximity::CbcHeuristicProximity ( )`

7.62.2.2 `CbcHeuristicProximity::CbcHeuristicProximity ( CbcModel & model )`

7.62.2.3 `CbcHeuristicProximity::CbcHeuristicProximity ( const CbcHeuristicProximity & )`

7.62.2.4 `CbcHeuristicProximity::~~CbcHeuristicProximity ( )`

### 7.62.3 Member Function Documentation

7.62.3.1 `virtual CbcHeuristic* CbcHeuristicProximity::clone ( ) const` [virtual]

Clone.

Implements [CbcHeuristic](#).

7.62.3.2 `CbcHeuristicProximity& CbcHeuristicProximity::operator= ( const CbcHeuristicProximity & rhs )`

Assignment operator.

7.62.3.3 `virtual void CbcHeuristicProximity::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.62.3.4 `virtual void CbcHeuristicProximity::resetModel ( CbcModel * model )` [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.62.3.5 `virtual void CbcHeuristicProximity::setModel ( CbcModel * model )` [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.62.3.6 `virtual int CbcHeuristicProximity::solution ( double & objectiveValue, double * newSolution )` [virtual]

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good)

Implements [CbcHeuristic](#).

7.62.3.7 `void CbcHeuristicProximity::setIncrement ( double value )` [inline]

Set extra increment.

Definition at line 128 of file [CbcHeuristicLocal.hpp](#).

7.62.3.8 `int* CbcHeuristicProximity::used ( ) const` [inline]

Used array so we can set.

Definition at line 131 of file [CbcHeuristicLocal.hpp](#).

#### 7.62.4 Member Data Documentation

7.62.4.1 `double CbcHeuristicProximity::increment_` [protected]

Increment to use if no change.

Definition at line 138 of file [CbcHeuristicLocal.hpp](#).

7.62.4.2 `CbcHeuristicFPump* CbcHeuristicProximity::feasibilityPump_` [protected]

Copy of Feasibility pump.

Definition at line 140 of file [CbcHeuristicLocal.hpp](#).

7.62.4.3 `int CbcHeuristicProximity::numberSolutions_` [protected]

Number of solutions so we only do after new solution.

Definition at line 142 of file [CbcHeuristicLocal.hpp](#).

7.62.4.4 `int* CbcHeuristicProximity::used_` [protected]

Whether a variable has been in a solution (also when)

Definition at line 144 of file [CbcHeuristicLocal.hpp](#).

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

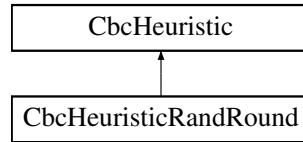
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicLocal.hpp](#)

## 7.63 CbcHeuristicRandRound Class Reference

LocalSearch class.

```
#include <CbcHeuristicRandRound.hpp>
```

Inheritance diagram for CbcHeuristicRandRound:



### Public Member Functions

- [CbcHeuristicRandRound](#) ()
- [CbcHeuristicRandRound](#) ([CbcModel](#) &model)
- [CbcHeuristicRandRound](#) (const [CbcHeuristicRandRound](#) &)
- [~CbcHeuristicRandRound](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicRandRound](#) & [operator=](#) (const [CbcHeuristicRandRound](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*

### Additional Inherited Members

#### 7.63.1 Detailed Description

LocalSearch class.

Definition at line 13 of file [CbcHeuristicRandRound.hpp](#).

#### 7.63.2 Constructor & Destructor Documentation

7.63.2.1 [CbcHeuristicRandRound::CbcHeuristicRandRound](#) ( )

7.63.2.2 [CbcHeuristicRandRound::CbcHeuristicRandRound](#) ( [CbcModel](#) & *model* )

7.63.2.3 [CbcHeuristicRandRound::CbcHeuristicRandRound](#) ( const [CbcHeuristicRandRound](#) & )

7.63.2.4 [CbcHeuristicRandRound::~~CbcHeuristicRandRound](#) ( )

#### 7.63.3 Member Function Documentation

7.63.3.1 virtual [CbcHeuristic](#)\* [CbcHeuristicRandRound::clone](#) ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

### 7.63.3.2 CbcHeuristicRandRound& CbcHeuristicRandRound::operator= ( const CbcHeuristicRandRound & rhs )

Assignment operator.

### 7.63.3.3 virtual void CbcHeuristicRandRound::generateCpp ( FILE \* fp ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

### 7.63.3.4 virtual void CbcHeuristicRandRound::resetModel ( CbcModel \* model ) [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

### 7.63.3.5 virtual void CbcHeuristicRandRound::setModel ( CbcModel \* model ) [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

### 7.63.3.6 virtual int CbcHeuristicRandRound::solution ( double & objectiveValue, double \* newSolution ) [virtual]

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) needs comments

Implements [CbcHeuristic](#).

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

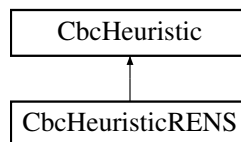
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRandRound.hpp](#)

## 7.64 CbcHeuristicRENS Class Reference

LocalSearch class.

```
#include <CbcHeuristicRENS.hpp>
```

Inheritance diagram for CbcHeuristicRENS:



### Public Member Functions

- [CbcHeuristicRENS \(\)](#)
- [CbcHeuristicRENS \(CbcModel &model\)](#)
- [CbcHeuristicRENS \(const CbcHeuristicRENS &\)](#)
- [~CbcHeuristicRENS \(\)](#)
- virtual [CbcHeuristic \\* clone \(\)](#) const  
*Clone.*
- [CbcHeuristicRENS & operator= \(const CbcHeuristicRENS &rhs\)](#)

*Assignment operator.*

- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- void [setRensType](#) (int value)  
*Set type.*

#### Protected Attributes

- int [numberTries\\_](#)  
*Number of tries.*
- int [rensType\\_](#)  
*Type 0 - fix at LB 1 - fix on dj 2 - fix at UB as well 3 - fix on 0.01\*average dj add 16 to allow two tries.*

#### 7.64.1 Detailed Description

LocalSearch class.

Definition at line 16 of file CbcHeuristicRENS.hpp.

#### 7.64.2 Constructor & Destructor Documentation

- 7.64.2.1 [CbcHeuristicRENS::CbcHeuristicRENS \( \)](#)
- 7.64.2.2 [CbcHeuristicRENS::CbcHeuristicRENS \( \[CbcModel\]\(#\) & \*model\* \)](#)
- 7.64.2.3 [CbcHeuristicRENS::CbcHeuristicRENS \( const \[CbcHeuristicRENS\]\(#\) & \)](#)
- 7.64.2.4 [CbcHeuristicRENS::~~CbcHeuristicRENS \( \)](#)

#### 7.64.3 Member Function Documentation

- 7.64.3.1 [virtual \[CbcHeuristic\]\(#\)\\* \[CbcHeuristicRENS::clone\]\(#\) \( \) const](#) [virtual]

Clone.

Implements [CbcHeuristic](#).

- 7.64.3.2 [CbcHeuristicRENS& \[CbcHeuristicRENS::operator=\]\(#\) \( const \[CbcHeuristicRENS\]\(#\) & \*rhs\* \)](#)

Assignment operator.

- 7.64.3.3 [virtual void \[CbcHeuristicRENS::resetModel\]\(#\) \( \[CbcModel\]\(#\) \\* \*model\* \)](#) [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).



7.64.3.4 `virtual void CbcHeuristicRENS::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.64.3.5 `virtual int CbcHeuristicRENS::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) This does Relaxation Extension Neighborhood Search  
Does not run if when\_ < 2 and a solution exists

Implements [CbcHeuristic](#).

7.64.3.6 `void CbcHeuristicRENS::setRensType ( int value ) [inline]`

Set type.

Definition at line 56 of file CbcHeuristicRENS.hpp.

#### 7.64.4 Member Data Documentation

7.64.4.1 `int CbcHeuristicRENS::numberTries_ [protected]`

Number of tries.

Definition at line 62 of file CbcHeuristicRENS.hpp.

7.64.4.2 `int CbcHeuristicRENS::rensType_ [protected]`

Type 0 - fix at LB 1 - fix on dj 2 - fix at UB as well 3 - fix on 0.01\*average dj add 16 to allow two tries.

Definition at line 70 of file CbcHeuristicRENS.hpp.

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

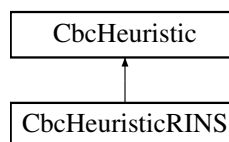
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRENS.hpp](#)

## 7.65 CbcHeuristicRINS Class Reference

LocalSearch class.

```
#include <CbcHeuristicRINS.hpp>
```

Inheritance diagram for CbcHeuristicRINS:



#### Public Member Functions

- [CbcHeuristicRINS \(\)](#)
- [CbcHeuristicRINS \(CbcModel &model\)](#)

- [CbchHeuristicRINS](#) (const [CbchHeuristicRINS](#) &)
- [~CbchHeuristicRINS](#) ()
- virtual [CbchHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbchHeuristicRINS](#) & [operator=](#) (const [CbchHeuristicRINS](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbchModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbchModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- int [solutionFix](#) (double &objectiveValue, double \*newSolution, const int \*keep)  
*This version fixes stuff and does IP.*
- void [setHowOften](#) (int value)  
*Sets how often to do it.*
- char \* [used](#) () const  
*Used array so we can set.*
- void [setLastNode](#) (int value)  
*Resets lastNode.*
- void [setSolutionCount](#) (int value)  
*Resets number of solutions.*

#### Protected Attributes

- int [numberSolutions\\_](#)  
*Number of solutions so we can do something at solution.*
- int [howOften\\_](#)  
*How often to do (code can change)*
- int [numberSuccesses\\_](#)  
*Number of successes.*
- int [numberTries\\_](#)  
*Number of tries.*
- int [stateOfFixing\\_](#)  
*State of fixing continuous variables - 0 - not tried +n - this divisor makes small enough -n - this divisor still not small enough.*
- int [lastNode\\_](#)  
*Node when last done.*
- char \* [used\\_](#)  
*Whether a variable has been in a solution.*

#### 7.65.1 Detailed Description

LocalSearch class.

Definition at line 17 of file CbchHeuristicRINS.hpp.

## 7.65.2 Constructor &amp; Destructor Documentation

7.65.2.1 CbcHeuristicRINS::CbcHeuristicRINS ( )

7.65.2.2 CbcHeuristicRINS::CbcHeuristicRINS ( CbcModel & *model* )

7.65.2.3 CbcHeuristicRINS::CbcHeuristicRINS ( const CbcHeuristicRINS &amp; )

7.65.2.4 CbcHeuristicRINS::~~CbcHeuristicRINS ( )

## 7.65.3 Member Function Documentation

7.65.3.1 virtual CbcHeuristic\* CbcHeuristicRINS::clone ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

7.65.3.2 CbcHeuristicRINS& CbcHeuristicRINS::operator= ( const CbcHeuristicRINS & *rhs* )

Assignment operator.

7.65.3.3 virtual void CbcHeuristicRINS::generateCpp ( FILE \* *fp* ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.65.3.4 virtual void CbcHeuristicRINS::resetModel ( CbcModel \* *model* ) [virtual]

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.65.3.5 virtual void CbcHeuristicRINS::setModel ( CbcModel \* *model* ) [virtual]

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.65.3.6 virtual int CbcHeuristicRINS::solution ( double & *objectiveValue*, double \* *newSolution* ) [virtual]

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) This does Relaxation Induced Neighborhood Search

Implements [CbcHeuristic](#).

7.65.3.7 int CbcHeuristicRINS::solutionFix ( double & *objectiveValue*, double \* *newSolution*, const int \* *keep* )

This version fixes stuff and does IP.

7.65.3.8 void CbcHeuristicRINS::setHowOften ( int *value* ) [inline]

Sets how often to do it.

Definition at line 63 of file CbcHeuristicRINS.hpp.

**7.65.3.9** `char* CbcHeuristicRINS::used ( ) const [inline]`

Used array so we can set.

Definition at line 67 of file CbcHeuristicRINS.hpp.

**7.65.3.10** `void CbcHeuristicRINS::setLastNode ( int value ) [inline]`

Resets lastNode.

Definition at line 71 of file CbcHeuristicRINS.hpp.

**7.65.3.11** `void CbcHeuristicRINS::setSolutionCount ( int value ) [inline]`

Resets number of solutions.

Definition at line 75 of file CbcHeuristicRINS.hpp.

#### **7.65.4 Member Data Documentation**

**7.65.4.1** `int CbcHeuristicRINS::numberSolutions_ [protected]`

Number of solutions so we can do something at solution.

Definition at line 83 of file CbcHeuristicRINS.hpp.

**7.65.4.2** `int CbcHeuristicRINS::howOften_ [protected]`

How often to do (code can change)

Definition at line 85 of file CbcHeuristicRINS.hpp.

**7.65.4.3** `int CbcHeuristicRINS::numberSuccesses_ [protected]`

Number of successes.

Definition at line 87 of file CbcHeuristicRINS.hpp.

**7.65.4.4** `int CbcHeuristicRINS::numberTries_ [protected]`

Number of tries.

Definition at line 89 of file CbcHeuristicRINS.hpp.

**7.65.4.5** `int CbcHeuristicRINS::stateOfFixing_ [protected]`

State of fixing continuous variables - 0 - not tried +n - this divisor makes small enough -n - this divisor still not small enough.

Definition at line 95 of file CbcHeuristicRINS.hpp.

**7.65.4.6** `int CbcHeuristicRINS::lastNode_ [protected]`

Node when last done.

Definition at line 97 of file CbcHeuristicRINS.hpp.

**7.65.4.7** `char* CbcHeuristicRINS::used_ [protected]`

Whether a variable has been in a solution.

Definition at line 99 of file CbcHeuristicRINS.hpp.

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

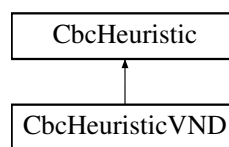
- </home/ted/COIN/trunk/Cbc/src/CbcHeuristicRINS.hpp>

## 7.66 CbcHeuristicVND Class Reference

LocalSearch class.

```
#include <CbcHeuristicVND.hpp>
```

Inheritance diagram for CbcHeuristicVND:



### Public Member Functions

- [CbcHeuristicVND](#) ()
- [CbcHeuristicVND](#) ([CbcModel](#) &model)
- [CbcHeuristicVND](#) (const [CbcHeuristicVND](#) &)
- [~CbcHeuristicVND](#) ()
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- [CbcHeuristicVND](#) & [operator=](#) (const [CbcHeuristicVND](#) &rhs)  
*Assignment operator.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- int [solutionFix](#) (double &objectiveValue, double \*newSolution, const int \*keep)  
*This version fixes stuff and does IP.*
- void [setHowOften](#) (int value)  
*Sets how often to do it.*
- double \* [baseSolution](#) () const  
*base solution array so we can set*

## Protected Attributes

- int [numberSolutions\\_](#)  
*Number of solutions so we can do something at solution.*
- int [howOften\\_](#)  
*How often to do (code can change)*
- int [numberSuccesses\\_](#)  
*Number of successes.*
- int [numberTries\\_](#)  
*Number of tries.*
- int [lastNode\\_](#)  
*Node when last done.*
- int [stepSize\\_](#)  
*Step size for decomposition.*
- int [k\\_](#)
- int [kmax\\_](#)
- int [nDifferent\\_](#)
- double \* [baseSolution\\_](#)  
*Base solution.*

### 7.66.1 Detailed Description

LocalSearch class.

Definition at line 17 of file CbcHeuristicVND.hpp.

### 7.66.2 Constructor & Destructor Documentation

7.66.2.1 `CbcHeuristicVND::CbcHeuristicVND ( )`

7.66.2.2 `CbcHeuristicVND::CbcHeuristicVND ( CbcModel & model )`

7.66.2.3 `CbcHeuristicVND::CbcHeuristicVND ( const CbcHeuristicVND & )`

7.66.2.4 `CbcHeuristicVND::~~CbcHeuristicVND ( )`

### 7.66.3 Member Function Documentation

7.66.3.1 `virtual CbcHeuristic* CbcHeuristicVND::clone ( ) const` [virtual]

Clone.

Implements [CbcHeuristic](#).

7.66.3.2 `CbcHeuristicVND& CbcHeuristicVND::operator= ( const CbcHeuristicVND & rhs )`

Assignment operator.

7.66.3.3 `virtual void CbcHeuristicVND::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.66.3.4 `virtual void CbcHeuristicVND::resetModel ( CbcModel * model ) [virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.66.3.5 `virtual void CbcHeuristicVND::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.66.3.6 `virtual int CbcHeuristicVND::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) This does Relaxation Induced Neighborhood Search

Implements [CbcHeuristic](#).

7.66.3.7 `int CbcHeuristicVND::solutionFix ( double & objectiveValue, double * newSolution, const int * keep )`

This version fixes stuff and does IP.

7.66.3.8 `void CbcHeuristicVND::setHowOften ( int value ) [inline]`

Sets how often to do it.

Definition at line 63 of file CbcHeuristicVND.hpp.

7.66.3.9 `double* CbcHeuristicVND::baseSolution ( ) const [inline]`

base solution array so we can set

Definition at line 67 of file CbcHeuristicVND.hpp.

#### 7.66.4 Member Data Documentation

7.66.4.1 `int CbcHeuristicVND::numberSolutions_ [protected]`

Number of solutions so we can do something at solution.

Definition at line 75 of file CbcHeuristicVND.hpp.

7.66.4.2 `int CbcHeuristicVND::howOften_ [protected]`

How often to do (code can change)

Definition at line 77 of file CbcHeuristicVND.hpp.

7.66.4.3 `int CbcHeuristicVND::numberSuccesses_ [protected]`

Number of successes.

Definition at line 79 of file CbcHeuristicVND.hpp.

7.66.4.4 `int CbcHeuristicVND::numberTries_ [protected]`

Number of tries.

Definition at line 81 of file CbcHeuristicVND.hpp.

#### 7.66.4.5 `int CbcHeuristicVND::lastNode_` [protected]

Node when last done.

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

#### 7.66.4.6 `int CbcHeuristicVND::stepSize_` [protected]

Step size for decomposition.

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

#### 7.66.4.7 `int CbcHeuristicVND::k_` [protected]

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

#### 7.66.4.8 `int CbcHeuristicVND::kmax_` [protected]

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

#### 7.66.4.9 `int CbcHeuristicVND::nDifferent_` [protected]

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

#### 7.66.4.10 `double* CbcHeuristicVND::baseSolution_` [protected]

Base solution.

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

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

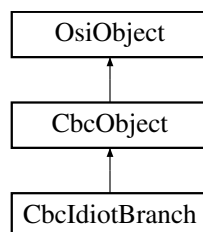
- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicVND.hpp](#)

## 7.67 CbcIdiotBranch Class Reference

Define an idiotic idea class.

```
#include <CbcFollowOn.hpp>
```

Inheritance diagram for `CbcIdiotBranch`:



### Public Member Functions

- [CbcIdiotBranch](#) ()
- [CbcIdiotBranch](#) ([CbcModel](#) \*model)
- *Useful constructor.*
- [CbcIdiotBranch](#) (const [CbcIdiotBranch](#) &)



- virtual [CbcObject](#) \* [clone](#) () const  
*Clone.*
- [CbcIdiotBranch](#) & [operator=](#) (const [CbcIdiotBranch](#) &rhs)
- [~CbcIdiotBranch](#) ()
- virtual double [infeasibility](#) (const [OsiBranchingInformation](#) \*info, int &[preferredWay](#)) const  
*Infeasibility - large is 0.5.*
- virtual void [feasibleRegion](#) ()  
*This looks at solution and sets bounds to contain solution.*
- virtual [CbcBranchingObject](#) \* [createCbcBranch](#) ([OsiSolverInterface](#) \*solver, const [OsiBranchingInformation](#) \*info, int way)  
*Creates a branching object.*
- virtual void [initializeForBranching](#) ([CbcModel](#) \*)  
*Initialize for branching.*

#### Protected Member Functions

- [OsiRowCut](#) [buildCut](#) (const [OsiBranchingInformation](#) \*info, int type, int &[preferredWay](#)) const  
*Build "cut".*

#### Protected Attributes

- [CoinThreadRandom](#) [randomNumberGenerator\\_](#)  
*data Thread specific random number generator*
- [CoinThreadRandom](#) [savedRandomNumberGenerator\\_](#)  
*Saved version of thread specific random number generator.*

#### 7.67.1 Detailed Description

Define an idiotic idea class.

The idea of this is that we take some integer variables away from integer and sum them with some randomness to get signed sum close to 0.5. We then can branch to exclude that gap.

This branching rule should be in addition to normal rules and have a high priority.

Definition at line 161 of file [CbcFollowOn.hpp](#).

#### 7.67.2 Constructor & Destructor Documentation

##### 7.67.2.1 [CbcIdiotBranch::CbcIdiotBranch](#) ( )

##### 7.67.2.2 [CbcIdiotBranch::CbcIdiotBranch](#) ( [CbcModel](#) \* *model* )

Useful constructor.

##### 7.67.2.3 [CbcIdiotBranch::CbcIdiotBranch](#) ( const [CbcIdiotBranch](#) & )

##### 7.67.2.4 [CbcIdiotBranch::~CbcIdiotBranch](#) ( )

#### 7.67.3 Member Function Documentation

**7.67.3.1** `virtual CbcObject* CbcldiotBranch::clone ( ) const` `[virtual]`

Clone.

Implements [CbcObject](#).

**7.67.3.2** `CbcldiotBranch& CbcldiotBranch::operator= ( const CbcldiotBranch & rhs )`

**7.67.3.3** `virtual double CbcldiotBranch::infeasibility ( const OsiBranchingInformation * info, int & preferredWay ) const`  
`[virtual]`

Infeasibility - large is 0.5.

Reimplemented from [CbcObject](#).

**7.67.3.4** `virtual void CbcldiotBranch::feasibleRegion ( )` `[virtual]`

This looks at solution and sets bounds to contain solution.

Implements [CbcObject](#).

**7.67.3.5** `virtual CbcBranchingObject* CbcldiotBranch::createCbcBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way )` `[virtual]`

Creates a branching object.

Reimplemented from [CbcObject](#).

**7.67.3.6** `virtual void CbcldiotBranch::initializeForBranching ( CbcModel * )` `[virtual]`

Initialize for branching.

Reimplemented from [CbcObject](#).

**7.67.3.7** `OsiRowCut CbcldiotBranch::buildCut ( const OsiBranchingInformation * info, int type, int & preferredWay ) const`  
`[protected]`

Build "cut".

## **7.67.4 Member Data Documentation**

**7.67.4.1** `CoinThreadRandom CbcldiotBranch::randomNumberGenerator_` `[mutable], [protected]`

data Thread specific random number generator

Definition at line 201 of file CbcFollowOn.hpp.

**7.67.4.2** `CoinThreadRandom CbcldiotBranch::savedRandomNumberGenerator_` `[mutable], [protected]`

Saved version of thread specific random number generator.

Definition at line 203 of file CbcFollowOn.hpp.

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

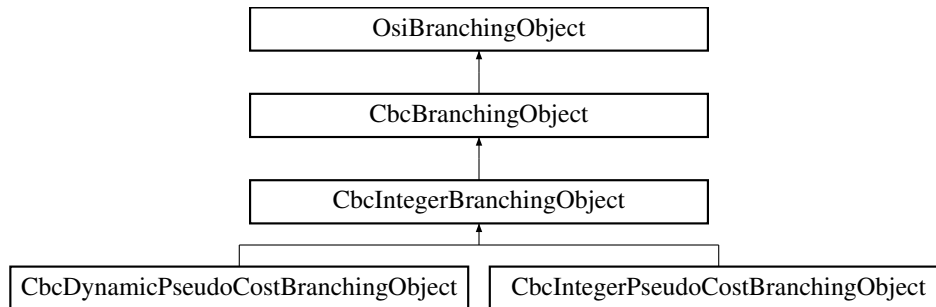
- [/home/ted/COIN/trunk/Cbc/src/CbcFollowOn.hpp](#)

## 7.68 CbcIntegerBranchingObject Class Reference

Simple branching object for an integer variable.

```
#include <CbcSimpleInteger.hpp>
```

Inheritance diagram for CbcIntegerBranchingObject:



### Public Member Functions

- [CbcIntegerBranchingObject](#) ()  
*Default constructor.*
- [CbcIntegerBranchingObject](#) ([CbcModel](#) \*model, int variable, int way, double value)  
*Create a standard floor/ceiling branch object.*
- [CbcIntegerBranchingObject](#) ([CbcModel](#) \*model, int variable, int way, double lowerValue, double upperValue)  
*Create a degenerate branch object.*
- [CbcIntegerBranchingObject](#) (const [CbcIntegerBranchingObject](#) &)  
*Copy constructor.*
- [CbcIntegerBranchingObject](#) & operator= (const [CbcIntegerBranchingObject](#) &rhs)  
*Assignment operator.*
- virtual [CbcBranchingObject](#) \* clone () const  
*Clone.*
- virtual ~[CbcIntegerBranchingObject](#) ()  
*Destructor.*
- void fillPart (int variable, int way, double value)  
*Does part of constructor.*
- virtual double branch ()  
*Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.*
- virtual void fix ([OsiSolverInterface](#) \*solver, double \*lower, double \*upper, int branchState) const  
*Update bounds in solver as in 'branch' and update given bounds.*
- virtual bool tighten ([OsiSolverInterface](#) \*)  
*Change (tighten) bounds in object to reflect bounds in solver.*
- virtual void print ()  
*Print something about branch - only if log level high.*
- const double \* downBounds () const  
*Lower and upper bounds for down branch.*
- const double \* upBounds () const  
*Lower and upper bounds for up branch.*
- void setDownBounds (const double bounds[2])

- Set lower and upper bounds for down branch.*

  - void `setUpBounds` (const double bounds[2])

*Set lower and upper bounds for up branch.*

  - virtual `CbcBranchObjType type` () const

*Return the type (an integer identifier) of `this`.*

  - virtual `CbcRangeCompare compareBranchingObject` (const `CbcBranchingObject` \*brObj, const bool replaceIfOverlap=false)

*Compare the `this` with `brObj`.*

#### Protected Attributes

- double `down_` [2]
- Lower [0] and upper [1] bounds for the down arm (way\_ = -1)*
- double `up_` [2]
- Lower [0] and upper [1] bounds for the up arm (way\_ = 1)*

#### 7.68.1 Detailed Description

Simple branching object for an integer variable.

This object can specify a two-way branch on an integer variable. For each arm of the branch, the upper and lower bounds on the variable can be independently specified.

`Variable_` holds the index of the integer variable in the `integerVariable_` array of the model.

Definition at line 23 of file `CbcSimpleInteger.hpp`.

#### 7.68.2 Constructor & Destructor Documentation

##### 7.68.2.1 `CbcIntegerBranchingObject::CbcIntegerBranchingObject ( )`

Default constructor.

##### 7.68.2.2 `CbcIntegerBranchingObject::CbcIntegerBranchingObject ( CbcModel * model, int variable, int way, double value )`

Create a standard floor/ceiling branch object.

Specifies a simple two-way branch. Let `value = x*`. One arm of the branch will be `lb <= x <= floor(x*)`, the other `ceil(x*) <= x <= ub`. Specify `way = -1` to set the object state to perform the down arm first, `way = 1` for the up arm.

##### 7.68.2.3 `CbcIntegerBranchingObject::CbcIntegerBranchingObject ( CbcModel * model, int variable, int way, double lowerValue, double upperValue )`

Create a degenerate branch object.

Specifies a 'one-way branch'. Calling `branch()` for this object will always result in `lowerValue <= x <= upperValue`. Used to fix a variable when `lowerValue = upperValue`.

##### 7.68.2.4 `CbcIntegerBranchingObject::CbcIntegerBranchingObject ( const CbcIntegerBranchingObject & )`

Copy constructor.

##### 7.68.2.5 `virtual CbcIntegerBranchingObject::~~CbcIntegerBranchingObject ( )` [virtual]

Destructor.

## 7.68.3 Member Function Documentation

**7.68.3.1** `CbcIntegerBranchingObject& CbcIntegerBranchingObject::operator= ( const CbcIntegerBranchingObject & rhs )`

Assignment operator.

**7.68.3.2** `virtual CbcBranchingObject* CbcIntegerBranchingObject::clone ( ) const` `[virtual]`

Clone.

Implements [CbcBranchingObject](#).

Reimplemented in [CbcIntegerPseudoCostBranchingObject](#), and [CbcDynamicPseudoCostBranchingObject](#).

**7.68.3.3** `void CbcIntegerBranchingObject::fillPart ( int variable, int way, double value )`

Does part of constructor.

**7.68.3.4** `virtual double CbcIntegerBranchingObject::branch ( )` `[virtual]`

Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.

Returns change in guessed objective on next branch

Implements [CbcBranchingObject](#).

Reimplemented in [CbcIntegerPseudoCostBranchingObject](#), and [CbcDynamicPseudoCostBranchingObject](#).

**7.68.3.5** `virtual void CbcIntegerBranchingObject::fix ( OsiSolverInterface * solver, double * lower, double * upper, int branchState ) const` `[virtual]`

Update bounds in solver as in 'branch' and update given bounds.

branchState is -1 for 'down' +1 for 'up'

Reimplemented from [CbcBranchingObject](#).

**7.68.3.6** `virtual bool CbcIntegerBranchingObject::tighten ( OsiSolverInterface * )` `[virtual]`

Change (tighten) bounds in object to reflect bounds in solver.

Return true if now fixed

Reimplemented from [CbcBranchingObject](#).

**7.68.3.7** `virtual void CbcIntegerBranchingObject::print ( )` `[virtual]`

Print something about branch - only if log level high.

**7.68.3.8** `const double* CbcIntegerBranchingObject::downBounds ( ) const` `[inline]`

Lower and upper bounds for down branch.

Definition at line 93 of file CbcSimpleInteger.hpp.

**7.68.3.9** `const double* CbcIntegerBranchingObject::upBounds ( ) const` `[inline]`

Lower and upper bounds for up branch.

Definition at line 97 of file CbcSimpleInteger.hpp.

**7.68.3.10** `void CbcIntegerBranchingObject::setDownBounds ( const double bounds[2] )` `[inline]`

Set lower and upper bounds for down branch.

Definition at line 101 of file CbcSimpleInteger.hpp.

**7.68.3.11** `void CbcIntegerBranchingObject::setUpBounds ( const double bounds[2] )` `[inline]`

Set lower and upper bounds for up branch.

Definition at line 105 of file CbcSimpleInteger.hpp.

**7.68.3.12** `virtual CbcBranchObjType CbcIntegerBranchingObject::type ( ) const` `[inline],[virtual]`

Return the type (an integer identifier) of `this`.

Implements [CbcBranchingObject](#).

Reimplemented in [CbcIntegerPseudoCostBranchingObject](#), and [CbcDynamicPseudoCostBranchingObject](#).

Definition at line 133 of file CbcSimpleInteger.hpp.

**7.68.3.13** `virtual CbcRangeCompare CbcIntegerBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false )` `[virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

Reimplemented in [CbcIntegerPseudoCostBranchingObject](#).

## 7.68.4 Member Data Documentation

**7.68.4.1** `double CbcIntegerBranchingObject::down_[2]` `[protected]`

Lower [0] and upper [1] bounds for the down arm (`way_ = -1`)

Definition at line 150 of file CbcSimpleInteger.hpp.

**7.68.4.2** `double CbcIntegerBranchingObject::up_[2]` `[protected]`

Lower [0] and upper [1] bounds for the up arm (`way_ = 1`)

Definition at line 152 of file CbcSimpleInteger.hpp.

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

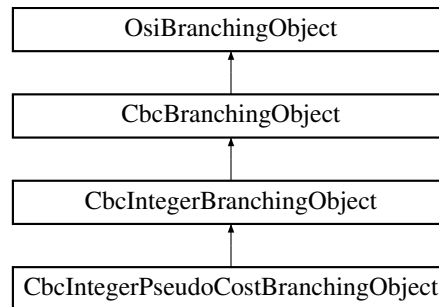
- [/home/ted/COIN/trunk/Cbc/src/CbcSimpleInteger.hpp](#)

## 7.69 CbcIntegerPseudoCostBranchingObject Class Reference

Simple branching object for an integer variable with pseudo costs.

```
#include <CbcSimpleIntegerDynamicPseudoCost.hpp>
```

Inheritance diagram for `CbcIntegerPseudoCostBranchingObject`:



### Public Member Functions

- [CbcIntegerPseudoCostBranchingObject](#) ()  
*Default constructor.*
- [CbcIntegerPseudoCostBranchingObject](#) ([CbcModel](#) \**model*, int *variable*, int *way*, double *value*)  
*Create a standard floor/ceiling branch object.*
- [CbcIntegerPseudoCostBranchingObject](#) ([CbcModel](#) \**model*, int *variable*, int *way*, double *lowerValue*, double *upperValue*)  
*Create a degenerate branch object.*
- [CbcIntegerPseudoCostBranchingObject](#) (const [CbcIntegerPseudoCostBranchingObject](#) &)  
*Copy constructor.*
- [CbcIntegerPseudoCostBranchingObject](#) & *operator=* (const [CbcIntegerPseudoCostBranchingObject](#) &*rhs*)  
*Assignment operator.*
- virtual [CbcBranchingObject](#) \* *clone* () const  
*Clone.*
- virtual ~[CbcIntegerPseudoCostBranchingObject](#) ()  
*Destructor.*
- virtual double *branch* ()  
*Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.*
- double *changeInGuessed* () const  
*Change in guessed.*
- void *setChangeInGuessed* (double *value*)  
*Set change in guessed.*
- virtual [CbcBranchObjType](#) *type* () const  
*Return the type (an integer identifier) of this.*
- virtual [CbcRangeCompare](#) *compareBranchingObject* (const [CbcBranchingObject](#) \**brObj*, const bool *replacelf-Overlap=false*)  
*Compare the this with brObj.*

### Protected Attributes

- double *changeInGuessed\_*  
*Change in guessed objective value for next branch.*

### 7.69.1 Detailed Description

Simple branching object for an integer variable with pseudo costs.

This object can specify a two-way branch on an integer variable. For each arm of the branch, the upper and lower bounds on the variable can be independently specified.

Variable\_ holds the index of the integer variable in the integerVariable\_ array of the model.

Definition at line 389 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

### 7.69.2 Constructor & Destructor Documentation

#### 7.69.2.1 CbcIntegerPseudoCostBranchingObject::CbcIntegerPseudoCostBranchingObject ( )

Default constructor.

#### 7.69.2.2 CbcIntegerPseudoCostBranchingObject::CbcIntegerPseudoCostBranchingObject ( CbcModel \* model, int variable, int way, double value )

Create a standard floor/ceiling branch object.

Specifies a simple two-way branch. Let `value = x*`. One arm of the branch will be  $lb \leq x \leq \text{floor}(x^*)$ , the other  $\text{ceil}(x^*) \leq x \leq ub$ . Specify `way = -1` to set the object state to perform the down arm first, `way = 1` for the up arm.

#### 7.69.2.3 CbcIntegerPseudoCostBranchingObject::CbcIntegerPseudoCostBranchingObject ( CbcModel \* model, int variable, int way, double lowerValue, double upperValue )

Create a degenerate branch object.

Specifies a 'one-way branch'. Calling [branch\(\)](#) for this object will always result in  $\text{lowerValue} \leq x \leq \text{upperValue}$ . Used to fix a variable when  $\text{lowerValue} = \text{upperValue}$ .

#### 7.69.2.4 CbcIntegerPseudoCostBranchingObject::CbcIntegerPseudoCostBranchingObject ( const CbcIntegerPseudoCostBranchingObject & )

Copy constructor.

#### 7.69.2.5 virtual CbcIntegerPseudoCostBranchingObject::~~CbcIntegerPseudoCostBranchingObject ( ) [virtual]

Destructor.

### 7.69.3 Member Function Documentation

#### 7.69.3.1 CbcIntegerPseudoCostBranchingObject& CbcIntegerPseudoCostBranchingObject::operator= ( const CbcIntegerPseudoCostBranchingObject & rhs )

Assignment operator.

#### 7.69.3.2 virtual CbcBranchingObject\* CbcIntegerPseudoCostBranchingObject::clone ( ) const [virtual]

Clone.

Reimplemented from [CbcIntegerBranchingObject](#).



**7.69.3.3** `virtual double CbcIntegerPseudoCostBranchingObject::branch ( ) [virtual]`

Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.

This version also changes guessed objective value

Reimplemented from [CbcIntegerBranchingObject](#).

**7.69.3.4** `double CbcIntegerPseudoCostBranchingObject::changeInGuessed ( ) const [inline]`

Change in guessed.

Definition at line 436 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.69.3.5** `void CbcIntegerPseudoCostBranchingObject::setChangeInGuessed ( double value ) [inline]`

Set change in guessed.

Definition at line 440 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.69.3.6** `virtual CbcBranchObjType CbcIntegerPseudoCostBranchingObject::type ( ) const [inline],[virtual]`

Return the type (an integer identifier) of `this`.

Reimplemented from [CbcIntegerBranchingObject](#).

Definition at line 445 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.69.3.7** `virtual CbcRangeCompare CbcIntegerPseudoCostBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false ) [virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Reimplemented from [CbcIntegerBranchingObject](#).

**7.69.4 Member Data Documentation****7.69.4.1** `double CbcIntegerPseudoCostBranchingObject::changeInGuessed_ [protected]`

Change in guessed objective value for next branch.

Definition at line 462 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

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

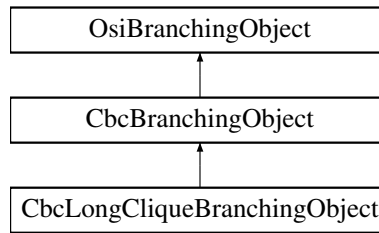
- `/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerDynamicPseudoCost.hpp`

**7.70 CbcLongCliqueBranchingObject Class Reference**

Unordered Clique Branching Object class.

```
#include <CbcClique.hpp>
```

Inheritance diagram for `CbcLongCliqueBranchingObject`:



### Public Member Functions

- [CbcLongCliqueBranchingObject](#) ()
- [CbcLongCliqueBranchingObject](#) ([CbcModel](#) \*[model](#), const [CbcClique](#) \*[clique](#), int [way](#), int [numberOnDownSide](#), const int \*[down](#), int [numberOnUpSide](#), const int \*[up](#))
- [CbcLongCliqueBranchingObject](#) (const [CbcLongCliqueBranchingObject](#) &)
- [CbcLongCliqueBranchingObject](#) & [operator=](#) (const [CbcLongCliqueBranchingObject](#) &[rhs](#))
- virtual [CbcBranchingObject](#) \* [clone](#) () const  
*Clone.*
- virtual [~CbcLongCliqueBranchingObject](#) ()
- virtual double [branch](#) ()  
*Does next branch and updates state.*
- virtual void [print](#) ()  
*Print something about branch - only if log level high.*
- virtual [CbcBranchObjType](#) [type](#) () const  
*Return the type (an integer identifier) of this.*
- virtual int [compareOriginalObject](#) (const [CbcBranchingObject](#) \*[brObj](#)) const  
*Compare the original object of this with the original object of brObj.*
- virtual [CbcRangeCompare](#) [compareBranchingObject](#) (const [CbcBranchingObject](#) \*[brObj](#), const bool [replacelfOverlap=false](#))  
*Compare the this with brObj.*

### Additional Inherited Members

#### 7.70.1 Detailed Description

Unordered Clique Branching Object class.

These are for cliques which are > 64 members Variable is number of clique.

Definition at line 234 of file [CbcClique.hpp](#).

#### 7.70.2 Constructor & Destructor Documentation

7.70.2.1 [CbcLongCliqueBranchingObject::CbcLongCliqueBranchingObject](#) ( )

7.70.2.2 [CbcLongCliqueBranchingObject::CbcLongCliqueBranchingObject](#) ( [CbcModel](#) \* [model](#), const [CbcClique](#) \* [clique](#), int [way](#), int [numberOnDownSide](#), const int \* [down](#), int [numberOnUpSide](#), const int \* [up](#) )

7.70.2.3 [CbcLongCliqueBranchingObject::CbcLongCliqueBranchingObject](#) ( const [CbcLongCliqueBranchingObject](#) & )

7.70.2.4 virtual [CbcLongCliqueBranchingObject::~CbcLongCliqueBranchingObject](#) ( ) [virtual]

## 7.70.3 Member Function Documentation

7.70.3.1 **CbcLongCliqueBranchingObject& CbcLongCliqueBranchingObject::operator= ( const CbcLongCliqueBranchingObject & rhs )**

7.70.3.2 **virtual CbcBranchingObject\* CbcLongCliqueBranchingObject::clone ( ) const** [virtual]

Clone.

Implements [CbcBranchingObject](#).

7.70.3.3 **virtual double CbcLongCliqueBranchingObject::branch ( )** [virtual]

Does next branch and updates state.

Implements [CbcBranchingObject](#).

7.70.3.4 **virtual void CbcLongCliqueBranchingObject::print ( )** [virtual]

Print something about branch - only if log level high.

7.70.3.5 **virtual CbcBranchObjType CbcLongCliqueBranchingObject::type ( ) const** [inline],[virtual]

Return the type (an integer identifier) of `this`.

Implements [CbcBranchingObject](#).

Definition at line 269 of file `CbcClique.hpp`.

7.70.3.6 **virtual int CbcLongCliqueBranchingObject::compareOriginalObject ( const CbcBranchingObject \* brObj ) const** [virtual]

Compare the original object of `this` with the original object of `brObj`.

Assumes that there is an ordering of the original objects. This method should be invoked only if `this` and `brObj` are of the same type. Return negative/0/positive depending on whether `this` is smaller/same/larger than the argument.

Reimplemented from [CbcBranchingObject](#).

7.70.3.7 **virtual CbcRangeCompare CbcLongCliqueBranchingObject::compareBranchingObject ( const CbcBranchingObject \* brObj, const bool replaceIfOverlap = false )** [virtual]

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

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

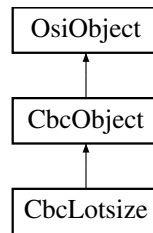
- `/home/ted/COIN/trunk/Cbc/src/CbcClique.hpp`

## 7.71 CbcLotsize Class Reference

Lotsize class.

```
#include <CbcBranchLotsize.hpp>
```

Inheritance diagram for CbcLotsize:



### Public Member Functions

- `CbcLotsize ()`
- `CbcLotsize (CbcModel *model, int iColumn, int numberPoints, const double *points, bool range=false)`
- `CbcLotsize (const CbcLotsize &)`
- virtual `CbcObject * clone () const`  
*Clone.*
- `CbcLotsize & operator= (const CbcLotsize &rhs)`
- `~CbcLotsize ()`
- virtual double `infeasibility (const OsiBranchingInformation *info, int &preferredWay) const`  
*Infeasibility - large is 0.5.*
- virtual void `feasibleRegion ()`  
*Set bounds to contain the current solution.*
- virtual `CbcBranchingObject * createCbcBranch (OsiSolverInterface *solver, const OsiBranchingInformation *info, int way)`  
*Creates a branching object.*
- virtual `CbcBranchingObject * preferredNewFeasible () const`  
*Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in the good direction.*
- virtual `CbcBranchingObject * notPreferredNewFeasible () const`  
*Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a bad direction.*
- virtual void `resetBounds (const OsiSolverInterface *solver)`  
*Reset original upper and lower bound values from the solver.*
- bool `findRange (double value) const`  
*Finds range of interest so value is feasible in range range\_ or infeasible between hi[range\_] and lo[range\_+1].*
- virtual void `floorCeiling (double &floorLotsize, double &ceilingLotsize, double value, double tolerance) const`  
*Returns floor and ceiling.*
- int `modelSequence () const`  
*Model column number.*
- void `setModelSequence (int value)`  
*Set model column number.*
- virtual int `columnNumber () const`  
*Column number if single column object -1 otherwise, so returns >= 0 Used by heuristics.*
- double `originalLowerBound () const`  
*Original variable bounds.*
- double `originalUpperBound () const`
- int `rangeType () const`

*Type - 1 points, 2 ranges.*

- int [numberRanges](#) () const

*Number of points.*

- double \* [bound](#) () const

*Ranges.*

- virtual bool [canDoHeuristics](#) () const

*Return true if object can take part in normal heuristics.*

## Additional Inherited Members

### 7.71.1 Detailed Description

Lotsize class.

Definition at line 13 of file CbcBranchLotsize.hpp.

### 7.71.2 Constructor & Destructor Documentation

#### 7.71.2.1 CbcLotsize::CbcLotsize ( )

#### 7.71.2.2 CbcLotsize::CbcLotsize ( CbcModel \* model, int iColumn, int numberPoints, const double \* points, bool range = false )

#### 7.71.2.3 CbcLotsize::CbcLotsize ( const CbcLotsize & )

#### 7.71.2.4 CbcLotsize::~~CbcLotsize ( )

### 7.71.3 Member Function Documentation

#### 7.71.3.1 virtual CbcObject\* CbcLotsize::clone ( ) const [virtual]

Clone.

Implements [CbcObject](#).

#### 7.71.3.2 CbcLotsize& CbcLotsize::operator= ( const CbcLotsize & rhs )

#### 7.71.3.3 virtual double CbcLotsize::infeasibility ( const OsiBranchingInformation \* info, int & preferredWay ) const [virtual]

Infeasibility - large is 0.5.

Reimplemented from [CbcObject](#).

#### 7.71.3.4 virtual void CbcLotsize::feasibleRegion ( ) [virtual]

Set bounds to contain the current solution.

More precisely, for the variable associated with this object, take the value given in the current solution, force it within the current bounds if required, then set the bounds to fix the variable at the integer nearest the solution value.

Implements [CbcObject](#).

#### 7.71.3.5 virtual CbcBranchingObject\* CbcLotsize::createCbcBranch ( OsiSolverInterface \* solver, const OsiBranchingInformation \* info, int way ) [virtual]

Creates a branching object.

Reimplemented from [CbcObject](#).

**7.71.3.6** `virtual CbcBranchingObject* CbcLotsize::preferredNewFeasible ( ) const [virtual]`

Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in the good direction.

The preferred branching object will force the variable to be +/-1 from its current value, depending on the reduced cost and objective sense. If movement in the direction which improves the objective is impossible due to bounds on the variable, the branching object will move in the other direction. If no movement is possible, the method returns NULL.

Only the bounds on this variable are considered when determining if the new point is feasible.

Reimplemented from [CbcObject](#).

**7.71.3.7** `virtual CbcBranchingObject* CbcLotsize::notPreferredNewFeasible ( ) const [virtual]`

Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a bad direction.

As for [preferredNewFeasible\(\)](#), but the preferred branching object will force movement in a direction that degrades the objective.

Reimplemented from [CbcObject](#).

**7.71.3.8** `virtual void CbcLotsize::resetBounds ( const OsiSolverInterface * solver ) [virtual]`

Reset original upper and lower bound values from the solver.

Handy for updating bounds held in this object after bounds held in the solver have been tightened.

Reimplemented from [CbcObject](#).

**7.71.3.9** `bool CbcLotsize::findRange ( double value ) const`

Finds range of interest so value is feasible in range range\_ or infeasible between hi[range\_] and lo[range\_+1].

Returns true if feasible.

**7.71.3.10** `virtual void CbcLotsize::floorCeiling ( double & floorLotsize, double & ceilingLotsize, double value, double tolerance ) const [virtual]`

Returns floor and ceiling.

Reimplemented from [CbcObject](#).

**7.71.3.11** `int CbcLotsize::modelSequence ( ) const [inline]`

Model column number.

Definition at line 97 of file CbcBranchLotsize.hpp.

**7.71.3.12** `void CbcLotsize::setModelSequence ( int value ) [inline]`

Set model column number.

Definition at line 101 of file CbcBranchLotsize.hpp.

**7.71.3.13** `virtual int CbcLotsize::columnNumber ( ) const [virtual]`

Column number if single column object -1 otherwise, so returns  $\geq 0$  Used by heuristics.

7.71.3.14 `double CbcLotsize::originalLowerBound ( ) const [inline]`

Original variable bounds.

Definition at line 111 of file CbcBranchLotsize.hpp.

7.71.3.15 `double CbcLotsize::originalUpperBound ( ) const [inline]`

Definition at line 114 of file CbcBranchLotsize.hpp.

7.71.3.16 `int CbcLotsize::rangeType ( ) const [inline]`

Type - 1 points, 2 ranges.

Definition at line 118 of file CbcBranchLotsize.hpp.

7.71.3.17 `int CbcLotsize::numberRanges ( ) const [inline]`

Number of points.

Definition at line 122 of file CbcBranchLotsize.hpp.

7.71.3.18 `double* CbcLotsize::bound ( ) const [inline]`

Ranges.

Definition at line 126 of file CbcBranchLotsize.hpp.

7.71.3.19 `virtual bool CbcLotsize::canDoHeuristics ( ) const [inline], [virtual]`

Return true if object can take part in normal heuristics.

Definition at line 131 of file CbcBranchLotsize.hpp.

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

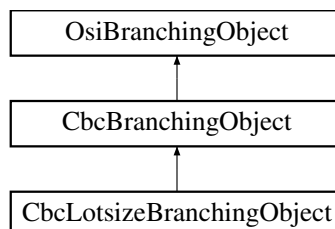
- [/home/ted/COIN/trunk/Cbc/src/CbcBranchLotsize.hpp](#)

## 7.72 CbcLotsizeBranchingObject Class Reference

Lotsize branching object.

```
#include <CbcBranchLotsize.hpp>
```

Inheritance diagram for CbcLotsizeBranchingObject:



### Public Member Functions

- [CbcLotsizeBranchingObject \( \)](#)

*Default constructor.*

- `CbcLotsizeBranchingObject` (`CbcModel *model`, `int variable`, `int way`, `double value`, `const CbcLotsize *lotsize`)

*Create a lotsize floor/ceiling branch object.*

- `CbcLotsizeBranchingObject` (`CbcModel *model`, `int variable`, `int way`, `double lowerValue`, `double upperValue`)

*Create a degenerate branch object.*

- `CbcLotsizeBranchingObject` (`const CbcLotsizeBranchingObject &`)

*Copy constructor.*

- `CbcLotsizeBranchingObject & operator=` (`const CbcLotsizeBranchingObject &rhs`)

*Assignment operator.*

- `virtual CbcBranchingObject * clone` () `const`

*Clone.*

- `virtual ~CbcLotsizeBranchingObject` ()

*Destructor.*

- `virtual double branch` ()

*Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.*

- `virtual void print` ()

*Print something about branch - only if log level high.*

- `virtual CbcBranchObjType type` () `const`

*Return the type (an integer identifier) of this.*

- `virtual CbcRangeCompare compareBranchingObject` (`const CbcBranchingObject *brObj`, `const bool replaceIfOverlap=false`)

*Compare the this with brObj.*

## Protected Attributes

- `double down_` [2]

*Lower [0] and upper [1] bounds for the down arm (way\_ = -1)*

- `double up_` [2]

*Lower [0] and upper [1] bounds for the up arm (way\_ = 1)*

## 7.72.1 Detailed Description

Lotsize branching object.

This object can specify a two-way branch on an integer variable. For each arm of the branch, the upper and lower bounds on the variable can be independently specified.

`Variable_` holds the index of the integer variable in the `integerVariable_` array of the model.

Definition at line 166 of file `CbcBranchLotsize.hpp`.

## 7.72.2 Constructor & Destructor Documentation

### 7.72.2.1 `CbcLotsizeBranchingObject::CbcLotsizeBranchingObject` ( )

Default constructor.



**7.72.2.2 CbcLotsizeBranchingObject::CbcLotsizeBranchingObject ( CbcModel \* *model*, int *variable*, int *way*, double *value*, const CbcLotsize \* *lotsize* )**

Create a lotsize floor/ceiling branch object.

Specifies a simple two-way branch. Let *value* = *x*\*. One arm of the branch will be  $lb \leq x \leq \text{valid range below}(x^*)$ , the other valid range  $\text{above}(x^*) \leq x \leq ub$ . Specify *way* = -1 to set the object state to perform the down arm first, *way* = 1 for the up arm.

**7.72.2.3 CbcLotsizeBranchingObject::CbcLotsizeBranchingObject ( CbcModel \* *model*, int *variable*, int *way*, double *lowerValue*, double *upperValue* )**

Create a degenerate branch object.

Specifies a 'one-way branch'. Calling [branch\(\)](#) for this object will always result in  $\text{lowerValue} \leq x \leq \text{upperValue}$ . Used to fix in valid range

**7.72.2.4 CbcLotsizeBranchingObject::CbcLotsizeBranchingObject ( const CbcLotsizeBranchingObject & )**

Copy constructor.

**7.72.2.5 virtual CbcLotsizeBranchingObject::~CbcLotsizeBranchingObject ( ) [virtual]**

Destructor.

### 7.72.3 Member Function Documentation

**7.72.3.1 CbcLotsizeBranchingObject& CbcLotsizeBranchingObject::operator= ( const CbcLotsizeBranchingObject & *rhs* )**

Assignment operator.

**7.72.3.2 virtual CbcBranchingObject\* CbcLotsizeBranchingObject::clone ( ) const [virtual]**

Clone.

Implements [CbcBranchingObject](#).

**7.72.3.3 virtual double CbcLotsizeBranchingObject::branch ( ) [virtual]**

Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.

Implements [CbcBranchingObject](#).

**7.72.3.4 virtual void CbcLotsizeBranchingObject::print ( ) [virtual]**

Print something about branch - only if log level high.

**7.72.3.5 virtual CbcBranchObjType CbcLotsizeBranchingObject::type ( ) const [inline], [virtual]**

Return the type (an integer identifier) of *this*.

Implements [CbcBranchingObject](#).

Definition at line 216 of file CbcBranchLotsize.hpp.

**7.72.3.6** `virtual CbcRangeCompare CbcLotsizeBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false ) [virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

#### 7.72.4 Member Data Documentation

**7.72.4.1** `double CbcLotsizeBranchingObject::down_[2] [protected]`

Lower [0] and upper [1] bounds for the down arm (`way_ = -1`)

Definition at line 236 of file `CbcBranchLotsize.hpp`.

**7.72.4.2** `double CbcLotsizeBranchingObject::up_[2] [protected]`

Lower [0] and upper [1] bounds for the up arm (`way_ = 1`)

Definition at line 238 of file `CbcBranchLotsize.hpp`.

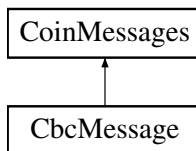
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcBranchLotsize.hpp](#)

### 7.73 CbcMessage Class Reference

```
#include <CbcMessage.hpp>
```

Inheritance diagram for `CbcMessage`:



#### Public Member Functions

##### Constructors etc

- [CbcMessage](#) (Language language=us\_en)  
*Constructor.*

#### 7.73.1 Detailed Description

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

## 7.73.2 Constructor &amp; Destructor Documentation

7.73.2.1 CbcMessage::CbcMessage ( Language *language* = us\_en )

Constructor.

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

- /home/ted/COIN/trunk/Cbc/src/CbcMessage.hpp

## 7.74 CbcModel Class Reference

Simple Branch and bound class.

```
#include <CbcModel.hpp>
```

## Public Types

- enum CbcIntParam {  
CbcMaxNumNode = 0, CbcMaxNumSol, CbcFathomDiscipline, CbcPrinting,  
CbcNumberBranches, CbcLastIntParam }
- enum CbcDblParam {  
CbcIntegerTolerance = 0, CbcInfeasibilityWeight, CbcCutoffIncrement, CbcAllowableGap,  
CbcAllowableFractionGap, CbcMaximumSeconds, CbcCurrentCutoff, CbcOptimizationDirection,  
CbcCurrentObjectiveValue, CbcCurrentMinimizationObjectiveValue, CbcStartSeconds, CbcHeuristicGap,  
CbcHeuristicFractionGap, CbcSmallestChange, CbcSumChange, CbcLargestChange,  
CbcSmallChange, CbcLastDblParam }

## Public Member Functions

- void [setMIPStart](#) (const std::vector< std::pair< std::string, double > > &mips)
- const std::vector< std::pair< std::string, double > > &[getMIPStart](#) ()

## Presolve methods

- [CbcModel \\* findCliques](#) (bool makeEquality, int atLeastThisMany, int lessThanThis, int defaultValue=1000)  
*Identify cliques and construct corresponding objects.*
- [CbcModel \\* integerPresolve](#) (bool weak=false)  
*Do integer presolve, creating a new (presolved) model.*
- bool [integerPresolveThisModel](#) (OsiSolverInterface \*originalSolver, bool weak=false)  
*Do integer presolve, modifying the current model.*
- void [originalModel](#) ([CbcModel](#) \*presolvedModel, bool weak)  
*Put back information into the original model after integer presolve.*
- bool [tightenVubs](#) (int type, bool allowMultipleBinary=false, double useCutoff=1.0e50)  
*For variables involved in VUB constraints, see if we can tighten bounds by solving lp's.*
- bool [tightenVubs](#) (int numberVubs, const int \*which, double useCutoff=1.0e50)  
*For variables involved in VUB constraints, see if we can tighten bounds by solving lp's.*
- void [analyzeObjective](#) ()  
*Analyze problem to find a minimum change in the objective function.*
- void [AddIntegers](#) ()  
*Add additional integers.*
- void [saveModel](#) (OsiSolverInterface \*saveSolver, double \*checkCutoffForRestart, bool \*feasible)

- *Save copy of the model.*
- void `flipModel ()`  
*Flip direction of optimization on all models.*

### Object manipulation routines

See `OsiObject` for an explanation of 'object' in the context of `CbcModel`.

- int `numberObjects ()` const  
*Get the number of objects.*
- void `setNumberObjects (int number)`  
*Set the number of objects.*
- `OsiObject ** objects ()` const  
*Get the array of objects.*
- const `OsiObject * object (int which)` const  
*Get the specified object.*
- `OsiObject * modifiableObject (int which)` const  
*Get the specified object.*
- void `setOptionalInteger (int index)`
- void `deleteObjects (bool findIntegers=true)`  
*Delete all object information (and just back to integers if true)*
- void `addObjects (int numberObjects, OsiObject **objects)`  
*Add in object information.*
- void `addObjects (int numberObjects, CbcObject **objects)`  
*Add in object information.*
- void `synchronizeModel ()`  
*Ensure attached objects point to this model.*
- void `findIntegers (bool startAgain, int type=0)`  
*Identify integer variables and create corresponding objects.*

### Parameter set/get methods

The set methods return true if the parameter was set to the given value, false if the value of the parameter is out of range.

The get methods return the value of the parameter.

- bool `setIntParam (CbcIntParam key, int value)`  
*Set an integer parameter.*
- bool `setDbiParam (CbcDbiParam key, double value)`  
*Set a double parameter.*
- int `getIntParam (CbcIntParam key)` const  
*Get an integer parameter.*
- double `getDbiParam (CbcDbiParam key)` const  
*Get a double parameter.*
- void `setCutoff (double value)`  
*Set cutoff bound on the objective function.*
- double `getCutoff ()` const  
*Get the cutoff bound on the objective function - always as minimize.*
- bool `setMaximumNodes (int value)`  
*Set the maximum node limit .*
- int `getMaximumNodes ()` const  
*Get the maximum node limit .*
- bool `setMaximumSolutions (int value)`  
*Set the maximum number of solutions desired.*
- int `getMaximumSolutions ()` const  
*Get the maximum number of solutions desired.*

- bool [setPrintingMode](#) (int value)  
*Set the printing mode.*
- int [getPrintingMode](#) () const  
*Get the printing mode.*
- bool [setMaximumSeconds](#) (double value)  
*Set the [maximum number of seconds](#) desired.*
- double [getMaximumSeconds](#) () const  
*Get the [maximum number of seconds](#) desired.*
- double [getCurrentSeconds](#) () const  
*Current time since start of branchAndbound.*
- bool [maximumSecondsReached](#) () const  
*Return true if maximum time reached.*
- bool [setIntegerTolerance](#) (double value)  
*Set the [integrality tolerance](#) .*
- double [getIntegerTolerance](#) () const  
*Get the [integrality tolerance](#) .*
- bool [setInfeasibilityWeight](#) (double value)  
*Set the [weight per integer infeasibility](#) .*
- double [getInfeasibilityWeight](#) () const  
*Get the [weight per integer infeasibility](#) .*
- bool [setAllowableGap](#) (double value)  
*Set the [allowable gap](#) between the best known solution and the best possible solution.*
- double [getAllowableGap](#) () const  
*Get the [allowable gap](#) between the best known solution and the best possible solution.*
- bool [setAllowableFractionGap](#) (double value)  
*Set the [fraction allowable gap](#) between the best known solution and the best possible solution.*
- double [getAllowableFractionGap](#) () const  
*Get the [fraction allowable gap](#) between the best known solution and the best possible solution.*
- bool [setAllowablePercentageGap](#) (double value)  
*Set the [percentage allowable gap](#) between the best known solution and the best possible solution.*
- double [getAllowablePercentageGap](#) () const  
*Get the [percentage allowable gap](#) between the best known solution and the best possible solution.*
- bool [setHeuristicGap](#) (double value)  
*Set the [heuristic gap](#) between the best known solution and the best possible solution.*
- double [getHeuristicGap](#) () const  
*Get the [heuristic gap](#) between the best known solution and the best possible solution.*
- bool [setHeuristicFractionGap](#) (double value)  
*Set the [fraction heuristic gap](#) between the best known solution and the best possible solution.*
- double [getHeuristicFractionGap](#) () const  
*Get the [fraction heuristic gap](#) between the best known solution and the best possible solution.*
- bool [setCutoffIncrement](#) (double value)  
*Set the [CbcModel::CbcCutoffIncrement](#) desired.*
- double [getCutoffIncrement](#) () const  
*Get the [CbcModel::CbcCutoffIncrement](#) desired.*
- bool [canStopOnGap](#) () const  
*See if can stop on gap.*
- void [setHotstartSolution](#) (const double \*solution, const int \*priorities=NULL)  
*Pass in target solution and optional priorities.*
- void [setMinimumDrop](#) (double value)  
*Set the minimum drop to continue cuts.*
- double [getMinimumDrop](#) () const  
*Get the minimum drop to continue cuts.*
- void [setMaximumCutPassesAtRoot](#) (int value)  
*Set the maximum number of cut passes at root node (default 20) Minimum drop can also be used for fine tuning.*
- int [getMaximumCutPassesAtRoot](#) () const

- Get the maximum number of cut passes at root node.*
- void `setMaximumCutPasses` (int value)
- Set the maximum number of cut passes at other nodes (default 10) Minimum drop can also be used for fine tuning.*
- int `getMaximumCutPasses` () const
- Get the maximum number of cut passes at other nodes (default 10)*
- int `getCurrentPassNumber` () const
- Get current cut pass number in this round of cuts.*
- void `setCurrentPassNumber` (int value)
- Set current cut pass number in this round of cuts.*
- void `setNumberStrong` (int number)
- Set the maximum number of candidates to be evaluated for strong branching.*
- int `numberStrong` () const
- Get the maximum number of candidates to be evaluated for strong branching.*
- void `setPreferredWay` (int value)
- Set global preferred way to branch -1 down, +1 up, 0 no preference.*
- int `getPreferredWay` () const
- Get the preferred way to branch (default 0)*
- int `whenCuts` () const
- Get at which depths to do cuts.*
- void `setWhenCuts` (int value)
- Set at which depths to do cuts.*
- bool `doCutsNow` (int allowForTopOfTree) const
- Return true if we want to do cuts If allowForTopOfTree zero then just does on multiples of depth if 1 then allows for doing at top of tree if 2 then says if cuts allowed anywhere apart from root.*
- void `setNumberBeforeTrust` (int number)
- Set the number of branches before pseudo costs believed in dynamic strong branching.*
- int `numberBeforeTrust` () const
- get the number of branches before pseudo costs believed in dynamic strong branching.*
- void `setNumberPenalties` (int number)
- Set the number of variables for which to compute penalties in dynamic strong branching.*
- int `numberPenalties` () const
- get the number of variables for which to compute penalties in dynamic strong branching.*
- const `CbcFullNodeInfo * topOfTree` () const
- Pointer to top of tree.*
- void `setNumberAnalyzeIterations` (int number)
- Number of analyze iterations to do.*
- int `numberAnalyzeIterations` () const
- double `penaltyScaleFactor` () const
- Get scale factor to make penalties match strong.*
- void `setPenaltyScaleFactor` (double value)
- Set scale factor to make penalties match strong.*
- void `setProblemType` (int number)
- Problem type as set by user or found by analysis.*
- int `problemType` () const
- int `currentDepth` () const
- Current depth.*
- void `setHowOftenGlobalScan` (int number)
- Set how often to scan global cuts.*
- int `howOftenGlobalScan` () const
- Get how often to scan global cuts.*
- int \* `originalColumns` () const
- Original columns as created by integerPresolve or preprocessing.*
- void `setOriginalColumns` (const int \*originalColumns, int numberGood=COIN\_INT\_MAX)
- Set original columns as created by preprocessing.*
- OsiRowCut \* `conflictCut` (const OsiSolverInterface \*solver, bool &localCuts)

- *Create conflict cut (well - most of)*
- void `setPrintFrequency` (int number)  
*Set the print frequency.*
- int `printFrequency` () const  
*Get the print frequency.*

#### Methods returning info on how the solution process terminated

- bool `isAbandoned` () const  
*Are there a numerical difficulties?*
- bool `isProvenOptimal` () const  
*Is optimality proven?*
- bool `isProvenInfeasible` () const  
*Is infeasibility proven (or none better than cutoff)?*
- bool `isContinuousUnbounded` () const  
*Was continuous solution unbounded.*
- bool `isProvenDualInfeasible` () const  
*Was continuous solution unbounded.*
- bool `isNodeLimitReached` () const  
*Node limit reached?*
- bool `isSecondsLimitReached` () const  
*Time limit reached?*
- bool `isSolutionLimitReached` () const  
*Solution limit reached?*
- int `getIterationCount` () const  
*Get how many iterations it took to solve the problem.*
- void `incrementIterationCount` (int value)  
*Increment how many iterations it took to solve the problem.*
- int `getNodeCount` () const  
*Get how many Nodes it took to solve the problem (including those in complete fathoming B&B inside CLP).*
- void `incrementNodeCount` (int value)  
*Increment how many nodes it took to solve the problem.*
- int `getExtraNodeCount` () const  
*Get how many Nodes were enumerated in complete fathoming B&B inside CLP.*
- int `status` () const  
*Final status of problem Some of these can be found out by is.....*
- void `setProblemStatus` (int value)
- int `secondaryStatus` () const  
*Secondary status of problem -1 unset (status\_ will also be -1) 0 search completed with solution 1 linear relaxation not feasible (or worse than cutoff) 2 stopped on gap 3 stopped on nodes 4 stopped on time 5 stopped on user event 6 stopped on solutions 7 linear relaxation unbounded 8 stopped on iteration limit.*
- void `setSecondaryStatus` (int value)
- bool `isInitialSolveAbandoned` () const  
*Are there numerical difficulties (for initialSolve) ?*
- bool `isInitialSolveProvenOptimal` () const  
*Is optimality proven (for initialSolve) ?*
- bool `isInitialSolveProvenPrimalInfeasible` () const  
*Is primal infeasibility proven (for initialSolve) ?*
- bool `isInitialSolveProvenDualInfeasible` () const  
*Is dual infeasibility proven (for initialSolve) ?*

#### Problem information methods

*These methods call the solver's query routines to return information about the problem referred to by the current object.*

Querying a problem 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 valid as long as the data is unchanged and the solver is not called.

- int `numberOfRowsAtContinuous` () const  
*Number of rows in continuous (root) problem.*
- int `getNumCols` () const  
*Get number of columns.*
- int `getNumRows` () const  
*Get number of rows.*
- CoinBigIndex `getNumElements` () const  
*Get number of nonzero elements.*
- int `numberOfIntegers` () const  
*Number of integers in problem.*
- const int \* `integerVariable` () const
- char `integerType` (int i) const  
*Whether or not integer.*
- const char \* `integerType` () const  
*Whether or not integer.*
- 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 row constraint senses.*
- const double \* `getRightHandSide` () const  
*Get pointer to array[getNumRows()] of rows 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.*
- 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 CoinPackedMatrix \* `getMatrixByRow` () const  
*Get pointer to row-wise copy of matrix.*
- const CoinPackedMatrix \* `getMatrixByCol` () const  
*Get pointer to column-wise copy of matrix.*
- double `getInfinity` () const  
*Get solver's value for infinity.*
- const double \* `getCbcColLower` () const



- Get pointer to array[getNumCols()] (for speed) of column lower bounds.*
- const double \* [getCbcColUpper](#) () const
- Get pointer to array[getNumCols()] (for speed) of column upper bounds.*
- const double \* [getCbcRowLower](#) () const
- Get pointer to array[getNumRows()] (for speed) of row lower bounds.*
- const double \* [getCbcRowUpper](#) () const
- Get pointer to array[getNumRows()] (for speed) of row upper bounds.*
- const double \* [getCbcColSolution](#) () const
- Get pointer to array[getNumCols()] (for speed) of primal solution vector.*
- const double \* [getCbcRowPrice](#) () const
- Get pointer to array[getNumRows()] (for speed) of dual prices.*
- const double \* [getCbcReducedCost](#) () const
- Get a pointer to array[getNumCols()] (for speed) of reduced costs.*
- const double \* [getCbcRowActivity](#) () const
- Get pointer to array[getNumRows()] (for speed) of row activity levels.*

### Methods related to querying the solution

- double \* [continuousSolution](#) () const
- Holds solution at continuous (after cuts if branchAndBound called)*
- int \* [usedInSolution](#) () const
- Array marked whenever a solution is found if non-zero.*
- void [incrementUsed](#) (const double \*solution)
- Increases usedInSolution for nonzeros.*
- void [setBestSolution](#) (CBC\_Message how, double &objectiveValue, const double \*solution, int fixVariables=0)
- Record a new incumbent solution and update objectiveValue.*
- void [setBestObjectiveValue](#) (double objectiveValue)
- Just update objectiveValue.*
- [CbcEventHandler::CbcAction](#) [dealWithEventHandler](#) ([CbcEventHandler::CbcEvent](#) event, double objValue, const double \*solution)
- Deals with event handler and solution.*
- virtual double [checkSolution](#) (double cutoff, double \*solution, int fixVariables, double originalObjValue)
- Call this to really test if a valid solution can be feasible Solution is number columns in size.*
- bool [feasibleSolution](#) (int &numberIntegerInfeasibilities, int &numberObjectInfeasibilities) const
- Test the current solution for feasibility.*
- double \* [currentSolution](#) () const
- Solution to the most recent lp relaxation.*
- const double \* [testSolution](#) () const
- For testing infeasibilities - will point to currentSolution\_ or solver->[getColSolution\(\)](#)*
- void [setTestSolution](#) (const double \*solution)
- void [reserveCurrentSolution](#) (const double \*solution=NULL)
- Make sure region there and optionally copy solution.*
- const double \* [getColSolution](#) () const
- Get pointer to array[getNumCols()] of primal solution vector.*
- const double \* [getRowPrice](#) () const
- Get pointer to array[getNumRows()] of dual prices.*
- 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.*
- double [getCurrentObjValue](#) () const
- Get current objective function value.*
- double [getCurrentMinimizationObjValue](#) () const
- Get current minimization objective function value.*
- double [getMinimizationObjValue](#) () const

- Get best objective function value as minimization.*
- void [setMinimizationObjValue](#) (double value)
- Set best objective function value as minimization.*
- double [getObjValue](#) () const
- Get best objective function value.*
- double [getBestPossibleObjValue](#) () const
- Get best possible objective function value.*
- void [setObjValue](#) (double value)
- Set best objective function value.*
- double [getSolverObjValue](#) () const
- Get solver objective function value (as minimization)*
- double \* [bestSolution](#) () const
- The best solution to the integer programming problem.*
- void [setBestSolution](#) (const double \*solution, int numberColumns, double objectiveValue, bool check=false)
- User callable setBestSolution.*
- int [getSolutionCount](#) () const
- Get number of solutions.*
- void [setSolutionCount](#) (int value)
- Set number of solutions (so heuristics will be different)*
- int [numberSavedSolutions](#) () const
- Number of saved solutions (including best)*
- int [maximumSavedSolutions](#) () const
- Maximum number of extra saved solutions.*
- void [setMaximumSavedSolutions](#) (int value)
- Set maximum number of extra saved solutions.*
- const double \* [savedSolution](#) (int which) const
- Return a saved solution (0==best) - NULL if off end.*
- double [savedSolutionObjective](#) (int which) const
- Return a saved solution objective (0==best) - COIN\_DBL\_MAX if off end.*
- void [deleteSavedSolution](#) (int which)
- Delete a saved solution and move others up.*
- int [phase](#) () const
- Current phase (so heuristics etc etc can find out).*
- int [getNumberHeuristicSolutions](#) () const
- Get number of heuristic solutions.*
- void [setNumberHeuristicSolutions](#) (int value)
- Set number of heuristic solutions.*
- void [setObjSense](#) (double s)
- Set objective function sense (1 for min (default), -1 for max.)*
- double [getContinuousObjective](#) () const
- Value of objective at continuous.*
- void [setContinuousObjective](#) (double value)
- int [getContinuousInfeasibilities](#) () const
- Number of infeasibilities at continuous.*
- void [setContinuousInfeasibilities](#) (int value)
- double [rootObjectiveAfterCuts](#) () const
- Value of objective after root node cuts added.*
- double [sumChangeObjective](#) () const
- Sum of Changes to objective by first solve.*
- int [numberGlobalViolations](#) () const
- Number of times global cuts violated.*
- void [clearNumberGlobalViolations](#) ()
- bool [resolveAfterTakeOffCuts](#) () const
- Whether to force a resolve after takeOffCuts.*
- void [setResolveAfterTakeOffCuts](#) (bool yesNo)

- int [maximumRows](#) () const  
*Maximum number of rows.*
- CoinWarmStartBasis & [workingBasis](#) ()  
*Work basis for temporary use.*
- int [getStopNumberIterations](#) () const  
*Get number of "iterations" to stop after.*
- void [setStopNumberIterations](#) (int value)  
*Set number of "iterations" to stop after.*
- [CbcModel](#) \* [heuristicModel](#) () const  
*A pointer to model from [CbcHeuristic](#).*
- void [setHeuristicModel](#) ([CbcModel](#) \*model)  
*Set a pointer to model from [CbcHeuristic](#).*

### Node selection

- [CbcCompareBase](#) \* [nodeComparison](#) () const
- void [setNodeComparison](#) ([CbcCompareBase](#) \*compare)
- void [setNodeComparison](#) ([CbcCompareBase](#) &compare)

### Problem feasibility checking

- [CbcFeasibilityBase](#) \* [problemFeasibility](#) () const
- void [setProblemFeasibility](#) ([CbcFeasibilityBase](#) \*feasibility)
- void [setProblemFeasibility](#) ([CbcFeasibilityBase](#) &feasibility)

### Tree methods and subtree methods

- [CbcTree](#) \* [tree](#) () const  
*Tree method e.g. heap (which may be overridden by inheritance)*
- void [passInTreeHandler](#) ([CbcTree](#) &tree)  
*For modifying tree handling (original is cloned)*
- void [passInSubTreeModel](#) ([CbcModel](#) &model)  
*For passing in an [CbcModel](#) to do a sub Tree (with derived tree handlers).*
- [CbcModel](#) \* [subTreeModel](#) (OsiSolverInterface \*solver=NULL) const  
*For retrieving a copy of subtree model with given OsiSolver.*
- int [numberStoppedSubTrees](#) () const  
*Returns number of times any subtree stopped on nodes, time etc.*
- void [incrementSubTreeStopped](#) ()  
*Says a sub tree was stopped.*
- int [typePresolve](#) () const  
*Whether to automatically do presolve before branch and bound (subTrees).*
- void [setTypePresolve](#) (int value)

### Branching Decisions

See the [CbcBranchDecision](#) class for additional information.

- [CbcBranchDecision](#) \* [branchingMethod](#) () const  
*Get the current branching decision method.*
- void [setBranchingMethod](#) ([CbcBranchDecision](#) \*method)  
*Set the branching decision method.*
- void [setBranchingMethod](#) ([CbcBranchDecision](#) &method)  
*Set the branching method.*
- [CbcCutModifier](#) \* [cutModifier](#) () const  
*Get the current cut modifier method.*
- void [setCutModifier](#) ([CbcCutModifier](#) \*modifier)

- *Set the cut modifier method.*
- void [setCutModifier](#) ([CbcCutModifier](#) &modifier)
- *Set the cut modifier method.*

### Row (constraint) and Column (variable) cut generation

- int [stateOfSearch](#) () const
- *State of search 0 - no solution 1 - only heuristic solutions 2 - branched to a solution 3 - no solution but many nodes.*
- void [setStateOfSearch](#) (int state)
- int [searchStrategy](#) () const
- *Strategy worked out - mainly at root node for use by [CbcNode](#).*
- void [setSearchStrategy](#) (int value)
- *Set strategy worked out - mainly at root node for use by [CbcNode](#).*
- int [strongStrategy](#) () const
- *Strong branching strategy.*
- void [setStrongStrategy](#) (int value)
- *Set strong branching strategy.*
- int [numberCutGenerators](#) () const
- *Get the number of cut generators.*
- [CbcCutGenerator](#) \*\* [cutGenerators](#) () const
- *Get the list of cut generators.*
- [CbcCutGenerator](#) \* [cutGenerator](#) (int i) const
- *Get the specified cut generator.*
- [CbcCutGenerator](#) \* [virginCutGenerator](#) (int i) const
- *Get the specified cut generator before any changes.*
- void [addCutGenerator](#) ([CglCutGenerator](#) \*generator, int howOften=1, const char \*name=NULL, bool normal=true, bool atSolution=false, bool infeasible=false, int howOftenInSub=-100, int whatDepth=-1, int whatDepthInSub=-1)
- *Add one generator - up to user to delete generators.*

### Strategy and sub models

See the [CbcStrategy](#) class for additional information.

- [CbcStrategy](#) \* [strategy](#) () const
- *Get the current strategy.*
- void [setStrategy](#) ([CbcStrategy](#) &strategy)
- *Set the strategy. Clones.*
- void [setStrategy](#) ([CbcStrategy](#) \*strategy)
- *Set the strategy. assigns.*
- [CbcModel](#) \* [parentModel](#) () const
- *Get the current parent model.*
- void [setParentModel](#) ([CbcModel](#) &parentModel)
- *Set the parent model.*

### Heuristics and priorities

- void [addHeuristic](#) ([CbcHeuristic](#) \*generator, const char \*name=NULL, int before=-1)
- *Add one heuristic - up to user to delete.*
- [CbcHeuristic](#) \* [heuristic](#) (int i) const
- *Get the specified heuristic.*
- int [numberHeuristics](#) () const
- *Get the number of heuristics.*
- void [setNumberHeuristics](#) (int value)
- *Set the number of heuristics.*

- `CbcHeuristic * lastHeuristic ()` const  
*Pointer to heuristic solver which found last solution (or NULL)*
- void `setLastHeuristic (CbcHeuristic *last)`  
*set last heuristic which found a solution*
- void `passInPriorities (const int *priorities, bool ifNotSimpleIntegers)`  
*Pass in branching priorities.*
- int `priority (int sequence)` const  
*Returns priority level for an object (or 1000 if no priorities exist)*
- void `passInEventHandler (const CbcEventHandler *eventHandler)`  
*Set an event handler.*
- `CbcEventHandler * getEventHandler ()` const  
*Retrieve a pointer to the event handler.*

### Setting/Accessing application data

- void `setApplicationData (void *appData)`  
*Set application data.*
- void \* `getApplicationData ()` const  
*Get application data.*
- void `passInSolverCharacteristics (OsiBabSolver *solverCharacteristics)`  
*For advanced applications you may wish to modify the behavior of Cbc e.g.*
- const OsiBabSolver \* `solverCharacteristics ()` const  
*Get solver characteristics.*

### Message handling etc

- void `passInMessageHandler (CoinMessageHandler *handler)`  
*Pass in Message handler (not deleted at end)*
- void `newLanguage (CoinMessages::Language language)`  
*Set language.*
- void `setLanguage (CoinMessages::Language language)`
- CoinMessageHandler \* `messageHandler ()` const  
*Return handler.*
- CoinMessages & `messages ()`  
*Return messages.*
- CoinMessages \* `messagesPointer ()`  
*Return pointer to messages.*
- void `setLogLevel (int value)`  
*Set log level.*
- int `logLevel ()` const  
*Get log level.*
- void `setDefaultHandler (bool yesNo)`  
*Set flag to say if handler\_ is the default handler.*
- bool `defaultHandler ()` const  
*Check default handler.*

### Specialized

- void `setSpecialOptions (int value)`  
*Set special options 0 bit (1) - check if cuts valid (if on debugger list) 1 bit (2) - use current basis to check integer solution (rather than all slack) 2 bit (4) - don't check integer solution (by solving LP) 3 bit (8) - fast analyze 4 bit (16) - non-linear model - so no well defined CoinPackedMatrix 5 bit (32) - keep names 6 bit (64) - try for dominated columns 7 bit (128) - SOS type 1 but all declared integer 8 bit (256) - Set to say solution just found, unset by doing cuts 9 bit (512) - Try reduced model after 100 nodes 10 bit (1024) - Switch on some heuristics even if seems unlikely 11 bit (2048) - Mark as in small branch and bound 12 bit (4096) - Funny cuts so do slow way (in some places) 13 bit (8192) - Funny cuts so do*

slow way (in other places) 14 bit (16384) - Use Cplex! for fathoming 15 bit (32768) - Try reduced model after 0 nodes 16 bit (65536) - Original model had integer bounds 17 bit (131072) - Perturbation switched off 18 bit (262144) - donor [CbcModel](#) 19 bit (524288) - recipient [CbcModel](#) 20 bit (1048576) - waiting for sub model to return 22 bit (4194304) - do not initialize random seed in solver (user has) 23 bit (8388608) - leave solver\_ with cuts 24 bit (16777216) - just get feasible if no cutoff.

- int [specialOptions](#) () const  
Get special options.
- void [setRandomSeed](#) (int value)  
Set random seed.
- int [getRandomSeed](#) () const  
Get random seed.
- void [setMultipleRootTries](#) (int value)  
Set multiple root tries.
- int [getMultipleRootTries](#) () const  
Get multiple root tries.
- void [sayEventHappened](#) ()  
Tell model to stop on event.
- bool [normalSolver](#) () const  
Says if normal solver i.e. has well defined CoinPackedMatrix.
- bool [waitingForMiniBranchAndBound](#) () const  
Says if model is sitting there waiting for mini branch and bound to finish This is because an event handler may only have access to parent model in mini branch and bound.
- void [setMoreSpecialOptions](#) (int value)  
Set more special options at present bottom 6 bits used for shadow price mode 1024 for experimental hotstart 2048,4096 breaking out of cuts 8192 slowly increase minimum drop 16384 gomory 32768 more heuristics in sub trees 65536 no cuts in preprocessing 131072 Time limits elapsed 18 bit (262144) - Perturb fathom nodes 19 bit (524288) - No limit on fathom nodes 20 bit (1048576) - Reduce sum of infeasibilities before cuts 21 bit (2097152) - Reduce sum of infeasibilities after cuts 22 bit (4194304) - Conflict analysis 23 bit (8388608) - Conflict analysis - temporary bit 24 bit (16777216) - Add cutoff as LP constraint (out) 25 bit (33554432) - diving/reordering 26 bit (67108864) - load global cuts from file 27 bit (134217728) - append binding global cuts to file 28 bit (268435456) - idiot branching 29 bit (536870912) - don't make fake objective.
- int [moreSpecialOptions](#) () const  
Get more special options.
- void [setMoreSpecialOptions2](#) (int value)  
Set more more special options 0 bit (1) - find switching variables 1 bit (2) - using fake objective until solution 2 bit (4) - switching variables exist 3 bit (8) - skip most of setBestSolution checks 4 bit (16) - very lightweight preprocessing in smallB&B 5 bit (32) - event handler needs to be cloned when parallel.
- int [moreSpecialOptions2](#) () const  
Get more special options2.
- void [setCutoffAsConstraint](#) (bool yesNo)  
Set cutoff as constraint.
- void [setUseElapsedTime](#) (bool yesNo)  
Set time method.
- bool [useElapsedTime](#) () const  
Get time method.
- void \* [temporaryPointer](#) () const  
Get useful temporary pointer.
- void [setTemporaryPointer](#) (void \*pointer)  
Set useful temporary pointer.
- void [goToDantzig](#) (int numberNodes, ClpDualRowPivot \*&savePivotMethod)  
Go to dantzig pivot selection if easy problem (clp only)
- bool [ownObjects](#) () const  
Now we may not own objects - just point to solver's objects.
- void [checkModel](#) ()  
Check original model before it gets messed up.

**Constructors and destructors etc**

- [CbcModel](#) ()  
*Default Constructor.*
- [CbcModel](#) (const [OsiSolverInterface](#) &)  
*Constructor from solver.*
- void [assignSolver](#) ([OsiSolverInterface](#) \*[solver](#), bool deleteSolver=true)  
*Assign a solver to the model (model assumes ownership)*
- void [setModelOwnsSolver](#) (bool ourSolver)  
*Set ownership of solver.*
- bool [modelOwnsSolver](#) ()  
*Get ownership of solver.*
- [CbcModel](#) (const [CbcModel](#) &rhs, bool cloneHandler=false)  
*Copy constructor.*
- virtual [CbcModel](#) \* [clone](#) (bool cloneHandler)  
*Clone.*
- [CbcModel](#) & [operator=](#) (const [CbcModel](#) &rhs)  
*Assignment operator.*
- virtual [~CbcModel](#) ()  
*Destructor.*
- [OsiSolverInterface](#) \* [solver](#) () const  
*Returns solver - has current state.*
- [OsiSolverInterface](#) \* [swapSolver](#) ([OsiSolverInterface](#) \*[solver](#))  
*Returns current solver - sets new one.*
- [OsiSolverInterface](#) \* [continuousSolver](#) () const  
*Returns solver with continuous state.*
- void [createContinuousSolver](#) ()  
*Create solver with continuous state.*
- void [clearContinuousSolver](#) ()  
*Clear solver with continuous state.*
- [OsiSolverInterface](#) \* [referenceSolver](#) () const  
*A copy of the solver, taken at constructor or by saveReferenceSolver.*
- void [saveReferenceSolver](#) ()  
*Save a copy of the current solver so can be reset to.*
- void [resetToReferenceSolver](#) ()  
*Uses a copy of reference solver to be current solver.*
- void [gutsOfDestructor](#) ()  
*Clears out as much as possible (except solver)*
- void [gutsOfDestructor2](#) ()  
*Clears out enough to reset [CbcModel](#) as if no branch and bound done.*
- void [resetModel](#) ()  
*Clears out enough to reset [CbcModel](#) cutoff etc.*
- void [gutsOfCopy](#) (const [CbcModel](#) &rhs, int mode=0)  
*Most of copy constructor mode - 0 copy but don't delete before 1 copy and delete before 2 copy and delete before (but use virgin generators)*
- void [moveInfo](#) (const [CbcModel](#) &rhs)  
*Move status, nodes etc etc across.*

**semi-private i.e. users should not use**

- int [getNodeCount2](#) () const  
*Get how many Nodes it took to solve the problem.*
- void [setPointers](#) (const [OsiSolverInterface](#) \*[solver](#))  
*Set pointers for speed.*
- int [reducedCostFix](#) ()



- Perform reduced cost fixing.*
- void [synchronizeHandlers](#) (int makeDefault)
- Makes all handlers same.*
- void [saveExtraSolution](#) (const double \*solution, double objectiveValue)
- Save a solution to saved list.*
- void [saveBestSolution](#) (const double \*solution, double objectiveValue)
- Save a solution to best and move current to saved.*
- void [deleteSolutions](#) ()
- Delete best and saved solutions.*
- int [resolve](#) (OsiSolverInterface \*solver)
- Encapsulates solver resolve.*
- int [chooseBranch](#) (CbcNode \*&newNode, int numberPassesLeft, CbcNode \*oldNode, OsiCuts &cuts, bool &resolved, CoinWarmStartBasis \*lastws, const double \*lowerBefore, const double \*upperBefore, OsiSolverBranch \*&branches)
- Encapsulates choosing a variable - anyAction -2, infeasible (-1 round again), 0 done.*
- int [chooseBranch](#) (CbcNode \*newNode, int numberPassesLeft, bool &resolved)
- CoinWarmStartBasis \* [getEmptyBasis](#) (int ns=0, int na=0) const
- Return an empty basis object of the specified size.*
- int [takeOffCuts](#) (OsiCuts &cuts, bool allowResolve, OsiCuts \*saveCuts, int numberNewCuts=0, const OsiRowCut \*\*newCuts=NULL)
- Remove inactive cuts from the model.*
- int [addCuts](#) (CbcNode \*node, CoinWarmStartBasis \*&lastws, bool canFix)
- Determine and install the active cuts that need to be added for the current subproblem.*
- bool [addCuts1](#) (CbcNode \*node, CoinWarmStartBasis \*&lastws)
- Traverse the tree from node to root and prep the model.*
- void [previousBounds](#) (CbcNode \*node, CbcNodeInfo \*where, int iColumn, double &lower, double &upper, int force)
- Returns bounds just before where - initially original bounds.*
- void [setObjectiveValue](#) (CbcNode \*thisNode, const CbcNode \*parentNode) const
- Set objective value in a node.*
- void [convertToDynamic](#) ()
- If numberBeforeTrust > 0 then we are going to use CbcBranchDynamic.*
- void [synchronizeNumberBeforeTrust](#) (int type=0)
- Set numberBeforeTrust in all objects.*
- void [zapIntegerInformation](#) (bool leaveObjects=true)
- Zap integer information in problem (may leave object info)*
- int [cliquePseudoCosts](#) (int doStatistics)
- Use cliques for pseudocost information - return nonzero if infeasible.*
- void [pseudoShadow](#) (int type)
- Fill in useful estimates.*
- void [fillPseudoCosts](#) (double \*downCosts, double \*upCosts, int \*priority=NULL, int \*numberDown=NULL, int \*numberUp=NULL, int \*numberDownInfeasible=NULL, int \*numberUpInfeasible=NULL) const
- Return pseudo costs If not all integers or not pseudo costs - returns all zero Length of arrays are [numberIntegers\(\)](#) and entries correspond to [integerVariable\(\)\[i\]](#) User must allocate arrays before call.*
- void [doHeuristicsAtRoot](#) (int deleteHeuristicsAfterwards=0)
- Do heuristics at root.*
- void [adjustHeuristics](#) ()
- Adjust heuristics based on model.*
- const double \* [hotstartSolution](#) () const
- Get the hotstart solution.*
- const int \* [hotstartPriorities](#) () const
- Get the hotstart priorities.*
- CbcCountRowCut \*\* [addedCuts](#) () const
- Return the list of cuts initially collected for this subproblem.*
- int [currentNumberCuts](#) () const



- Number of entries in the list returned by [addedCuts\(\)](#)
- [CbcRowCuts](#) \* [globalCuts](#) ()

Global cuts.
- void [setNextRowCut](#) (const OsiRowCut &cut)

Copy and set a pointer to a row cut which will be added instead of normal branching.
- [CbcNode](#) \* [currentNode](#) () const

Get a pointer to current node (be careful)
- [CglTreeProbingInfo](#) \* [probingInfo](#) () const

Get a pointer to probing info.
- [CoinThreadRandom](#) \* [randomNumberGenerator](#) ()

Thread specific random number generator.
- void [setNumberStrongIterations](#) (int number)

Set the number of iterations done in strong branching.
- int [numberStrongIterations](#) () const

Get the number of iterations done in strong branching.
- int [maximumNumberIterations](#) () const

Get maximum number of iterations (designed to be used in heuristics)
- void [setMaximumNumberIterations](#) (int value)

Set maximum number of iterations (designed to be used in heuristics)
- void [setFastNodeDepth](#) (int value)

Set depth for fast nodes.
- int [fastNodeDepth](#) () const

Get depth for fast nodes.
- int [continuousPriority](#) () const

Get anything with priority >= this can be treated as continuous.
- void [setContinuousPriority](#) (int value)

Set anything with priority >= this can be treated as continuous.
- void [incrementExtra](#) (int nodes, int iterations)
- int [numberExtraIterations](#) () const

Number of extra iterations.
- void [incrementStrongInfo](#) (int numberTimes, int numberIterations, int numberFixed, bool ifInfeasible)

Increment strong info.
- const int \* [strongInfo](#) () const

Return strong info.
- int \* [mutableStrongInfo](#) ()

Return mutable strong info.
- [CglStored](#) \* [storedRowCuts](#) () const

Get stored row cuts for donor/recipient [CbcModel](#).
- void [setStoredRowCuts](#) ([CglStored](#) \*cuts)

Set stored row cuts for donor/recipient [CbcModel](#).
- bool [allDynamic](#) () const

Says whether all dynamic integers.
- void [generateCpp](#) (FILE \*fp, int options)

Create C++ lines to get to current state.
- [OsiBranchingInformation](#) [usefulInformation](#) () const

Generate an [OsiBranchingInformation](#) object.
- void [setBestSolutionBasis](#) (const [CoinWarmStartBasis](#) &bestSolutionBasis)

Warm start object produced by heuristic or strong branching.
- void [redoWalkBack](#) ()

Redo walkback arrays.

## Solve methods

- void `initialSolve` ()  
*Solve the initial LP relaxation.*
- void `branchAndBound` (int doStatistics=0)  
*Invoke the branch & cut algorithm.*
- void `addUpdateInformation` (const `CbcObjectUpdateData` &data)  
*Adds an update information object.*
- int `doOneNode` (`CbcModel` \*baseModel, `CbcNode` \*&node, `CbcNode` \*&newNode)  
*Do one node - broken out for clarity? also for parallel (when baseModel!=this) Returns 1 if solution found node NULL on return if no branches left newNode NULL if no new node created.*
- int `resolve` (`CbcNodeInfo` \*parent, int whereFrom, double \*saveSolution=NULL, double \*saveLower=NULL, double \*saveUpper=NULL)  
*Reoptimise an LP relaxation.*
- void `makeGlobalCuts` (int numberOfRows, const int \*which)  
*Make given rows (L or G) into global cuts and remove from lp.*
- void `makeGlobalCut` (const `OsiRowCut` \*cut)  
*Make given cut into a global cut.*
- void `makeGlobalCut` (const `OsiRowCut` &cut)  
*Make given cut into a global cut.*
- void `makeGlobalCut` (const `OsiColCut` \*cut)  
*Make given column cut into a global cut.*
- void `makeGlobalCut` (const `OsiColCut` &cut)  
*Make given column cut into a global cut.*
- void `makePartialCut` (const `OsiRowCut` \*cut, const `OsiSolverInterface` \*solver=NULL)  
*Make partial cut into a global cut and save.*
- void `makeGlobalCuts` ()  
*Make partial cuts into global cuts.*
- const int \* `whichGenerator` () const  
*Which cut generator generated this cut.*

## Multithreading

- `CbcThread` \* `masterThread` () const  
*Get pointer to masterthread.*
- `CbcNodeInfo` \*\* `walkback` () const  
*Get pointer to walkback.*
- int `getNumberThreads` () const  
*Get number of threads.*
- void `setNumberThreads` (int value)  
*Set number of threads.*
- int `getThreadMode` () const  
*Get thread mode.*
- void `setThreadMode` (int value)  
*Set thread mode always use numberThreads for branching 1 set then deterministic 2 set then use numberThreads for root cuts 4 set then use numberThreads in root mini branch and bound 8 set and numberThreads - do heuristics numberThreads at a time 8 set and numberThreads==0 do all heuristics at once default is 0.*
- int `parallelMode` () const

*Return -2 if deterministic threaded and main thread -1 if deterministic threaded and serial thread 0 if serial 1 if opportunistic threaded.*

- bool `isLocked` () const

*From here to end of section - code in CbcThread.cpp until class changed Returns true if locked.*

- void `lockThread` ()
- void `unlockThread` ()
- void `setInfoInChild` (int type, `CbcThread` \*info)

*Set information in a child -3 pass pointer to child thread info -2 just stop -1 delete simple child stuff 0 delete opportunistic child stuff 1 delete deterministic child stuff.*

- void `moveToModel` (`CbcModel` \*baseModel, int mode)

*Move/copy information from one model to another -1 - initialization 0 - from base model 1 - to base model (and reset) 2 - add in final statistics etc (and reset so can do clean destruction)*

- int `splitModel` (int numberModels, `CbcModel` \*\*model, int numberNodes)

*Split up nodes.*

- void `startSplitModel` (int numberIterations)

*Start threads.*

- void `mergeModels` (int numberModel, `CbcModel` \*\*model, int numberNodes)

*Merge models.*

- static bool `haveMultiThreadSupport` ()

*Indicates whether Cbc library has been compiled with multithreading support.*

#### 7.74.1 Detailed Description

Simple Branch and bound class.

The `initialSolve()` method solves the initial LP relaxation of the MIP problem. The `branchAndBound()` method can then be called to finish using a branch and cut algorithm.

##### Search Tree Traversal

Subproblems (aka nodes) requiring additional evaluation are stored using the `CbcNode` and `CbcNodeInfo` objects. Ancestry linkage is maintained in the `CbcNodeInfo` object. Evaluation of a subproblem within `branchAndBound()` proceeds as follows:

- The node representing the most promising parent subproblem is popped from the heap which holds the set of subproblems requiring further evaluation.
- Using branching instructions stored in the node, and information in its ancestors, the model and solver are adjusted to create the active subproblem.
- If the parent subproblem will require further evaluation (*i.e.*, there are branches remaining) its node is pushed back on the heap. Otherwise, the node is deleted. This may trigger recursive deletion of ancestors.
- The newly created subproblem is evaluated.
- If the subproblem requires further evaluation, a node is created. All information needed to recreate the subproblem (branching information, row and column cuts) is placed in the node and the node is added to the set of subproblems awaiting further evaluation.

Note that there is never a node representing the active subproblem; the model and solver represent the active subproblem.

## Row (Constraint) Cut Handling

For a typical subproblem, the sequence of events is as follows:

- The subproblem is rebuilt for further evaluation: One result of a call to [addCuts\(\)](#) is a traversal of ancestors, leaving a list of all cuts used in the ancestors in `#addedCuts_`. This list is then scanned to construct a basis that includes only tight cuts. Entries for loose cuts are set to NULL.
- The subproblem is evaluated: One result of a call to `solveWithCuts()` is the return of a set of newly generated cuts for the subproblem. `#addedCuts_` is also kept up-to-date as old cuts become loose.
- The subproblem is stored for further processing: A call to [CbcNodeInfo::addCuts\(\)](#) adds the newly generated cuts to the [CbcNodeInfo](#) object associated with this node.

See [CbcCountRowCut](#) for details of the bookkeeping associated with cut management.

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

### 7.74.2 Member Enumeration Documentation

#### 7.74.2.1 enum `CbcModel::CbcIntParam`

Enumerator

***CbcMaxNumNode*** The maximum number of nodes before terminating.

***CbcMaxNumSol*** The maximum number of solutions before terminating.

***CbcFathomDiscipline*** Fathoming discipline. Controls objective function comparisons for purposes of fathoming by bound or determining monotonic variables.

If 1, action is taken only when the current objective is strictly worse than the target. Implementation is handled by adding a small tolerance to the target.

***CbcPrinting*** Adjusts printout 1 does different node message with number unsatisfied on last branch.

***CbcNumberBranches*** Number of branches (may be more than number of nodes as may include strong branching)

***CbcLastIntParam*** Just a marker, so that a static sized array can store parameters.

Definition at line 104 of file `CbcModel.hpp`.

#### 7.74.2.2 enum `CbcModel::CbcDbIParam`

Enumerator

***CbcIntegerTolerance*** The maximum amount the value of an integer variable can vary from integer and still be considered feasible.

***CbcInfeasibilityWeight*** The objective is assumed to worsen by this amount for each integer infeasibility.

***CbcCutoffIncrement*** The amount by which to tighten the objective function cutoff when a new solution is discovered.

***CbcAllowableGap*** Stop when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this. This is an absolute value. Conversion from a percentage is left to the client.

***CbcAllowableFractionGap*** Stop when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this fraction of of the absolute value of best known solution. Code stops if either this test or `CbcAllowableGap` test succeeds

***CbcMaximumSeconds*** The maximum number of seconds before terminating. A double should be adequate!

**CbcCurrentCutoff** Cutoff - stored for speed.

**CbcOptimizationDirection** Optimization direction - stored for speed.

**CbcCurrentObjectiveValue** Current objective value.

**CbcCurrentMinimizationObjectiveValue** Current minimization objective value.

**CbcStartSeconds** The time at start of model. So that other pieces of code can access

**CbcHeuristicGap** Stop doing heuristics when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this. This is an absolute value. Conversion from a percentage is left to the client.

**CbcHeuristicFractionGap** Stop doing heuristics when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this fraction of the absolute value of best known solution. Code stops if either this test or CbcAllowableGap test succeeds

**CbcSmallestChange** Smallest non-zero change on a branch.

**CbcSumChange** Sum of non-zero changes on a branch.

**CbcLargestChange** Largest non-zero change on a branch.

**CbcSmallChange** Small non-zero change on a branch to be used as guess.

**CbcLastDbIParam** Just a marker, so that a static sized array can store parameters.

Definition at line 130 of file CbcModel.hpp.

### 7.74.3 Constructor & Destructor Documentation

#### 7.74.3.1 CbcModel::CbcModel ( )

Default Constructor.

#### 7.74.3.2 CbcModel::CbcModel ( const OsiSolverInterface & )

Constructor from solver.

#### 7.74.3.3 CbcModel::CbcModel ( const CbcModel & rhs, bool cloneHandler = false )

Copy constructor .

If cloneHandler is true then message handler is cloned

#### 7.74.3.4 virtual CbcModel::~~CbcModel ( ) [virtual]

Destructor.

### 7.74.4 Member Function Documentation

#### 7.74.4.1 void CbcModel::initialSolve ( )

Solve the initial LP relaxation.

Invoke the solver's initialSolve() method.

#### 7.74.4.2 void CbcModel::branchAndBound ( int doStatistics = 0 )

Invoke the branch & cut algorithm.

The method assumes that `initialSolve()` has been called to solve the LP relaxation. It processes the root node, then proceeds to explore the branch & cut search tree. The search ends when the tree is exhausted or one of several execution limits is reached. If `doStatistics` is 1 summary statistics are printed if 2 then also the path to best solution (if found by branching) if 3 then also one line per node

**7.74.4.3** `void CbcModel::addUpdateInformation ( const CbcObjectUpdateData & data )`

Adds an update information object.

**7.74.4.4** `int CbcModel::doOneNode ( CbcModel * baseModel, CbcNode *& node, CbcNode *& newNode )`

Do one node - broken out for clarity? also for parallel (when `baseModel!=this`) Returns 1 if solution found node NULL on return if no branches left `newNode` NULL if no new node created.

**7.74.4.5** `int CbcModel::resolve ( CbcNodeInfo * parent, int whereFrom, double * saveSolution = NULL, double * saveLower = NULL, double * saveUpper = NULL )`

Reoptimise an LP relaxation.

Invoke the solver's `resolve()` method. `whereFrom` - 0 - initial continuous 1 - resolve on branch (before new cuts) 2 - after new cuts 3 - obsolete code or something modified problem in unexpected way 10 - after strong branching has fixed variables at root 11 - after strong branching has fixed variables in tree

returns 1 feasible, 0 infeasible, -1 feasible but skip cuts

**7.74.4.6** `void CbcModel::makeGlobalCuts ( int numberRows, const int * which )`

Make given rows (L or G) into global cuts and remove from lp.

**7.74.4.7** `void CbcModel::makeGlobalCut ( const OsiRowCut * cut )`

Make given cut into a global cut.

**7.74.4.8** `void CbcModel::makeGlobalCut ( const OsiRowCut & cut )`

Make given cut into a global cut.

**7.74.4.9** `void CbcModel::makeGlobalCut ( const OsiColCut * cut )`

Make given column cut into a global cut.

**7.74.4.10** `void CbcModel::makeGlobalCut ( const OsiColCut & cut )`

Make given column cut into a global cut.

**7.74.4.11** `void CbcModel::makePartialCut ( const OsiRowCut * cut, const OsiSolverInterface * solver = NULL )`

Make partial cut into a global cut and save.

**7.74.4.12** `void CbcModel::makeGlobalCuts ( )`

Make partial cuts into global cuts.

**7.74.4.13** `const int* CbcModel::whichGenerator ( ) const [inline]`

Which cut generator generated this cut.

Definition at line 364 of file `CbcModel.hpp`.

**7.74.4.14** `CbcModel* CbcModel::findCliques ( bool makeEquality, int atLeastThisMany, int lessThanThis, int defaultValue = 1000 )`

Identify cliques and construct corresponding objects.

Find cliques with size in the range [*atLeastThisMany*, *lessThanThis*] and construct corresponding `CbcClique` objects. If *makeEquality* is true then a new model may be returned if modifications had to be made, otherwise *this* is returned. If the problem is infeasible *#numberOfObjects\_* is set to -1. A client must use `deleteObjects()` before a second call to `findCliques()`. If priorities exist, clique priority is set to the default.

**7.74.4.15** `CbcModel* CbcModel::integerPresolve ( bool weak = false )`

Do integer presolve, creating a new (presolved) model.

Returns the new model, or NULL if feasibility is lost. If *weak* is true then just does a normal presolve

**Todo** It remains to work out the cleanest way of getting a solution to the original problem at the end. So this is very preliminary.

**7.74.4.16** `bool CbcModel::integerPresolveThisModel ( OsiSolverInterface * originalSolver, bool weak = false )`

Do integer presolve, modifying the current model.

Returns true if the model remains feasible after presolve.

**7.74.4.17** `void CbcModel::originalModel ( CbcModel * presolvedModel, bool weak )`

Put back information into the original model after integer presolve.

**7.74.4.18** `bool CbcModel::tightenVubs ( int type, bool allowMultipleBinary = false, double useCutoff = 1.0e50 )`

For variables involved in VUB constraints, see if we can tighten bounds by solving lp's.

Returns false if feasibility is lost. If `CglProbing` is available, it will be tried as well to see if it can tighten bounds. This routine is just a front end for `tightenVubs(int, const int*, double)`.

If *type* = -1 all variables are processed (could be very slow). If *type* = 0 only variables involved in VUBs are processed. If *type* = *n* > 0, only the *n* most expensive VUB variables are processed, where it is assumed that *x* is at its maximum so delta would have to go to 1 (if *x* not at bound).

If *allowMultipleBinary* is true, then a VUB constraint is a row with one continuous variable and any number of binary variables.

If *useCutoff* < 1.0e30, the original objective is installed as a constraint with *useCutoff* as a bound.

**7.74.4.19** `bool CbcModel::tightenVubs ( int numberVubs, const int * which, double useCutoff = 1.0e50 )`

For variables involved in VUB constraints, see if we can tighten bounds by solving lp's.

This version is just handed a list of variables to be processed.

**7.74.4.20** `void CbcModel::analyzeObjective ( )`

Analyze problem to find a minimum change in the objective function.

**7.74.4.21** `void CbcModel::AddIntegers ( )`

Add additional integers.

**7.74.4.22** `void CbcModel::saveModel ( OsiSolverInterface * saveSolver, double * checkCutoffForRestart, bool * feasible )`

Save copy of the model.

**7.74.4.23** `void CbcModel::flipModel ( )`

Flip direction of optimization on all models.

**7.74.4.24** `int CbcModel::numberOfObjects ( ) const [inline]`

Get the number of objects.

Definition at line 462 of file CbcModel.hpp.

**7.74.4.25** `void CbcModel::setNumberOfObjects ( int number ) [inline]`

Set the number of objects.

Definition at line 466 of file CbcModel.hpp.

**7.74.4.26** `OsiObject** CbcModel::objects ( ) const [inline]`

Get the array of objects.

Definition at line 471 of file CbcModel.hpp.

**7.74.4.27** `const OsiObject* CbcModel::object ( int which ) const [inline]`

Get the specified object.

Definition at line 476 of file CbcModel.hpp.

**7.74.4.28** `OsiObject* CbcModel::modifiableObject ( int which ) const [inline]`

Get the specified object.

Definition at line 480 of file CbcModel.hpp.

**7.74.4.29** `void CbcModel::setOptionalInteger ( int index )`

**7.74.4.30** `void CbcModel::deleteObjects ( bool findIntegers = true )`

Delete all object information (and just back to integers if true)

**7.74.4.31** `void CbcModel::addObjects ( int numberOfObjects, OsiObject ** objects )`

Add in object information.

Objects are cloned; the owner can delete the originals.

**7.74.4.32** `void CbcModel::addObjects ( int numberOfObjects, CbcObject ** objects )`

Add in object information.

Objects are cloned; the owner can delete the originals.

**7.74.4.33** `void CbcModel::synchronizeModel ( )`

Ensure attached objects point to this model.



**7.74.4.34** void CbcModel::findIntegers ( bool *startAgain*, int *type* = 0 )

Identify integer variables and create corresponding objects.

Record integer variables and create an [CbcSimpleInteger](#) object for each one. If *startAgain* is true, a new scan is forced, overwriting any existing integer variable information. If *type* > 0 then 1==PseudoCost, 2 new ones low priority

**7.74.4.35** bool CbcModel::setIntParam ( CbcIntParam *key*, int *value* ) [inline]

Set an integer parameter.

Definition at line 538 of file CbcModel.hpp.

**7.74.4.36** bool CbcModel::setDbiParam ( CbcDbiParam *key*, double *value* ) [inline]

Set a double parameter.

Definition at line 543 of file CbcModel.hpp.

**7.74.4.37** int CbcModel::getIntParam ( CbcIntParam *key* ) const [inline]

Get an integer parameter.

Definition at line 548 of file CbcModel.hpp.

**7.74.4.38** double CbcModel::getDbiParam ( CbcDbiParam *key* ) const [inline]

Get a double parameter.

Definition at line 552 of file CbcModel.hpp.

**7.74.4.39** void CbcModel::setCutoff ( double *value* )

Set cutoff bound on the objective function.

When using strict comparison, the bound is adjusted by a tolerance to avoid accidentally cutting off the optimal solution.

**7.74.4.40** double CbcModel::getCutoff ( ) const [inline]

Get the cutoff bound on the objective function - always as minimize.

Definition at line 563 of file CbcModel.hpp.

**7.74.4.41** bool CbcModel::setMaximumNodes ( int *value* ) [inline]

Set the [maximum node limit](#) .

Definition at line 570 of file CbcModel.hpp.

**7.74.4.42** int CbcModel::getMaximumNodes ( ) const [inline]

Get the [maximum node limit](#) .

Definition at line 575 of file CbcModel.hpp.

**7.74.4.43** bool CbcModel::setMaximumSolutions ( int *value* ) [inline]

Set the [maximum number of solutions](#) desired.

Definition at line 583 of file CbcModel.hpp.

**7.74.4.44** `int CbcModel::getMaximumSolutions ( ) const` `[inline]`

Get the [maximum number of solutions](#) desired.

Definition at line 590 of file CbcModel.hpp.

**7.74.4.45** `bool CbcModel::setPrintingMode ( int value )` `[inline]`

Set the printing mode.

Definition at line 594 of file CbcModel.hpp.

**7.74.4.46** `int CbcModel::getPrintingMode ( ) const` `[inline]`

Get the printing mode.

Definition at line 599 of file CbcModel.hpp.

**7.74.4.47** `bool CbcModel::setMaximumSeconds ( double value )` `[inline]`

Set the [maximum number of seconds](#) desired.

Definition at line 607 of file CbcModel.hpp.

**7.74.4.48** `double CbcModel::getMaximumSeconds ( ) const` `[inline]`

Get the [maximum number of seconds](#) desired.

Definition at line 614 of file CbcModel.hpp.

**7.74.4.49** `double CbcModel::getCurrentSeconds ( ) const`

Current time since start of branchAndbound.

**7.74.4.50** `bool CbcModel::maximumSecondsReached ( ) const`

Return true if maximum time reached.

**7.74.4.51** `bool CbcModel::setIntegerTolerance ( double value )` `[inline]`

Set the [integrality tolerance](#) .

Definition at line 626 of file CbcModel.hpp.

**7.74.4.52** `double CbcModel::getIntegerTolerance ( ) const` `[inline]`

Get the [integrality tolerance](#) .

Definition at line 632 of file CbcModel.hpp.

**7.74.4.53** `bool CbcModel::setInfeasibilityWeight ( double value )` `[inline]`

Set the [weight per integer infeasibility](#) .

Definition at line 640 of file CbcModel.hpp.

**7.74.4.54** `double CbcModel::getInfeasibilityWeight ( ) const` `[inline]`

Get the [weight per integer infeasibility](#) .

Definition at line 647 of file CbcModel.hpp.

7.74.4.55 `bool CbcModel::setAllowableGap ( double value ) [inline]`

Set the [allowable gap](#) between the best known solution and the best possible solution.

Definition at line 654 of file CbcModel.hpp.

7.74.4.56 `double CbcModel::getAllowableGap ( ) const [inline]`

Get the [allowable gap](#) between the best known solution and the best possible solution.

Definition at line 660 of file CbcModel.hpp.

7.74.4.57 `bool CbcModel::setAllowableFractionGap ( double value ) [inline]`

Set the [fraction allowable gap](#) between the best known solution and the best possible solution.

Definition at line 667 of file CbcModel.hpp.

7.74.4.58 `double CbcModel::getAllowableFractionGap ( ) const [inline]`

Get the [fraction allowable gap](#) between the best known solution and the best possible solution.

Definition at line 673 of file CbcModel.hpp.

7.74.4.59 `bool CbcModel::setAllowablePercentageGap ( double value ) [inline]`

Set the [percentage allowable gap](#) between the best known solution and the best possible solution.

Definition at line 679 of file CbcModel.hpp.

7.74.4.60 `double CbcModel::getAllowablePercentageGap ( ) const [inline]`

Get the [percentage allowable gap](#) between the best known solution and the best possible solution.

Definition at line 685 of file CbcModel.hpp.

7.74.4.61 `bool CbcModel::setHeuristicGap ( double value ) [inline]`

Set the [heuristic gap](#) between the best known solution and the best possible solution.

Definition at line 691 of file CbcModel.hpp.

7.74.4.62 `double CbcModel::getHeuristicGap ( ) const [inline]`

Get the [heuristic gap](#) between the best known solution and the best possible solution.

Definition at line 697 of file CbcModel.hpp.

7.74.4.63 `bool CbcModel::setHeuristicFractionGap ( double value ) [inline]`

Set the [fraction heuristic gap](#) between the best known solution and the best possible solution.

Definition at line 704 of file CbcModel.hpp.

7.74.4.64 `double CbcModel::getHeuristicFractionGap ( ) const [inline]`

Get the [fraction heuristic gap](#) between the best known solution and the best possible solution.

Definition at line 710 of file CbcModel.hpp.

**7.74.4.65** `bool CbcModel::setCutoffIncrement ( double value ) [inline]`

Set the `CbcModel::CbcCutoffIncrement` desired.

Definition at line 717 of file CbcModel.hpp.

**7.74.4.66** `double CbcModel::getCutoffIncrement ( ) const [inline]`

Get the `CbcModel::CbcCutoffIncrement` desired.

Definition at line 724 of file CbcModel.hpp.

**7.74.4.67** `bool CbcModel::canStopOnGap ( ) const`

See if can stop on gap.

**7.74.4.68** `void CbcModel::setHotstartSolution ( const double * solution, const int * priorities = NULL )`

Pass in target solution and optional priorities.

If priorities then  $>0$  means only branch if incorrect while  $<0$  means branch even if correct. +1 or -1 are highest priority

**7.74.4.69** `void CbcModel::setMinimumDrop ( double value ) [inline]`

Set the minimum drop to continue cuts.

Definition at line 737 of file CbcModel.hpp.

**7.74.4.70** `double CbcModel::getMinimumDrop ( ) const [inline]`

Get the minimum drop to continue cuts.

Definition at line 741 of file CbcModel.hpp.

**7.74.4.71** `void CbcModel::setMaximumCutPassesAtRoot ( int value ) [inline]`

Set the maximum number of cut passes at root node (default 20) Minimum drop can also be used for fine tuning.

Definition at line 747 of file CbcModel.hpp.

**7.74.4.72** `int CbcModel::getMaximumCutPassesAtRoot ( ) const [inline]`

Get the maximum number of cut passes at root node.

Definition at line 751 of file CbcModel.hpp.

**7.74.4.73** `void CbcModel::setMaximumCutPasses ( int value ) [inline]`

Set the maximum number of cut passes at other nodes (default 10) Minimum drop can also be used for fine tuning.

Definition at line 757 of file CbcModel.hpp.

**7.74.4.74** `int CbcModel::getMaximumCutPasses ( ) const [inline]`

Get the maximum number of cut passes at other nodes (default 10)

Definition at line 761 of file CbcModel.hpp.

**7.74.4.75** `int CbcModel::getCurrentPassNumber ( ) const [inline]`

Get current cut pass number in this round of cuts.

(1 is first pass)

Definition at line 766 of file CbcModel.hpp.

**7.74.4.76** void CbcModel::setCurrentPassNumber ( int *value* ) [inline]

Set current cut pass number in this round of cuts.

(1 is first pass)

Definition at line 771 of file CbcModel.hpp.

**7.74.4.77** void CbcModel::setNumberStrong ( int *number* )

Set the maximum number of candidates to be evaluated for strong branching.

A value of 0 disables strong branching.

**7.74.4.78** int CbcModel::numberStrong ( ) const [inline]

Get the maximum number of candidates to be evaluated for strong branching.

Definition at line 784 of file CbcModel.hpp.

**7.74.4.79** void CbcModel::setPreferredWay ( int *value* ) [inline]

Set global preferred way to branch -1 down, +1 up, 0 no preference.

Definition at line 789 of file CbcModel.hpp.

**7.74.4.80** int CbcModel::getPreferredWay ( ) const [inline]

Get the preferred way to branch (default 0)

Definition at line 793 of file CbcModel.hpp.

**7.74.4.81** int CbcModel::whenCuts ( ) const [inline]

Get at which depths to do cuts.

Definition at line 797 of file CbcModel.hpp.

**7.74.4.82** void CbcModel::setWhenCuts ( int *value* ) [inline]

Set at which depths to do cuts.

Definition at line 801 of file CbcModel.hpp.

**7.74.4.83** bool CbcModel::doCutsNow ( int *allowForTopOfTree* ) const

Return true if we want to do cuts If allowForTopOfTree zero then just does on multiples of depth if 1 then allows for doing at top of tree if 2 then says if cuts allowed anywhere apart from root.

**7.74.4.84** void CbcModel::setNumberBeforeTrust ( int *number* )

Set the number of branches before pseudo costs believed in dynamic strong branching.

A value of 0 disables dynamic strong branching.

**7.74.4.85** int CbcModel::numberBeforeTrust ( ) const [inline]

get the number of branches before pseudo costs believed in dynamic strong branching.

Definition at line 819 of file CbcModel.hpp.

**7.74.4.86** void CbcModel::setNumberPenalties ( int *number* )

Set the number of variables for which to compute penalties in dynamic strong branching.

A value of 0 disables penalties.

**7.74.4.87** int CbcModel::numberPenalties ( ) const [inline]

get the number of variables for which to compute penalties in dynamic strong branching.

Definition at line 830 of file CbcModel.hpp.

**7.74.4.88** const CbcFullNodeInfo\* CbcModel::topOfTree ( ) const [inline]

Pointer to top of tree.

Definition at line 834 of file CbcModel.hpp.

**7.74.4.89** void CbcModel::setNumberAnalyzeIterations ( int *number* ) [inline]

Number of analyze iterations to do.

Definition at line 837 of file CbcModel.hpp.

**7.74.4.90** int CbcModel::numberAnalyzeIterations ( ) const [inline]

Definition at line 840 of file CbcModel.hpp.

**7.74.4.91** double CbcModel::penaltyScaleFactor ( ) const [inline]

Get scale factor to make penalties match strong.

Should/will be computed

Definition at line 845 of file CbcModel.hpp.

**7.74.4.92** void CbcModel::setPenaltyScaleFactor ( double *value* )

Set scale factor to make penalties match strong.

Should/will be computed

**7.74.4.93** void CbcModel::setProblemType ( int *number* ) [inline]

Problem type as set by user or found by analysis.

This will be extended 0 - not known 1 - Set partitioning <= 2 - Set partitioning == 3 - Set covering 4 - all +- 1 or all +1 and odd

Definition at line 858 of file CbcModel.hpp.

**7.74.4.94** int CbcModel::problemType ( ) const [inline]

Definition at line 861 of file CbcModel.hpp.

**7.74.4.95** int CbcModel::currentDepth ( ) const [inline]

Current depth.

Definition at line 865 of file CbcModel.hpp.

7.74.4.96 void CbcModel::setHowOftenGlobalScan ( int *number* )

Set how often to scan global cuts.

7.74.4.97 int CbcModel::howOftenGlobalScan ( ) const [inline]

Get how often to scan global cuts.

Definition at line 872 of file CbcModel.hpp.

7.74.4.98 int\* CbcModel::originalColumns ( ) const [inline]

Original columns as created by integerPresolve or preprocessing.

Definition at line 876 of file CbcModel.hpp.

7.74.4.99 void CbcModel::setOriginalColumns ( const int \* *originalColumns*, int *numberGood* = COIN\_INT\_MAX )

Set original columns as created by preprocessing.

7.74.4.100 OsiRowCut\* CbcModel::conflictCut ( const OsiSolverInterface \* *solver*, bool & *localCuts* )

Create conflict cut (well - most of)

7.74.4.101 void CbcModel::setPrintFrequency ( int *number* ) [inline]

Set the print frequency.

Controls the number of nodes evaluated between status prints. If *number* <=0 the print frequency is set to 100 nodes for large problems, 1000 for small problems. Print frequency has very slight overhead if small.

Definition at line 892 of file CbcModel.hpp.

7.74.4.102 int CbcModel::printFrequency ( ) const [inline]

Get the print frequency.

Definition at line 896 of file CbcModel.hpp.

7.74.4.103 bool CbcModel::isAbandoned ( ) const

Are there a numerical difficulties?

7.74.4.104 bool CbcModel::isProvenOptimal ( ) const

Is optimality proven?

7.74.4.105 bool CbcModel::isProvenInfeasible ( ) const

Is infeasibility proven (or none better than cutoff)?

7.74.4.106 bool CbcModel::isContinuousUnbounded ( ) const

Was continuous solution unbounded.

7.74.4.107 bool CbcModel::isProvenDualInfeasible ( ) const

Was continuous solution unbounded.

**7.74.4.108** `bool CbcModel::isNodeLimitReached ( ) const`

Node limit reached?

**7.74.4.109** `bool CbcModel::isSecondsLimitReached ( ) const`

Time limit reached?

**7.74.4.110** `bool CbcModel::isSolutionLimitReached ( ) const`

Solution limit reached?

**7.74.4.111** `int CbcModel::getIterationCount ( ) const` `[inline]`

Get how many iterations it took to solve the problem.

Definition at line 921 of file CbcModel.hpp.

**7.74.4.112** `void CbcModel::incrementIterationCount ( int value )` `[inline]`

Increment how many iterations it took to solve the problem.

Definition at line 925 of file CbcModel.hpp.

**7.74.4.113** `int CbcModel::getNodeCount ( ) const` `[inline]`

Get how many Nodes it took to solve the problem (including those in complete fathoming B&B inside CLP).

Definition at line 929 of file CbcModel.hpp.

**7.74.4.114** `void CbcModel::incrementNodeCount ( int value )` `[inline]`

Increment how many nodes it took to solve the problem.

Definition at line 933 of file CbcModel.hpp.

**7.74.4.115** `int CbcModel::getExtraNodeCount ( ) const` `[inline]`

Get how many Nodes were enumerated in complete fathoming B&B inside CLP.

Definition at line 937 of file CbcModel.hpp.

**7.74.4.116** `int CbcModel::status ( ) const` `[inline]`

Final status of problem Some of these can be found out by is.....

functions -1 before branchAndBound 0 finished - check isProvenOptimal or isProvenInfeasible to see if solution found (or check value of best solution) 1 stopped - on maxnodes, maxsols, maxtime 2 difficulties so run was abandoned (5 event user programmed event occurred)

Definition at line 949 of file CbcModel.hpp.

**7.74.4.117** `void CbcModel::setProblemStatus ( int value )` `[inline]`

Definition at line 952 of file CbcModel.hpp.

**7.74.4.118** `int CbcModel::secondaryStatus ( ) const` `[inline]`

Secondary status of problem -1 unset (status\_ will also be -1) 0 search completed with solution 1 linear relaxation not feasible (or worse than cutoff) 2 stopped on gap 3 stopped on nodes 4 stopped on time 5 stopped on user event 6 stopped on solutions 7 linear relaxation unbounded 8 stopped on iteration limit.



Definition at line 967 of file CbcModel.hpp.

**7.74.4.119** `void CbcModel::setSecondaryStatus ( int value ) [inline]`

Definition at line 970 of file CbcModel.hpp.

**7.74.4.120** `bool CbcModel::isInitialSolveAbandoned ( ) const`

Are there numerical difficulties (for initialSolve) ?

**7.74.4.121** `bool CbcModel::isInitialSolveProvenOptimal ( ) const`

Is optimality proven (for initialSolve) ?

**7.74.4.122** `bool CbcModel::isInitialSolveProvenPrimalInfeasible ( ) const`

Is primal infeasibility proven (for initialSolve) ?

**7.74.4.123** `bool CbcModel::isInitialSolveProvenDualInfeasible ( ) const`

Is dual infeasibility proven (for initialSolve) ?

**7.74.4.124** `int CbcModel::numberOfRowsAtContinuous ( ) const [inline]`

Number of rows in continuous (root) problem.

Definition at line 998 of file CbcModel.hpp.

**7.74.4.125** `int CbcModel::getNumCols ( ) const [inline]`

Get number of columns.

Definition at line 1003 of file CbcModel.hpp.

**7.74.4.126** `int CbcModel::getNumRows ( ) const [inline]`

Get number of rows.

Definition at line 1008 of file CbcModel.hpp.

**7.74.4.127** `CoinBigIndex CbcModel::getNumElements ( ) const [inline]`

Get number of nonzero elements.

Definition at line 1013 of file CbcModel.hpp.

**7.74.4.128** `int CbcModel::numberOfIntegers ( ) const [inline]`

Number of integers in problem.

Definition at line 1018 of file CbcModel.hpp.

**7.74.4.129** `const int* CbcModel::integerVariable ( ) const [inline]`

Definition at line 1022 of file CbcModel.hpp.

**7.74.4.130** `char CbcModel::integerType ( int i ) const [inline]`

Whether or not integer.

Definition at line 1026 of file CbcModel.hpp.

**7.74.4.131** `const char* CbcModel::integerType ( ) const [inline]`

Whether or not integer.

Definition at line 1032 of file CbcModel.hpp.

**7.74.4.132** `const double* CbcModel::getColLower ( ) const [inline]`

Get pointer to array[getNumCols()] of column lower bounds.

Definition at line 1037 of file CbcModel.hpp.

**7.74.4.133** `const double* CbcModel::getColUpper ( ) const [inline]`

Get pointer to array[getNumCols()] of column upper bounds.

Definition at line 1042 of file CbcModel.hpp.

**7.74.4.134** `const char* CbcModel::getRowSense ( ) const [inline]`

Get pointer to array[getNumRows()] of row constraint senses.

- 'L':  $\leq$  constraint
- 'E': = constraint
- 'G':  $\geq$  constraint
- 'R': ranged constraint
- 'N': free constraint

Definition at line 1055 of file CbcModel.hpp.

**7.74.4.135** `const double* CbcModel::getRightHandSide ( ) const [inline]`

Get pointer to array[getNumRows()] of rows right-hand sides.

- 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

Definition at line 1067 of file CbcModel.hpp.

**7.74.4.136** `const double* CbcModel::getRowRange ( ) const [inline]`

Get pointer to array[getNumRows()] of row ranges.

- if rowsense()[i] == 'R' then rowrange()[i] == rowupper()[i] - rowlower()[i]
- if rowsense()[i] != 'R' then rowrange()[i] is 0.0

Definition at line 1079 of file CbcModel.hpp.

**7.74.4.137** `const double* CbcModel::getRowLower ( ) const [inline]`

Get pointer to array[getNumRows()] of row lower bounds.

Definition at line 1084 of file CbcModel.hpp.

**7.74.4.138** `const double* CbcModel::getRowUpper ( ) const [inline]`

Get pointer to array[getNumRows()] of row upper bounds.

Definition at line 1089 of file CbcModel.hpp.

**7.74.4.139** `const double* CbcModel::getObjCoefficients ( ) const [inline]`

Get pointer to array[getNumCols()] of objective function coefficients.

Definition at line 1094 of file CbcModel.hpp.

**7.74.4.140** `double CbcModel::getObjSense ( ) const [inline]`

Get objective function sense (1 for min (default), -1 for max)

Definition at line 1099 of file CbcModel.hpp.

**7.74.4.141** `bool CbcModel::isContinuous ( int colIndex ) const [inline]`

Return true if variable is continuous.

Definition at line 1105 of file CbcModel.hpp.

**7.74.4.142** `bool CbcModel::isBinary ( int colIndex ) const [inline]`

Return true if variable is binary.

Definition at line 1110 of file CbcModel.hpp.

**7.74.4.143** `bool CbcModel::isInteger ( int colIndex ) const [inline]`

Return true if column is integer.

Note: This function returns true if the the column is binary or a general integer.

Definition at line 1118 of file CbcModel.hpp.

**7.74.4.144** `bool CbcModel::isIntegerNonBinary ( int colIndex ) const [inline]`

Return true if variable is general integer.

Definition at line 1123 of file CbcModel.hpp.

**7.74.4.145** `bool CbcModel::isFreeBinary ( int colIndex ) const [inline]`

Return true if variable is binary and not fixed at either bound.

Definition at line 1128 of file CbcModel.hpp.

**7.74.4.146** `const CoinPackedMatrix* CbcModel::getMatrixByRow ( ) const [inline]`

Get pointer to row-wise copy of matrix.

Definition at line 1133 of file CbcModel.hpp.

**7.74.4.147** `const CoinPackedMatrix* CbcModel::getMatrixByCol ( ) const` `[inline]`

Get pointer to column-wise copy of matrix.

Definition at line 1138 of file CbcModel.hpp.

**7.74.4.148** `double CbcModel::getInfinity ( ) const` `[inline]`

Get solver's value for infinity.

Definition at line 1143 of file CbcModel.hpp.

**7.74.4.149** `const double* CbcModel::getCbcColLower ( ) const` `[inline]`

Get pointer to array[[getNumCols\(\)](#)] (for speed) of column lower bounds.

Definition at line 1147 of file CbcModel.hpp.

**7.74.4.150** `const double* CbcModel::getCbcColUpper ( ) const` `[inline]`

Get pointer to array[[getNumCols\(\)](#)] (for speed) of column upper bounds.

Definition at line 1151 of file CbcModel.hpp.

**7.74.4.151** `const double* CbcModel::getCbcRowLower ( ) const` `[inline]`

Get pointer to array[[getNumRows\(\)](#)] (for speed) of row lower bounds.

Definition at line 1155 of file CbcModel.hpp.

**7.74.4.152** `const double* CbcModel::getCbcRowUpper ( ) const` `[inline]`

Get pointer to array[[getNumRows\(\)](#)] (for speed) of row upper bounds.

Definition at line 1159 of file CbcModel.hpp.

**7.74.4.153** `const double* CbcModel::getCbcColSolution ( ) const` `[inline]`

Get pointer to array[[getNumCols\(\)](#)] (for speed) of primal solution vector.

Definition at line 1163 of file CbcModel.hpp.

**7.74.4.154** `const double* CbcModel::getCbcRowPrice ( ) const` `[inline]`

Get pointer to array[[getNumRows\(\)](#)] (for speed) of dual prices.

Definition at line 1167 of file CbcModel.hpp.

**7.74.4.155** `const double* CbcModel::getCbcReducedCost ( ) const` `[inline]`

Get a pointer to array[[getNumCols\(\)](#)] (for speed) of reduced costs.

Definition at line 1171 of file CbcModel.hpp.

**7.74.4.156** `const double* CbcModel::getCbcRowActivity ( ) const` `[inline]`

Get pointer to array[[getNumRows\(\)](#)] (for speed) of row activity levels.

Definition at line 1175 of file CbcModel.hpp.

7.74.4.157 `double* CbcModel::continuousSolution ( ) const [inline]`

Holds solution at continuous (after cuts if branchAndBound called)

Definition at line 1184 of file CbcModel.hpp.

7.74.4.158 `int* CbcModel::usedInSolution ( ) const [inline]`

Array marked whenever a solution is found if non-zero.

Code marks if heuristic returns better so heuristic need only mark if it wants to on solutions which are worse than current

Definition at line 1191 of file CbcModel.hpp.

7.74.4.159 `void CbcModel::incrementUsed ( const double * solution )`

Increases usedInSolution for nonzeros.

7.74.4.160 `void CbcModel::setBestSolution ( CBC_Message how, double & objectiveValue, const double * solution, int fixVariables = 0 )`

Record a new incumbent solution and update objectiveValue.

7.74.4.161 `void CbcModel::setBestObjectiveValue ( double objectiveValue )`

Just update objectiveValue.

7.74.4.162 `CbcEventHandler::CbcAction CbcModel::dealWithEventHandler ( CbcEventHandler::CbcEvent event, double objValue, const double * solution )`

Deals with event handler and solution.

7.74.4.163 `virtual double CbcModel::checkSolution ( double cutoff, double * solution, int fixVariables, double originalObjValue ) [virtual]`

Call this to really test if a valid solution can be feasible Solution is number columns in size.

If fixVariables true then bounds of continuous solver updated. Returns objective value (worse than cutoff if not feasible)

Previously computed objective value is now passed in (in case user does not do solve) virtual so user can override

7.74.4.164 `bool CbcModel::feasibleSolution ( int & numberIntegerInfeasibilities, int & numberObjectInfeasibilities ) const`

Test the current solution for feasibility.

Scan all objects for indications of infeasibility. This is broken down into simple integer infeasibility (numberIntegerInfeasibilities) and all other reports of infeasibility (numberObjectInfeasibilities).

7.74.4.165 `double* CbcModel::currentSolution ( ) const [inline]`

Solution to the most recent lp relaxation.

The solver's solution to the most recent lp relaxation.

Definition at line 1230 of file CbcModel.hpp.

7.74.4.166 `const double* CbcModel::testSolution ( ) const [inline]`

For testing infeasibilities - will point to currentSolution\_ or solver->getColSolution()

Definition at line 1236 of file CbcModel.hpp.

**7.74.4.167** `void CbcModel::setTestSolution ( const double * solution )` `[inline]`

Definition at line 1239 of file CbcModel.hpp.

**7.74.4.168** `void CbcModel::reserveCurrentSolution ( const double * solution = NULL )`

Make sure region there and optionally copy solution.

**7.74.4.169** `const double* CbcModel::getColSolution ( ) const` `[inline]`

Get pointer to array[getNumCols()] of primal solution vector.

Definition at line 1246 of file CbcModel.hpp.

**7.74.4.170** `const double* CbcModel::getRowPrice ( ) const` `[inline]`

Get pointer to array[getNumRows()] of dual prices.

Definition at line 1251 of file CbcModel.hpp.

**7.74.4.171** `const double* CbcModel::getReducedCost ( ) const` `[inline]`

Get a pointer to array[getNumCols()] of reduced costs.

Definition at line 1256 of file CbcModel.hpp.

**7.74.4.172** `const double* CbcModel::getRowActivity ( ) const` `[inline]`

Get pointer to array[getNumRows()] of row activity levels.

Definition at line 1261 of file CbcModel.hpp.

**7.74.4.173** `double CbcModel::getCurrentObjValue ( ) const` `[inline]`

Get current objective function value.

Definition at line 1266 of file CbcModel.hpp.

**7.74.4.174** `double CbcModel::getCurrentMinimizationObjValue ( ) const` `[inline]`

Get current minimization objective function value.

Definition at line 1270 of file CbcModel.hpp.

**7.74.4.175** `double CbcModel::getMinimizationObjValue ( ) const` `[inline]`

Get best objective function value as minimization.

Definition at line 1275 of file CbcModel.hpp.

**7.74.4.176** `void CbcModel::setMinimizationObjValue ( double value )` `[inline]`

Set best objective function value as minimization.

Definition at line 1279 of file CbcModel.hpp.

**7.74.4.177** `double CbcModel::getObjValue ( ) const` `[inline]`

Get best objective function value.

Definition at line 1284 of file CbcModel.hpp.

**7.74.4.178** `double CbcModel::getBestPossibleObjValue ( ) const`

Get best possible objective function value.

This is better of best possible left on tree and best solution found. If called from within branch and cut may be optimistic.

**7.74.4.179** `void CbcModel::setObjValue ( double value ) [inline]`

Set best objective function value.

Definition at line 1294 of file CbcModel.hpp.

**7.74.4.180** `double CbcModel::getSolverObjValue ( ) const [inline]`

Get solver objective function value (as minimization)

Definition at line 1298 of file CbcModel.hpp.

**7.74.4.181** `double* CbcModel::bestSolution ( ) const [inline]`

The best solution to the integer programming problem.

The best solution to the integer programming problem found during the search. If no solution is found, the method returns null.

Definition at line 1308 of file CbcModel.hpp.

**7.74.4.182** `void CbcModel::setBestSolution ( const double * solution, int numberColumns, double objectiveValue, bool check = false )`

User callable setBestSolution.

If check false does not check valid If true then sees if feasible and warns if objective value worse than given (so just set to COIN\_DBL\_MAX if you don't care). If check true then does not save solution if not feasible

**7.74.4.183** `int CbcModel::getSolutionCount ( ) const [inline]`

Get number of solutions.

Definition at line 1321 of file CbcModel.hpp.

**7.74.4.184** `void CbcModel::setSolutionCount ( int value ) [inline]`

Set number of solutions (so heuristics will be different)

Definition at line 1326 of file CbcModel.hpp.

**7.74.4.185** `int CbcModel::numberSavedSolutions ( ) const`

Number of saved solutions (including best)

**7.74.4.186** `int CbcModel::maximumSavedSolutions ( ) const [inline]`

Maximum number of extra saved solutions.

Definition at line 1332 of file CbcModel.hpp.

**7.74.4.187** `void CbcModel::setMaximumSavedSolutions ( int value )`

Set maximum number of extra saved solutions.

**7.74.4.188** `const double* CbcModel::savedSolution ( int which ) const`

Return a saved solution (0==best) - NULL if off end.

**7.74.4.189** `double CbcModel::savedSolutionObjective ( int which ) const`

Return a saved solution objective (0==best) - COIN\_DBL\_MAX if off end.

**7.74.4.190** `void CbcModel::deleteSavedSolution ( int which )`

Delete a saved solution and move others up.

**7.74.4.191** `int CbcModel::phase ( ) const` `[inline]`

Current phase (so heuristics etc etc can find out).

0 - initial solve 1 - solve with cuts at root 2 - solve with cuts 3 - other e.g. strong branching 4 - trying to validate a solution  
5 - at end of search

Definition at line 1352 of file CbcModel.hpp.

**7.74.4.192** `int CbcModel::getNumberHeuristicSolutions ( ) const` `[inline]`

Get number of heuristic solutions.

Definition at line 1357 of file CbcModel.hpp.

**7.74.4.193** `void CbcModel::setNumberHeuristicSolutions ( int value )` `[inline]`

Set number of heuristic solutions.

Definition at line 1361 of file CbcModel.hpp.

**7.74.4.194** `void CbcModel::setObjSense ( double s )` `[inline]`

Set objective function sense (1 for min (default), -1 for max,)

Definition at line 1366 of file CbcModel.hpp.

**7.74.4.195** `double CbcModel::getContinuousObjective ( ) const` `[inline]`

Value of objective at continuous.

Definition at line 1372 of file CbcModel.hpp.

**7.74.4.196** `void CbcModel::setContinuousObjective ( double value )` `[inline]`

Definition at line 1375 of file CbcModel.hpp.

**7.74.4.197** `int CbcModel::getContinuousInfeasibilities ( ) const` `[inline]`

Number of infeasibilities at continuous.

Definition at line 1379 of file CbcModel.hpp.

**7.74.4.198** `void CbcModel::setContinuousInfeasibilities ( int value )` `[inline]`

Definition at line 1382 of file CbcModel.hpp.



**7.74.4.199** `double CbcModel::rootObjectiveAfterCuts ( ) const [inline]`

Value of objective after root node cuts added.

Definition at line 1386 of file CbcModel.hpp.

**7.74.4.200** `double CbcModel::sumChangeObjective ( ) const [inline]`

Sum of Changes to objective by first solve.

Definition at line 1390 of file CbcModel.hpp.

**7.74.4.201** `int CbcModel::numberGlobalViolations ( ) const [inline]`

Number of times global cuts violated.

When global cut pool then this should be kept for each cut and type of cut

Definition at line 1395 of file CbcModel.hpp.

**7.74.4.202** `void CbcModel::clearNumberGlobalViolations ( ) [inline]`

Definition at line 1398 of file CbcModel.hpp.

**7.74.4.203** `bool CbcModel::resolveAfterTakeOffCuts ( ) const [inline]`

Whether to force a resolve after takeOffCuts.

Definition at line 1402 of file CbcModel.hpp.

**7.74.4.204** `void CbcModel::setResolveAfterTakeOffCuts ( bool yesNo ) [inline]`

Definition at line 1405 of file CbcModel.hpp.

**7.74.4.205** `int CbcModel::maximumRows ( ) const [inline]`

Maximum number of rows.

Definition at line 1409 of file CbcModel.hpp.

**7.74.4.206** `CoinWarmStartBasis& CbcModel::workingBasis ( ) [inline]`

Work basis for temporary use.

Definition at line 1413 of file CbcModel.hpp.

**7.74.4.207** `int CbcModel::getStopNumberIterations ( ) const [inline]`

Get number of "iterations" to stop after.

Definition at line 1417 of file CbcModel.hpp.

**7.74.4.208** `void CbcModel::setStopNumberIterations ( int value ) [inline]`

Set number of "iterations" to stop after.

Definition at line 1421 of file CbcModel.hpp.

**7.74.4.209** `CbcModel* CbcModel::heuristicModel ( ) const [inline]`

A pointer to model from [CbcHeuristic](#).

Definition at line 1425 of file CbcModel.hpp.

**7.74.4.210** `void CbcModel::setHeuristicModel ( CbcModel * model ) [inline]`

Set a pointer to model from [CbcHeuristic](#).

Definition at line 1428 of file CbcModel.hpp.

**7.74.4.211** `CbcCompareBase* CbcModel::nodeComparison ( ) const [inline]`

Definition at line 1435 of file CbcModel.hpp.

**7.74.4.212** `void CbcModel::setNodeComparison ( CbcCompareBase * compare )`

**7.74.4.213** `void CbcModel::setNodeComparison ( CbcCompareBase & compare )`

**7.74.4.214** `CbcFeasibilityBase* CbcModel::problemFeasibility ( ) const [inline]`

Definition at line 1445 of file CbcModel.hpp.

**7.74.4.215** `void CbcModel::setProblemFeasibility ( CbcFeasibilityBase * feasibility )`

**7.74.4.216** `void CbcModel::setProblemFeasibility ( CbcFeasibilityBase & feasibility )`

**7.74.4.217** `CbcTree* CbcModel::tree ( ) const [inline]`

Tree method e.g. heap (which may be overridden by inheritance)

Definition at line 1455 of file CbcModel.hpp.

**7.74.4.218** `void CbcModel::passInTreeHandler ( CbcTree & tree )`

For modifying tree handling (original is cloned)

**7.74.4.219** `void CbcModel::passInSubTreeModel ( CbcModel & model )`

For passing in an [CbcModel](#) to do a sub Tree (with derived tree handlers).

Passed in model must exist for duration of branch and bound

**7.74.4.220** `CbcModel* CbcModel::subTreeModel ( OsiSolverInterface * solver = NULL ) const`

For retrieving a copy of subtree model with given OsiSolver.

If no subtree model will use self (up to user to reset cutoff etc). If solver NULL uses current

**7.74.4.221** `int CbcModel::numberStoppedSubTrees ( ) const [inline]`

Returns number of times any subtree stopped on nodes, time etc.

Definition at line 1470 of file CbcModel.hpp.

**7.74.4.222** `void CbcModel::incrementSubTreeStopped ( ) [inline]`

Says a sub tree was stopped.

Definition at line 1474 of file CbcModel.hpp.

**7.74.4.223** `int CbcModel::typePresolve ( ) const [inline]`

Whether to automatically do presolve before branch and bound (subTrees).

0 - no 1 - ordinary presolve 2 - integer presolve (dodgy)

Definition at line 1482 of file CbcModel.hpp.

**7.74.4.224** `void CbcModel::setTypePresolve ( int value ) [inline]`

Definition at line 1485 of file CbcModel.hpp.

**7.74.4.225** `CbcBranchDecision* CbcModel::branchingMethod ( ) const [inline]`

Get the current branching decision method.

Definition at line 1498 of file CbcModel.hpp.

**7.74.4.226** `void CbcModel::setBranchingMethod ( CbcBranchDecision * method ) [inline]`

Set the branching decision method.

Definition at line 1502 of file CbcModel.hpp.

**7.74.4.227** `void CbcModel::setBranchingMethod ( CbcBranchDecision & method ) [inline]`

Set the branching method.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Definition at line 1510 of file CbcModel.hpp.

**7.74.4.228** `CbcCutModifier* CbcModel::cutModifier ( ) const [inline]`

Get the current cut modifier method.

Definition at line 1515 of file CbcModel.hpp.

**7.74.4.229** `void CbcModel::setCutModifier ( CbcCutModifier * modifier )`

Set the cut modifier method.

**7.74.4.230** `void CbcModel::setCutModifier ( CbcCutModifier & modifier )`

Set the cut modifier method.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**7.74.4.231** `int CbcModel::stateOfSearch ( ) const [inline]`

State of search 0 - no solution 1 - only heuristic solutions 2 - branched to a solution 3 - no solution but many nodes.

Definition at line 1536 of file CbcModel.hpp.

**7.74.4.232** `void CbcModel::setStateOfSearch ( int state ) [inline]`

Definition at line 1539 of file CbcModel.hpp.

7.74.4.233 `int CbcModel::searchStrategy ( ) const [inline]`

Strategy worked out - mainly at root node for use by [CbcNode](#).

Definition at line 1543 of file CbcModel.hpp.

7.74.4.234 `void CbcModel::setSearchStrategy ( int value ) [inline]`

Set strategy worked out - mainly at root node for use by [CbcNode](#).

Definition at line 1547 of file CbcModel.hpp.

7.74.4.235 `int CbcModel::strongStrategy ( ) const [inline]`

Strong branching strategy.

Definition at line 1551 of file CbcModel.hpp.

7.74.4.236 `void CbcModel::setStrongStrategy ( int value ) [inline]`

Set strong branching strategy.

Definition at line 1555 of file CbcModel.hpp.

7.74.4.237 `int CbcModel::numberCutGenerators ( ) const [inline]`

Get the number of cut generators.

Definition at line 1560 of file CbcModel.hpp.

7.74.4.238 `CbcCutGenerator** CbcModel::cutGenerators ( ) const [inline]`

Get the list of cut generators.

Definition at line 1564 of file CbcModel.hpp.

7.74.4.239 `CbcCutGenerator* CbcModel::cutGenerator ( int i ) const [inline]`

Get the specified cut generator.

Definition at line 1568 of file CbcModel.hpp.

7.74.4.240 `CbcCutGenerator* CbcModel::virginCutGenerator ( int i ) const [inline]`

Get the specified cut generator before any changes.

Definition at line 1572 of file CbcModel.hpp.

7.74.4.241 `void CbcModel::addCutGenerator ( CglCutGenerator * generator, int howOften = 1, const char * name = NULL, bool normal = true, bool atSolution = false, bool infeasible = false, int howOftenInSub = -100, int whatDepth = -1, int whatDepthInSub = -1 )`

Add one generator - up to user to delete generators.

howoften affects how generator is used. 0 or 1 means always, >1 means every that number of nodes. Negative values have same meaning as positive but they may be switched off (-> -100) by code if not many cuts generated at continuous. -99 is just done at root. Name is just for printout. If depth >0 overrides how often generator is called (if howOften== -1 or >0).

**7.74.4.242** **CbcStrategy\*** CbcModel::strategy ( ) const [inline]

Get the current strategy.

Definition at line 1596 of file CbcModel.hpp.

**7.74.4.243** void CbcModel::setStrategy ( CbcStrategy & strategy )

Set the strategy. Clones.

**7.74.4.244** void CbcModel::setStrategy ( CbcStrategy \* strategy ) [inline]

Set the strategy. assigns.

Definition at line 1602 of file CbcModel.hpp.

**7.74.4.245** **CbcModel\*** CbcModel::parentModel ( ) const [inline]

Get the current parent model.

Definition at line 1606 of file CbcModel.hpp.

**7.74.4.246** void CbcModel::setParentModel ( CbcModel & parentModel ) [inline]

Set the parent model.

Definition at line 1610 of file CbcModel.hpp.

**7.74.4.247** void CbcModel::addHeuristic ( CbcHeuristic \* generator, const char \* name = NULL, int before = -1 )

Add one heuristic - up to user to delete.

The name is just used for print messages.

**7.74.4.248** **CbcHeuristic\*** CbcModel::heuristic ( int i ) const [inline]

Get the specified heuristic.

Definition at line 1625 of file CbcModel.hpp.

**7.74.4.249** int CbcModel::numberHeuristics ( ) const [inline]

Get the number of heuristics.

Definition at line 1629 of file CbcModel.hpp.

**7.74.4.250** void CbcModel::setNumberHeuristics ( int value ) [inline]

Set the number of heuristics.

Definition at line 1633 of file CbcModel.hpp.

**7.74.4.251** **CbcHeuristic\*** CbcModel::lastHeuristic ( ) const [inline]

Pointer to heuristic solver which found last solution (or NULL)

Definition at line 1637 of file CbcModel.hpp.

**7.74.4.252** void CbcModel::setLastHeuristic ( CbcHeuristic \* last ) [inline]

set last heuristic which found a solution

Definition at line 1641 of file CbcModel.hpp.

**7.74.4.253** void CbcModel::passInPriorities ( const int \* *priorities*, bool *ifNotSimpleIntegers* )

Pass in branching priorities.

If *ifClique* then priorities are on cliques otherwise priorities are on integer variables. Other type (if exists set to default) 1 is highest priority. (well actually -INT\_MAX is but that's ugly) If *hotstart* > 0 then branches are created to force the variable to the value given by best solution. This enables a sort of hot start. The node choice should be greatest depth and *hotstart* should normally be switched off after a solution.

If *ifNotSimpleIntegers* true then appended to normal integers

This is now deprecated except for simple usage. If user creates Cbcobjects then set priority in them

**7.74.4.254** int CbcModel::priority ( int *sequence* ) const [inline]

Returns priority level for an object (or 1000 if no priorities exist)

Definition at line 1666 of file CbcModel.hpp.

**7.74.4.255** void CbcModel::passInEventHandler ( const CbcEventHandler \* *eventHandler* )

Set an event handler.

A clone of the handler passed as a parameter is stored in [CbcModel](#).

**7.74.4.256** CbcEventHandler\* CbcModel::getEventHandler ( ) const [inline]

Retrieve a pointer to the event handler.

Definition at line 1677 of file CbcModel.hpp.

**7.74.4.257** void CbcModel::setApplicationData ( void \* *appData* )

Set application data.

This is a pointer that the application can store into and retrieve from the solver interface. This field is available for the application to optionally define and use.

**7.74.4.258** void\* CbcModel::getApplicationData ( ) const

Get application data.

**7.74.4.259** void CbcModel::passInSolverCharacteristics ( OsiBabSolver \* *solverCharacteristics* )

For advanced applications you may wish to modify the behavior of Cbc e.g.

if the solver is a NLP solver then you may not have an exact optimum solution at each step. Information could be built into OsiSolverInterface but this is an alternative so that that interface does not have to be changed. If something similar is useful to enough solvers then it could be migrated You can also pass in by using *solver->setAuxiliaryInfo*. You should do that if solver is odd - if solver is normal simplex then use this. NOTE - characteristics are not cloned

**7.74.4.260** const OsiBabSolver\* CbcModel::solverCharacteristics ( ) const [inline]

Get solver characteristics.

Definition at line 1710 of file CbcModel.hpp.

7.74.4.261 void CbcModel::passInMessageHandler ( CoinMessageHandler \* *handler* )

Pass in Message handler (not deleted at end)

7.74.4.262 void CbcModel::newLanguage ( CoinMessages::Language *language* )

Set language.

7.74.4.263 void CbcModel::setLanguage ( CoinMessages::Language *language* ) [inline]

Definition at line 1723 of file CbcModel.hpp.

7.74.4.264 CoinMessageHandler\* CbcModel::messageHandler ( ) const [inline]

Return handler.

Definition at line 1727 of file CbcModel.hpp.

7.74.4.265 CoinMessages& CbcModel::messages ( ) [inline]

Return messages.

Definition at line 1731 of file CbcModel.hpp.

7.74.4.266 CoinMessages\* CbcModel::messagesPointer ( ) [inline]

Return pointer to messages.

Definition at line 1735 of file CbcModel.hpp.

7.74.4.267 void CbcModel::setLogLevel ( int *value* )

Set log level.

7.74.4.268 int CbcModel::logLevel ( ) const [inline]

Get log level.

Definition at line 1741 of file CbcModel.hpp.

7.74.4.269 void CbcModel::setDefaultHandler ( bool *yesNo* ) [inline]

Set flag to say if handler\_ is the default handler.

The default handler is deleted when the model is deleted. Other handlers (supplied by the client) will not be deleted.

Definition at line 1749 of file CbcModel.hpp.

7.74.4.270 bool CbcModel::defaultHandler ( ) const [inline]

Check default handler.

Definition at line 1753 of file CbcModel.hpp.

7.74.4.271 void CbcModel::setSpecialOptions ( int *value* ) [inline]

Set special options 0 bit (1) - check if cuts valid (if on debugger list) 1 bit (2) - use current basis to check integer solution (rather than all slack) 2 bit (4) - don't check integer solution (by solving LP) 3 bit (8) - fast analyze 4 bit (16) - non-linear model - so no well defined CoinPackedMatrix 5 bit (32) - keep names 6 bit (64) - try for dominated columns 7 bit (128) - SOS type 1 but all declared integer 8 bit (256) - Set to say solution just found, unset by doing cuts 9 bit (512) - Try reduced model after 100 nodes 10 bit (1024) - Switch on some heuristics even if seems unlikely 11 bit (2048) - Mark as

in small branch and bound 12 bit (4096) - Funny cuts so do slow way (in some places) 13 bit (8192) - Funny cuts so do slow way (in other places) 14 bit (16384) - Use Cplex! for fathoming 15 bit (32768) - Try reduced model after 0 nodes 16 bit (65536) - Original model had integer bounds 17 bit (131072) - Perturbation switched off 18 bit (262144) - donor [CbcModel](#) 19 bit (524288) - recipient [CbcModel](#) 20 bit (1048576) - waiting for sub model to return 22 bit (4194304) - do not initialize random seed in solver (user has) 23 bit (8388608) - leave solver\_ with cuts 24 bit (16777216) - just get feasible if no cutoff.

Definition at line 1788 of file CbcModel.hpp.

**7.74.4.272** `int CbcModel::specialOptions ( ) const [inline]`

Get special options.

Definition at line 1792 of file CbcModel.hpp.

**7.74.4.273** `void CbcModel::setRandomSeed ( int value ) [inline]`

Set random seed.

Definition at line 1796 of file CbcModel.hpp.

**7.74.4.274** `int CbcModel::getRandomSeed ( ) const [inline]`

Get random seed.

Definition at line 1800 of file CbcModel.hpp.

**7.74.4.275** `void CbcModel::setMultipleRootTries ( int value ) [inline]`

Set multiple root tries.

Definition at line 1804 of file CbcModel.hpp.

**7.74.4.276** `int CbcModel::getMultipleRootTries ( ) const [inline]`

Get multiple root tries.

Definition at line 1808 of file CbcModel.hpp.

**7.74.4.277** `void CbcModel::sayEventHappened ( ) [inline]`

Tell model to stop on event.

Definition at line 1812 of file CbcModel.hpp.

**7.74.4.278** `bool CbcModel::normalSolver ( ) const [inline]`

Says if normal solver i.e. has well defined CoinPackedMatrix.

Definition at line 1815 of file CbcModel.hpp.

**7.74.4.279** `bool CbcModel::waitingForMiniBranchAndBound ( ) const [inline]`

Says if model is sitting there waiting for mini branch and bound to finish This is because an event handler may only have access to parent model in mini branch and bound.

Definition at line 1822 of file CbcModel.hpp.



**7.74.4.280** void CbcModel::setMoreSpecialOptions ( int *value* ) [inline]

Set more special options at present bottom 6 bits used for shadow price mode 1024 for experimental hotstart 2048,4096 breaking out of cuts 8192 slowly increase minimum drop 16384 gomory 32768 more heuristics in sub trees 65536 no cuts in preprocessing 131072 Time limits elapsed 18 bit (262144) - Perturb fathom nodes 19 bit (524288) - No limit on fathom nodes 20 bit (1048576) - Reduce sum of infeasibilities before cuts 21 bit (2097152) - Reduce sum of infeasibilities after cuts 22 bit (4194304) - Conflict analysis 23 bit (8388608) - Conflict analysis - temporary bit 24 bit (16777216) - Add cutoff as LP constraint (out) 25 bit (33554432) - diving/reordering 26 bit (67108864) - load global cuts from file 27 bit (134217728) - append binding global cuts to file 28 bit (268435456) - idiot branching 29 bit (536870912) - don't make fake objective.

Definition at line 1847 of file CbcModel.hpp.

**7.74.4.281** int CbcModel::moreSpecialOptions ( ) const [inline]

Get more special options.

Definition at line 1851 of file CbcModel.hpp.

**7.74.4.282** void CbcModel::setMoreSpecialOptions2 ( int *value* ) [inline]

Set more more special options 0 bit (1) - find switching variables 1 bit (2) - using fake objective until solution 2 bit (4) - switching variables exist 3 bit (8) - skip most of setBestSolution checks 4 bit (16) - very lightweight preprocessing in smallB&B 5 bit (32) - event handler needs to be cloned when parallel.

Definition at line 1862 of file CbcModel.hpp.

**7.74.4.283** int CbcModel::moreSpecialOptions2 ( ) const [inline]

Get more special options2.

Definition at line 1866 of file CbcModel.hpp.

**7.74.4.284** void CbcModel::setCutoffAsConstraint ( bool *yesNo* ) [inline]

Set cutoff as constraint.

Definition at line 1870 of file CbcModel.hpp.

**7.74.4.285** void CbcModel::setUseElapsedTime ( bool *yesNo* ) [inline]

Set time method.

Definition at line 1874 of file CbcModel.hpp.

**7.74.4.286** bool CbcModel::useElapsedTime ( ) const [inline]

Get time method.

Definition at line 1881 of file CbcModel.hpp.

**7.74.4.287** void\* CbcModel::temporaryPointer ( ) const [inline]

Get useful temporary pointer.

Definition at line 1885 of file CbcModel.hpp.

**7.74.4.288** void CbcModel::setTemporaryPointer ( void \* *pointer* ) [inline]

Set useful temporary pointer.

Definition at line 1888 of file CbcModel.hpp.

**7.74.4.289** `void CbcModel::goToDantzig ( int numberNodes, ClpDualRowPivot *& savePivotMethod )`

Go to dantzig pivot selection if easy problem (clp only)

**7.74.4.290** `bool CbcModel::ownObjects ( ) const [inline]`

Now we may not own objects - just point to solver's objects.

Definition at line 1893 of file CbcModel.hpp.

**7.74.4.291** `void CbcModel::checkModel ( )`

Check original model before it gets messed up.

**7.74.4.292** `void CbcModel::assignSolver ( OsiSolverInterface *& solver, bool deleteSolver = true )`

Assign a solver to the model (model assumes ownership)

On return, `solver` will be NULL. If `deleteSolver` then current solver deleted (if model owned)

#### Note

Parameter settings in the outgoing solver are not inherited by the incoming solver.

**7.74.4.293** `void CbcModel::setModelOwnsSolver ( bool ourSolver ) [inline]`

Set ownership of solver.

A parameter of false tells [CbcModel](#) it does not own the solver and should not delete it. Once you claim ownership of the solver, you're responsible for eventually deleting it. Note that [CbcModel](#) clones solvers with abandon. Unless you have a deep understanding of the workings of [CbcModel](#), the only time you want to claim ownership is when you're about to delete the [CbcModel](#) object but want the solver to continue to exist (as, for example, when `branchAndBound` has finished and you want to hang on to the answer).

Definition at line 1930 of file CbcModel.hpp.

**7.74.4.294** `bool CbcModel::modelOwnsSolver ( ) [inline]`

Get ownership of solver.

A return value of true means that [CbcModel](#) owns the solver and will take responsibility for deleting it when that becomes necessary.

Definition at line 1939 of file CbcModel.hpp.

**7.74.4.295** `virtual CbcModel* CbcModel::clone ( bool cloneHandler ) [virtual]`

Clone.

**7.74.4.296** `CbcModel& CbcModel::operator= ( const CbcModel & rhs )`

Assignment operator.

**7.74.4.297** `OsiSolverInterface* CbcModel::solver ( ) const [inline]`

Returns solver - has current state.

Definition at line 1958 of file CbcModel.hpp.

7.74.4.298 `OsiSolverInterface* CbcModel::swapSolver ( OsiSolverInterface * solver )` `[inline]`

Returns current solver - sets new one.

Definition at line 1963 of file CbcModel.hpp.

7.74.4.299 `OsiSolverInterface* CbcModel::continuousSolver ( ) const` `[inline]`

Returns solver with continuous state.

Definition at line 1970 of file CbcModel.hpp.

7.74.4.300 `void CbcModel::createContinuousSolver ( )` `[inline]`

Create solver with continuous state.

Definition at line 1975 of file CbcModel.hpp.

7.74.4.301 `void CbcModel::clearContinuousSolver ( )` `[inline]`

Clear solver with continuous state.

Definition at line 1979 of file CbcModel.hpp.

7.74.4.302 `OsiSolverInterface* CbcModel::referenceSolver ( ) const` `[inline]`

A copy of the solver, taken at constructor or by saveReferenceSolver.

Definition at line 1985 of file CbcModel.hpp.

7.74.4.303 `void CbcModel::saveReferenceSolver ( )`

Save a copy of the current solver so can be reset to.

7.74.4.304 `void CbcModel::resetToReferenceSolver ( )`

Uses a copy of reference solver to be current solver.

Because of possible mismatches all exotic integer information is lost (apart from normal information in OsiSolverInterface) so SOS etc and priorities will have to be redone

7.74.4.305 `void CbcModel::gutsOfDestructor ( )`

Clears out as much as possible (except solver)

7.74.4.306 `void CbcModel::gutsOfDestructor2 ( )`

Clears out enough to reset [CbcModel](#) as if no branch and bound done.

7.74.4.307 `void CbcModel::resetModel ( )`

Clears out enough to reset [CbcModel](#) cutoff etc.

7.74.4.308 `void CbcModel::gutsOfCopy ( const CbcModel & rhs, int mode = 0 )`

Most of copy constructor mode - 0 copy but don't delete before 1 copy and delete before 2 copy and delete before (but use virgin generators)

7.74.4.309 `void CbcModel::moveInfo ( const CbcModel & rhs )`

Move status, nodes etc etc across.

7.74.4.310 `static bool CbcModel::haveMultiThreadSupport ( ) [static]`

Indicates whether Cbc library has been compiled with multithreading support.

7.74.4.311 `CbcThread* CbcModel::masterThread ( ) const [inline]`

Get pointer to masterthread.

Definition at line 2022 of file CbcModel.hpp.

7.74.4.312 `CbcNodeInfo** CbcModel::walkback ( ) const [inline]`

Get pointer to walkback.

Definition at line 2026 of file CbcModel.hpp.

7.74.4.313 `int CbcModel::getNumberThreads ( ) const [inline]`

Get number of threads.

Definition at line 2030 of file CbcModel.hpp.

7.74.4.314 `void CbcModel::setNumberThreads ( int value ) [inline]`

Set number of threads.

Definition at line 2034 of file CbcModel.hpp.

7.74.4.315 `int CbcModel::getThreadMode ( ) const [inline]`

Get thread mode.

Definition at line 2038 of file CbcModel.hpp.

7.74.4.316 `void CbcModel::setThreadMode ( int value ) [inline]`

Set thread mode always use numberThreads for branching 1 set then deterministic 2 set then use numberThreads for root cuts 4 set then use numberThreads in root mini branch and bound 8 set and numberThreads - do heuristics numberThreads at a time 8 set and numberThreads==0 do all heuristics at once default is 0.

Definition at line 2050 of file CbcModel.hpp.

7.74.4.317 `int CbcModel::parallelMode ( ) const [inline]`

Return -2 if deterministic threaded and main thread -1 if deterministic threaded and serial thread 0 if serial 1 if opportunistic threaded.

Definition at line 2059 of file CbcModel.hpp.

7.74.4.318 `bool CbcModel::isLocked ( ) const`

From here to end of section - code in CbcThread.cpp until class changed Returns true if locked.

7.74.4.319 `void CbcModel::lockThread ( ) [inline]`

Definition at line 2087 of file CbcModel.hpp.

7.74.4.320 void CbcModel::unlockThread ( ) [inline]

Definition at line 2088 of file CbcModel.hpp.

7.74.4.321 void CbcModel::setInfoInChild ( int *type*, CbcThread \* *info* )

Set information in a child -3 pass pointer to child thread info -2 just stop -1 delete simple child stuff 0 delete opportunistic child stuff 1 delete deterministic child stuff.

7.74.4.322 void CbcModel::moveToModel ( CbcModel \* *baseModel*, int *mode* )

Move/copy information from one model to another -1 - initialization 0 - from base model 1 - to base model (and reset) 2 - add in final statistics etc (and reset so can do clean destruction)

7.74.4.323 int CbcModel::splitModel ( int *numberModels*, CbcModel \*\* *model*, int *numberNodes* )

Split up nodes.

7.74.4.324 void CbcModel::startSplitModel ( int *numberIterations* )

Start threads.

7.74.4.325 void CbcModel::mergeModels ( int *numberModel*, CbcModel \*\* *model*, int *numberNodes* )

Merge models.

7.74.4.326 int CbcModel::getNodeCount2 ( ) const [inline]

Get how many Nodes it took to solve the problem.

Definition at line 2118 of file CbcModel.hpp.

7.74.4.327 void CbcModel::setPointers ( const OsiSolverInterface \* *solver* )

Set pointers for speed.

7.74.4.328 int CbcModel::reducedCostFix ( )

Perform reduced cost fixing.

Fixes integer variables at their current value based on reduced cost penalties. Returns number fixed

7.74.4.329 void CbcModel::synchronizeHandlers ( int *makeDefault* )

Makes all handlers same.

If makeDefault 1 then makes top level default and rest point to that. If 2 then each is copy

7.74.4.330 void CbcModel::saveExtraSolution ( const double \* *solution*, double *objectiveValue* )

Save a solution to saved list.

7.74.4.331 void CbcModel::saveBestSolution ( const double \* *solution*, double *objectiveValue* )

Save a solution to best and move current to saved.

7.74.4.332 void CbcModel::deleteSolutions ( )

Delete best and saved solutions.

7.74.4.333 `int CbcModel::resolve ( OsiSolverInterface * solver )`

Encapsulates solver resolve.

7.74.4.334 `int CbcModel::chooseBranch ( CbcNode *& newNode, int numberPassesLeft, CbcNode * oldNode, OsiCuts & cuts, bool & resolved, CoinWarmStartBasis * lastws, const double * lowerBefore, const double * upperBefore, OsiSolverBranch *& branches )`

Encapsulates choosing a variable - anyAction -2, infeasible (-1 round again), 0 done.

7.74.4.335 `int CbcModel::chooseBranch ( CbcNode * newNode, int numberPassesLeft, bool & resolved )`

7.74.4.336 `CoinWarmStartBasis* CbcModel::getEmptyBasis ( int ns = 0, int na = 0 ) const`

Return an empty basis object of the specified size.

A useful utility when constructing a basis for a subproblem from scratch. The object returned will be of the requested capacity and appropriate for the solver attached to the model.

7.74.4.337 `int CbcModel::takeOffCuts ( OsiCuts & cuts, bool allowResolve, OsiCuts * saveCuts, int numberNewCuts = 0, const OsiRowCut ** newCuts = NULL )`

Remove inactive cuts from the model.

An OsiSolverInterface is expected to maintain a valid basis, but not a valid solution, when loose cuts are deleted. Restoring a valid solution requires calling the solver to reoptimise. If it's certain the solution will not be required, set allowResolve to false to suppress reoptimisation. If saveCuts then slack cuts will be saved On input current cuts are cuts and newCuts on exit current cuts will be correct. Returns number dropped

7.74.4.338 `int CbcModel::addCuts ( CbcNode * node, CoinWarmStartBasis *& lastws, bool canFix )`

Determine and install the active cuts that need to be added for the current subproblem.

The whole truth is a bit more complicated. The first action is a call to [addCuts1\(\)](#). [addCuts\(\)](#) then sorts through the list, installs the tight cuts in the model, and does bookkeeping (adjusts reference counts). The basis returned from [addCuts1\(\)](#) is adjusted accordingly.

If it turns out that the node should really be fathomed by bound, [addCuts\(\)](#) simply treats all the cuts as loose as it does the bookkeeping.

canFix true if extra information being passed

7.74.4.339 `bool CbcModel::addCuts1 ( CbcNode * node, CoinWarmStartBasis *& lastws )`

Traverse the tree from node to root and prep the model.

[addCuts1\(\)](#) begins the job of prepping the model to match the current subproblem. The model is stripped of all cuts, and the search tree is traversed from node to root to determine the changes required. Appropriate bounds changes are installed, a list of cuts is collected but not installed, and an appropriate basis (minus the cuts, but big enough to accommodate them) is constructed.

Returns true if new problem similar to old

**Todo** [addCuts1\(\)](#) is called in contexts where it's known in advance that all that's desired is to determine a list of cuts and do the bookkeeping (adjust the reference counts). The work of installing bounds and building a basis goes to waste.

7.74.4.340 void CbcModel::previousBounds ( CbcNode \* *node*, CbcNodeInfo \* *where*, int *iColumn*, double & *lower*, double & *upper*, int *force* )

Returns bounds just before where - initially original bounds.

Also sets downstream nodes (lower if force 1, upper if 2)

7.74.4.341 void CbcModel::setObjectiveValue ( CbcNode \* *thisNode*, const CbcNode \* *parentNode* ) const

Set objective value in a node.

This is separated out so that odd solvers can use. It may look at extra information in solverCharacteriscs\_ and will also use bound from parent node

7.74.4.342 void CbcModel::convertToDynamic ( )

If numberBeforeTrust >0 then we are going to use CbcBranchDynamic.

Scan and convert CbcSimpleInteger objects

7.74.4.343 void CbcModel::synchronizeNumberBeforeTrust ( int *type* = 0 )

Set numberBeforeTrust in all objects.

7.74.4.344 void CbcModel::zapIntegerInformation ( bool *leaveObjects* = true )

Zap integer information in problem (may leave object info)

7.74.4.345 int CbcModel::cliquePseudoCosts ( int *doStatistics* )

Use cliques for pseudocost information - return nonzero if infeasible.

7.74.4.346 void CbcModel::pseudoShadow ( int *type* )

Fill in useful estimates.

7.74.4.347 void CbcModel::fillPseudoCosts ( double \* *downCosts*, double \* *upCosts*, int \* *priority* = NULL, int \* *numberDown* = NULL, int \* *numberUp* = NULL, int \* *numberDownInfeasible* = NULL, int \* *numberUpInfeasible* = NULL ) const

Return pseudo costs If not all integers or not pseudo costs - returns all zero Length of arrays are numberIntegers() and entries correspond to integerVariable()[i] User must allocate arrays before call.

7.74.4.348 void CbcModel::doHeuristicsAtRoot ( int *deleteHeuristicsAfterwards* = 0 )

Do heuristics at root.

0 - don't delete 1 - delete 2 - just delete - don't even use

7.74.4.349 void CbcModel::adjustHeuristics ( )

Adjust heuristics based on model.

7.74.4.350 const double\* CbcModel::hotstartSolution ( ) const [inline]

Get the hotstart solution.

Definition at line 2254 of file CbcModel.hpp.

**7.74.4.351** `const int* CbcModel::hotstartPriorities ( ) const` `[inline]`

Get the hotstart priorities.

Definition at line 2258 of file CbcModel.hpp.

**7.74.4.352** `CbcCountRowCut** CbcModel::addedCuts ( ) const` `[inline]`

Return the list of cuts initially collected for this subproblem.

Definition at line 2263 of file CbcModel.hpp.

**7.74.4.353** `int CbcModel::currentNumberCuts ( ) const` `[inline]`

Number of entries in the list returned by [addedCuts\(\)](#)

Definition at line 2267 of file CbcModel.hpp.

**7.74.4.354** `CbcRowCuts* CbcModel::globalCuts ( )` `[inline]`

Global cuts.

Definition at line 2271 of file CbcModel.hpp.

**7.74.4.355** `void CbcModel::setNextRowCut ( const OsiRowCut & cut )`

Copy and set a pointer to a row cut which will be added instead of normal branching.

**7.74.4.356** `CbcNode* CbcModel::currentNode ( ) const` `[inline]`

Get a pointer to current node (be careful)

Definition at line 2277 of file CbcModel.hpp.

**7.74.4.357** `CglTreeProbingInfo* CbcModel::probingInfo ( ) const` `[inline]`

Get a pointer to probing info.

Definition at line 2281 of file CbcModel.hpp.

**7.74.4.358** `CoinThreadRandom* CbcModel::randomNumberGenerator ( )` `[inline]`

Thread specific random number generator.

Definition at line 2285 of file CbcModel.hpp.

**7.74.4.359** `void CbcModel::setNumberStrongIterations ( int number )` `[inline]`

Set the number of iterations done in strong branching.

Definition at line 2289 of file CbcModel.hpp.

**7.74.4.360** `int CbcModel::numberStrongIterations ( ) const` `[inline]`

Get the number of iterations done in strong branching.

Definition at line 2293 of file CbcModel.hpp.

**7.74.4.361** `int CbcModel::maximumNumberIterations ( ) const` `[inline]`

Get maximum number of iterations (designed to be used in heuristics)



Definition at line 2297 of file CbcModel.hpp.

**7.74.4.362** void CbcModel::setMaximumNumberIterations ( int *value* ) [inline]

Set maximum number of iterations (designed to be used in heuristics)

Definition at line 2301 of file CbcModel.hpp.

**7.74.4.363** void CbcModel::setFastNodeDepth ( int *value* ) [inline]

Set depth for fast nodes.

Definition at line 2305 of file CbcModel.hpp.

**7.74.4.364** int CbcModel::fastNodeDepth ( ) const [inline]

Get depth for fast nodes.

Definition at line 2309 of file CbcModel.hpp.

**7.74.4.365** int CbcModel::continuousPriority ( ) const [inline]

Get anything with priority  $\geq$  this can be treated as continuous.

Definition at line 2313 of file CbcModel.hpp.

**7.74.4.366** void CbcModel::setContinuousPriority ( int *value* ) [inline]

Set anything with priority  $\geq$  this can be treated as continuous.

Definition at line 2317 of file CbcModel.hpp.

**7.74.4.367** void CbcModel::incrementExtra ( int *nodes*, int *iterations* ) [inline]

Definition at line 2320 of file CbcModel.hpp.

**7.74.4.368** int CbcModel::numberExtraIterations ( ) const [inline]

Number of extra iterations.

Definition at line 2325 of file CbcModel.hpp.

**7.74.4.369** void CbcModel::incrementStrongInfo ( int *numberTimes*, int *numberIterations*, int *numberFixed*, bool *ifInfeasible* )

Increment strong info.

**7.74.4.370** const int\* CbcModel::strongInfo ( ) const [inline]

Return strong info.

Definition at line 2332 of file CbcModel.hpp.

**7.74.4.371** int\* CbcModel::mutableStrongInfo ( ) [inline]

Return mutable strong info.

Definition at line 2337 of file CbcModel.hpp.

**7.74.4.372** CglStored\* CbcModel::storedRowCuts ( ) const [inline]

Get stored row cuts for donor/recipient [CbcModel](#).

Definition at line 2341 of file CbcModel.hpp.

**7.74.4.373** `void CbcModel::setStoredRowCuts ( CglStored * cuts ) [inline]`

Set stored row cuts for donor/recipient [CbcModel](#).

Definition at line 2345 of file CbcModel.hpp.

**7.74.4.374** `bool CbcModel::allDynamic ( ) const [inline]`

Says whether all dynamic integers.

Definition at line 2349 of file CbcModel.hpp.

**7.74.4.375** `void CbcModel::generateCpp ( FILE * fp, int options )`

Create C++ lines to get to current state.

**7.74.4.376** `OsiBranchingInformation CbcModel::usefullInformation ( ) const`

Generate an OsiBranchingInformation object.

**7.74.4.377** `void CbcModel::setBestSolutionBasis ( const CoinWarmStartBasis & bestSolutionBasis ) [inline]`

Warm start object produced by heuristic or strong branching.

If get a valid integer solution outside branch and bound then it can take a reasonable time to solve LP which produces clean solution. If this object has any size then it will be used in solve.

Definition at line 2362 of file CbcModel.hpp.

**7.74.4.378** `void CbcModel::redoWalkBack ( )`

Redo walkback arrays.

**7.74.4.379** `void CbcModel::setMIPStart ( const std::vector< std::pair< std::string, double > > & mips ) [inline]`

Definition at line 2369 of file CbcModel.hpp.

**7.74.4.380** `const std::vector< std::pair< std::string, double > > & CbcModel::getMIPStart ( ) [inline]`

Definition at line 2373 of file CbcModel.hpp.

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

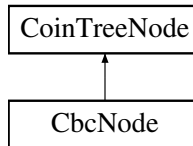
- [/home/ted/COIN/trunk/Cbc/src/CbcModel.hpp](#)

## 7.75 CbcNode Class Reference

Information required while the node is live.

```
#include <CbcNode.hpp>
```

Inheritance diagram for CbcNode:



### Public Member Functions

- `CbcNode ()`  
*Default Constructor.*
- `CbcNode (CbcModel *model, CbcNode *lastNode)`  
*Construct and increment parent reference count.*
- `CbcNode (const CbcNode &)`  
*Copy constructor.*
- `CbcNode & operator= (const CbcNode &rhs)`  
*Assignment operator.*
- `~CbcNode ()`  
*Destructor.*
- `void createInfo (CbcModel *model, CbcNode *lastNode, const CoinWarmStartBasis *lastws, const double *lastLower, const double *lastUpper, int numberOldActiveCuts, int numberNewCuts)`  
*Create a description of the subproblem at this node.*
- `int chooseBranch (CbcModel *model, CbcNode *lastNode, int numberPassesLeft)`  
*Create a branching object for the node.*
- `int chooseDynamicBranch (CbcModel *model, CbcNode *lastNode, OsiSolverBranch *&branches, int numberPassesLeft)`  
*Create a branching object for the node - when dynamic pseudo costs.*
- `int chooseOsiBranch (CbcModel *model, CbcNode *lastNode, OsiBranchingInformation *usefullInfo, int branchState)`  
*Create a branching object for the node.*
- `int chooseClipBranch (CbcModel *model, CbcNode *lastNode)`  
*Create a branching object for the node.*
- `int analyze (CbcModel *model, double *results)`
- `void decrementCuts (int change=1)`  
*Decrement active cut counts.*
- `void decrementParentCuts (CbcModel *model, int change=1)`  
*Decrement all active cut counts in chain starting at parent.*
- `void nullNodeInfo ()`  
*Nulls out node info.*
- `void initializeInfo ()`  
*Initialize reference counts in attached CbcNodeInfo.*
- `int branch (OsiSolverInterface *solver)`  
*Does next branch and updates state.*
- `double checksCutoff (double cutoff)`  
*Double checks in case node can change its mind! Returns objective value Can change objective etc.*
- `CbcNodeInfo * nodeInfo () const`
- `double objectiveValue () const`
- `void setObjectiveValue (double value)`
- `int numberBranches () const`

*Number of arms defined for the attached OsiBranchingObject.*

- int `way` () const
- int `depth` () const

*Depth in branch-and-cut search tree.*

- void `setDepth` (int value)

*Set depth in branch-and-cut search tree.*

- int `numberUnsatisfied` () const

*Get the number of objects unsatisfied at this node.*

- void `setNumberUnsatisfied` (int value)

*Set the number of objects unsatisfied at this node.*

- double `sumInfeasibilities` () const

*Get sum of "infeasibilities" reported by each object.*

- void `setSumInfeasibilities` (double value)

*Set sum of "infeasibilities" reported by each object.*

- double `guessedObjectiveValue` () const
- void `setGuessedObjectiveValue` (double value)
- const OsiBranchingObject \* `branchingObject` () const

*Branching object for this node.*

- OsiBranchingObject \* `modifiableBranchingObject` () const

*Modifiable branching object for this node.*

- void `setBranchingObject` (OsiBranchingObject \*`branchingObject`)

*Set branching object for this node (takes ownership)*

- int `nodeNumber` () const

*The node number.*

- void `setNodeNumber` (int node)

- bool `onTree` () const

*Returns true if on tree.*

- void `setOnTree` (bool yesNo)

*Sets true if on tree.*

- bool `active` () const

*Returns true if active.*

- void `setActive` (bool yesNo)

*Sets true if active.*

- int `getState` () const

*Get state (really for debug)*

- void `setState` (int value)

*Set state (really for debug)*

- void `print` () const

*Print.*

- void `checkInfo` () const

*Debug.*

## 7.75.1 Detailed Description

Information required while the node is live.

When a subproblem is initially created, it is represented by an [CbcNode](#) object and an attached [CbcNodeInfo](#) object.

The [CbcNode](#) contains information (depth, branching instructions), that's needed while the subproblem remains 'live', *i.e.*, while the subproblem is not fathomed and there are branch arms still to be evaluated. The [CbcNode](#) is deleted when the last branch arm has been evaluated.

The [CbcNodeInfo](#) object contains the information needed to maintain the search tree and recreate the subproblem for the node. It remains in existence until there are no nodes remaining in the subtree rooted at this node.

Definition at line 49 of file [CbcNode.hpp](#).

## 7.75.2 Constructor &amp; Destructor Documentation

7.75.2.1 [CbcNode::CbcNode \( \)](#)

Default Constructor.

7.75.2.2 [CbcNode::CbcNode \( CbcModel \\* model, CbcNode \\* lastNode \)](#)

Construct and increment parent reference count.

7.75.2.3 [CbcNode::CbcNode \( const CbcNode & \)](#)

Copy constructor.

7.75.2.4 [CbcNode::~~CbcNode \( \)](#)

Destructor.

## 7.75.3 Member Function Documentation

7.75.3.1 [CbcNode& CbcNode::operator= \( const CbcNode & rhs \)](#)

Assignment operator.

7.75.3.2 [void CbcNode::createInfo \( CbcModel \\* model, CbcNode \\* lastNode, const CoinWarmStartBasis \\* lastws, const double \\* lastLower, const double \\* lastUpper, int numberOldActiveCuts, int numberNewCuts \)](#)

Create a description of the subproblem at this node.

The [CbcNodeInfo](#) structure holds the information (basis & variable bounds) required to recreate the subproblem for this node. It also links the node to its parent (via the parent's [CbcNodeInfo](#) object).

If `lastNode == NULL`, a [CbcFullNodeInfo](#) object will be created. All parameters except `model` are unused.

If `lastNode != NULL`, a [CbcPartialNodeInfo](#) object will be created. Basis and bounds information will be stored in the form of differences between the parent subproblem and this subproblem. (More precisely, `lastws`, `lastUpper`, `lastLower`, `numberOldActiveCuts`, and `numberNewCuts` are used.)

7.75.3.3 [int CbcNode::chooseBranch \( CbcModel \\* model, CbcNode \\* lastNode, int numberPassesLeft \)](#)

Create a branching object for the node.

The routine scans the object list of the model and selects a set of unsatisfied objects as candidates for branching. The

candidates are evaluated, and an appropriate branch object is installed.

The numberPassesLeft is decremented to stop fixing one variable each time and going on and on (e.g. for stock cutting, air crew scheduling)

If evaluation determines that an object is monotone or infeasible, the routine returns immediately. In the case of a monotone object, the branch object has already been called to modify the model.

Return value:

- 0: A branching object has been installed
- -1: A monotone object was discovered
- -2: An infeasible object was discovered

#### 7.75.3.4 `int CbcNode::chooseDynamicBranch ( CbcModel * model, CbcNode * lastNode, OsiSolverBranch *& branches, int numberPassesLeft )`

Create a branching object for the node - when dynamic pseudo costs.

The routine scans the object list of the model and selects a set of unsatisfied objects as candidates for branching. The candidates are evaluated, and an appropriate branch object is installed. This version gives preference in evaluation to variables which have not been evaluated many times. It also uses numberStrong to say give up if last few tries have not changed incumbent. See Achterberg, Koch and Martin.

The numberPassesLeft is decremented to stop fixing one variable each time and going on and on (e.g. for stock cutting, air crew scheduling)

If evaluation determines that an object is monotone or infeasible, the routine returns immediately. In the case of a monotone object, the branch object has already been called to modify the model.

Return value:

- 0: A branching object has been installed
- -1: A monotone object was discovered
- -2: An infeasible object was discovered
- >0: Number of quick branching objects (and branches will be non NULL)

#### 7.75.3.5 `int CbcNode::chooseOsiBranch ( CbcModel * model, CbcNode * lastNode, OsiBranchingInformation * usefulInfo, int branchState )`

Create a branching object for the node.

The routine scans the object list of the model and selects a set of unsatisfied objects as candidates for branching. The candidates are evaluated, and an appropriate branch object is installed.

The numberPassesLeft is decremented to stop fixing one variable each time and going on and on (e.g. for stock cutting, air crew scheduling)

If evaluation determines that an object is monotone or infeasible, the routine returns immediately. In the case of a monotone object, the branch object has already been called to modify the model.

Return value:

- 0: A branching object has been installed
- -1: A monotone object was discovered

- -2: An infeasible object was discovered

Branch state:

- -1: start
- -1: A monotone object was discovered
- -2: An infeasible object was discovered

**7.75.3.6** `int CbcNode::chooseCipBranch ( CbcModel * model, CbcNode * lastNode )`

Create a branching object for the node.

The routine scans the object list of the model and selects a set of unsatisfied objects as candidates for branching. It then solves a series of problems and a [CbcGeneral](#) branch object is installed.

If evaluation determines that an object is infeasible, the routine returns immediately.

Return value:

- 0: A branching object has been installed
- -2: An infeasible object was discovered

**7.75.3.7** `int CbcNode::analyze ( CbcModel * model, double * results )`

**7.75.3.8** `void CbcNode::decrementCuts ( int change = 1 )`

Decrement active cut counts.

**7.75.3.9** `void CbcNode::decrementParentCuts ( CbcModel * model, int change = 1 )`

Decrement all active cut counts in chain starting at parent.

**7.75.3.10** `void CbcNode::nullNodeInfo ( )`

Nulls out node info.

**7.75.3.11** `void CbcNode::initializeInfo ( )`

Initialize reference counts in attached [CbcNodeInfo](#).

This is a convenience routine, which will initialize the reference counts in the attached [CbcNodeInfo](#) object based on the attached [OsiBranchingObject](#).

See Also

[CbcNodeInfo::initializeInfo\(int\)](#).

**7.75.3.12** `int CbcNode::branch ( OsiSolverInterface * solver )`

Does next branch and updates state.

**7.75.3.13** `double CbcNode::checkIsCutoff ( double cutoff )`

Double checks in case node can change its mind! Returns objective value Can change objective etc.

**7.75.3.14** `CbcNodeInfo* CbcNode::nodeInfo ( ) const [inline]`

Definition at line 216 of file CbcNode.hpp.

**7.75.3.15** `double CbcNode::objectiveValue ( ) const [inline]`

Definition at line 221 of file CbcNode.hpp.

**7.75.3.16** `void CbcNode::setObjectiveValue ( double value ) [inline]`

Definition at line 224 of file CbcNode.hpp.

**7.75.3.17** `int CbcNode::numberBranches ( ) const [inline]`

Number of arms defined for the attached OsiBranchingObject.

Definition at line 228 of file CbcNode.hpp.

**7.75.3.18** `int CbcNode::way ( ) const`

**7.75.3.19** `int CbcNode::depth ( ) const [inline]`

Depth in branch-and-cut search tree.

Definition at line 243 of file CbcNode.hpp.

**7.75.3.20** `void CbcNode::setDepth ( int value ) [inline]`

Set depth in branch-and-cut search tree.

Definition at line 247 of file CbcNode.hpp.

**7.75.3.21** `int CbcNode::numberUnsatisfied ( ) const [inline]`

Get the number of objects unsatisfied at this node.

Definition at line 251 of file CbcNode.hpp.

**7.75.3.22** `void CbcNode::setNumberUnsatisfied ( int value ) [inline]`

Set the number of objects unsatisfied at this node.

Definition at line 255 of file CbcNode.hpp.

**7.75.3.23** `double CbcNode::sumInfeasibilities ( ) const [inline]`

Get sum of "infeasibilities" reported by each object.

Definition at line 259 of file CbcNode.hpp.

**7.75.3.24** `void CbcNode::setSumInfeasibilities ( double value ) [inline]`

Set sum of "infeasibilities" reported by each object.

Definition at line 263 of file CbcNode.hpp.

**7.75.3.25** `double CbcNode::guessedObjectiveValue ( ) const [inline]`

Definition at line 267 of file CbcNode.hpp.



**7.75.3.26** void CbcNode::setGuessedObjectiveValue ( double *value* ) [inline]

Definition at line 270 of file CbcNode.hpp.

**7.75.3.27** const OsiBranchingObject\* CbcNode::branchingObject ( ) const [inline]

Branching object for this node.

Definition at line 274 of file CbcNode.hpp.

**7.75.3.28** OsiBranchingObject\* CbcNode::modifiableBranchingObject ( ) const [inline]

Modifiable branching object for this node.

Definition at line 278 of file CbcNode.hpp.

**7.75.3.29** void CbcNode::setBranchingObject ( OsiBranchingObject \* *branchingObject* ) [inline]

Set branching object for this node (takes ownership)

Definition at line 282 of file CbcNode.hpp.

**7.75.3.30** int CbcNode::nodeNumber ( ) const [inline]

The node number.

Definition at line 286 of file CbcNode.hpp.

**7.75.3.31** void CbcNode::setNodeNumber ( int *node* ) [inline]

Definition at line 289 of file CbcNode.hpp.

**7.75.3.32** bool CbcNode::onTree ( ) const [inline]

Returns true if on tree.

Definition at line 293 of file CbcNode.hpp.

**7.75.3.33** void CbcNode::setOnTree ( bool *yesNo* ) [inline]

Sets true if on tree.

Definition at line 297 of file CbcNode.hpp.

**7.75.3.34** bool CbcNode::active ( ) const [inline]

Returns true if active.

Definition at line 302 of file CbcNode.hpp.

**7.75.3.35** void CbcNode::setActive ( bool *yesNo* ) [inline]

Sets true if active.

Definition at line 306 of file CbcNode.hpp.

**7.75.3.36** int CbcNode::getState ( ) const [inline]

Get state (really for debug)

Definition at line 311 of file CbcNode.hpp.

7.75.3.37 void CbcNode::setState ( int value ) [inline]

Set state (really for debug)

Definition at line 314 of file CbcNode.hpp.

7.75.3.38 void CbcNode::print ( ) const

Print.

7.75.3.39 void CbcNode::checkInfo ( ) const [inline]

Debug.

Definition at line 319 of file CbcNode.hpp.

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

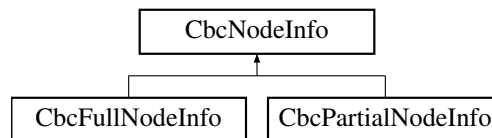
- /home/ted/COIN/trunk/Cbc/src/CbcNode.hpp

## 7.76 CbcNodeInfo Class Reference

Information required to recreate the subproblem at this node.

```
#include <CbcNodeInfo.hpp>
```

Inheritance diagram for CbcNodeInfo:



### Public Member Functions

- virtual void [applyToModel](#) (CbcModel \*model, CoinWarmStartBasis \*&basis, CbcCountRowCut \*\*addCuts, int &currentNumberCuts) const =0  
*Modify model according to information at node.*
- virtual int [applyBounds](#) (int iColumn, double &lower, double &upper, int force)=0  
*Just apply bounds to one variable - force means overwrite by lower,upper (1=>infeasible)*
- virtual CbcNodeInfo \* [buildRowBasis](#) (CoinWarmStartBasis &basis) const =0  
*Builds up row basis backwards (until original model).*
- virtual CbcNodeInfo \* [clone](#) () const =0  
*Clone.*
- virtual void [allBranchesGone](#) ()  
*Called when number branches left down to zero.*
- void [increment](#) (int amount=1)  
*Increment number of references.*
- int [decrement](#) (int amount=1)  
*Decrement number of references and return number left.*
- void [initializeInfo](#) (int number)  
*Initialize reference counts.*

- int `numberBranchesLeft` () const  
*Return number of branches left in object.*
- void `setNumberBranchesLeft` (int value)  
*Set number of branches left in object.*
- int `numberPointingToThis` () const  
*Return number of objects pointing to this.*
- void `setNumberPointingToThis` (int number)  
*Set number of objects pointing to this.*
- void `incrementNumberPointingToThis` ()  
*Increment number of objects pointing to this.*
- int `branchedOn` ()  
*Say one branch taken.*
- void `throwAway` ()  
*Say thrown away.*
- `CbcNodeInfo` \* `parent` () const  
*Parent of this.*
- void `nullParent` ()  
*Set parent null.*
- void `addCuts` (OsiCuts &`cuts`, int numberToBranch, int `numberPointingToThis`)
- void `addCuts` (int `numberCuts`, `CbcCountRowCut` \*\*`cuts`, int numberToBranch)
- void `deleteCuts` (int numberToDelete, `CbcCountRowCut` \*\*`cuts`)  
*Delete cuts (decrements counts) Slow unless cuts in same order as saved.*
- void `deleteCuts` (int numberToDelete, int \*which)
- void `deleteCut` (int whichOne)  
*Really delete a cut.*
- void `decrementCuts` (int change=1)  
*Decrement active cut counts.*
- void `incrementCuts` (int change=1)  
*Increment active cut counts.*
- void `decrementParentCuts` (`CbcModel` \*model, int change=1)  
*Decrement all active cut counts in chain starting at parent.*
- void `incrementParentCuts` (`CbcModel` \*model, int change=1)  
*Increment all active cut counts in parent chain.*
- `CbcCountRowCut` \*\* `cuts` () const  
*Array of pointers to cuts.*
- int `numberCuts` () const  
*Number of row cuts (this node)*
- void `setNumberCuts` (int value)
- void `nullOwner` ()  
*Set owner null.*
- const `CbcNode` \* `owner` () const
- `CbcNode` \* `mutableOwner` () const
- int `nodeNumber` () const  
*The node number.*
- void `setNodeNumber` (int node)
- void `deactivate` (int mode=3)  
*Deactivate node information.*

- bool `allActivated` () const  
*Say if normal.*
- bool `marked` () const  
*Say if marked.*
- void `mark` ()  
*Mark.*
- void `unmark` ()  
*Unmark.*
- const OsiBranchingObject \* `parentBranch` () const  
*Branching object for the parent.*
- void `unsetParentBasedData` ()  
*If we need to take off parent based data.*

### Constructors & destructors

- `CbcNodeInfo` ()  
*Default Constructor.*
- `CbcNodeInfo` (const `CbcNodeInfo` &)  
*Copy constructor.*
- `CbcNodeInfo` (`CbcNodeInfo` \*parent, `CbcNode` \*owner)  
*Construct with parent and owner.*
- virtual `~CbcNodeInfo` ()  
*Destructor.*

### Protected Attributes

- int `numberPointingToThis_`  
*Number of other nodes pointing to this node.*
- `CbcNodeInfo` \* `parent_`  
*parent*
- OsiBranchingObject \* `parentBranch_`  
*Copy of the branching object of the parent when the node is created.*
- `CbcNode` \* `owner_`  
*Owner.*
- int `numberCuts_`  
*Number of row cuts (this node)*
- int `nodeNumber_`  
*The node number.*
- `CbcCountRowCut` \*\* `cuts_`  
*Array of pointers to cuts.*
- int `numberRows_`  
*Number of rows in problem (before these cuts).*
- int `numberBranchesLeft_`  
*Number of branch arms left to explore at this node.*
- int `active_`  
*Active node information.*

### 7.76.1 Detailed Description

Information required to recreate the subproblem at this node.

When a subproblem is initially created, it is represented by a [CbcNode](#) object and an attached [CbcNodeInfo](#) object.

The [CbcNode](#) contains information needed while the subproblem remains live. The [CbcNode](#) is deleted when the last branch arm has been evaluated.

The [CbcNodeInfo](#) contains information required to maintain the branch-and-cut search tree structure (links and reference counts) and to recreate the subproblem for this node (basis, variable bounds, cutting planes). A [CbcNodeInfo](#) object remains in existence until all nodes have been pruned from the subtree rooted at this node.

The principle used to maintain the reference count is that the reference count is always the sum of all potential and actual children of the node. Specifically,

- Once it's determined how the node will branch, the reference count is set to the number of potential children (*i.e.*, the number of arms of the branch).
- As each child is created by [CbcNode::branch\(\)](#) (converting a potential child to the active subproblem), the reference count is decremented.
- If the child survives and will become a node in the search tree (converting the active subproblem into an actual child), increment the reference count.

Notice that the active subproblem lives in a sort of limbo, neither a potential or an actual node in the branch-and-cut tree.

[CbcNodeInfo](#) objects come in two flavours. A [CbcFullNodeInfo](#) object contains a full record of the information required to recreate a subproblem. A [CbcPartialNodeInfo](#) object expresses this information in terms of differences from the parent.

Definition at line 68 of file [CbcNodeInfo.hpp](#).

### 7.76.2 Constructor & Destructor Documentation

#### 7.76.2.1 [CbcNodeInfo::CbcNodeInfo \( \)](#)

Default Constructor.

Creates an empty NodeInfo object.

#### 7.76.2.2 [CbcNodeInfo::CbcNodeInfo \( const \[CbcNodeInfo\]\(#\) & \)](#)

Copy constructor.

#### 7.76.2.3 [CbcNodeInfo::CbcNodeInfo \( \[CbcNodeInfo\]\(#\) \\* \*parent\*, \[CbcNode\]\(#\) \\* \*owner\* \)](#)

Construct with parent and owner.

As for 'construct with parent', and attached to `owner`.

#### 7.76.2.4 [virtual CbcNodeInfo::~~CbcNodeInfo \( \)](#) `[virtual]`

Destructor.

Note that the destructor will recursively delete the parent if this nodeInfo is the last child.

### 7.76.3 Member Function Documentation

**7.76.3.1** `virtual void CbcNodeInfo::applyToModel ( CbcModel * model, CoinWarmStartBasis *& basis, CbcCountRowCut ** addCuts, int & currentNumberCuts ) const` `[pure virtual]`

Modify model according to information at node.

The routine modifies the model according to bound and basis information at node and adds any cuts to the `addCuts` array.

Implemented in [CbcFullNodeInfo](#), and [CbcPartialNodeInfo](#).

**7.76.3.2** `virtual int CbcNodeInfo::applyBounds ( int iColumn, double & lower, double & upper, int force )` `[pure virtual]`

Just apply bounds to one variable - force means overwrite by lower,upper (1=>infeasible)

Implemented in [CbcFullNodeInfo](#), and [CbcPartialNodeInfo](#).

**7.76.3.3** `virtual CbcNodeInfo* CbcNodeInfo::buildRowBasis ( CoinWarmStartBasis & basis ) const` `[pure virtual]`

Builds up row basis backwards (until original model).

Returns NULL or previous one to apply . Depends on Free being 0 and impossible for cuts

Implemented in [CbcFullNodeInfo](#), and [CbcPartialNodeInfo](#).

**7.76.3.4** `virtual CbcNodeInfo* CbcNodeInfo::clone ( ) const` `[pure virtual]`

Clone.

Implemented in [CbcFullNodeInfo](#), and [CbcPartialNodeInfo](#).

**7.76.3.5** `virtual void CbcNodeInfo::allBranchesGone ( )` `[inline],[virtual]`

Called when number branches left down to zero.

Definition at line 126 of file `CbcNodeInfo.hpp`.

**7.76.3.6** `void CbcNodeInfo::increment ( int amount = 1 )` `[inline]`

Increment number of references.

Definition at line 129 of file `CbcNodeInfo.hpp`.

**7.76.3.7** `int CbcNodeInfo::decrement ( int amount = 1 )` `[inline]`

Decrement number of references and return number left.

Definition at line 134 of file `CbcNodeInfo.hpp`.

**7.76.3.8** `void CbcNodeInfo::initializeInfo ( int number )` `[inline]`

Initialize reference counts.

Initialize the reference counts used for tree maintenance.

Definition at line 149 of file `CbcNodeInfo.hpp`.

**7.76.3.9** `int CbcNodeInfo::numberBranchesLeft ( ) const` `[inline]`

Return number of branches left in object.

Definition at line 155 of file `CbcNodeInfo.hpp`.

7.76.3.10 void CbcNodeInfo::setNumberBranchesLeft ( int *value* ) [inline]

Set number of branches left in object.

Definition at line 160 of file CbcNodeInfo.hpp.

7.76.3.11 int CbcNodeInfo::numberPointingToThis ( ) const [inline]

Return number of objects pointing to this.

Definition at line 165 of file CbcNodeInfo.hpp.

7.76.3.12 void CbcNodeInfo::setNumberPointingToThis ( int *number* ) [inline]

Set number of objects pointing to this.

Definition at line 170 of file CbcNodeInfo.hpp.

7.76.3.13 void CbcNodeInfo::incrementNumberPointingToThis ( ) [inline]

Increment number of objects pointing to this.

Definition at line 175 of file CbcNodeInfo.hpp.

7.76.3.14 int CbcNodeInfo::branchedOn ( ) [inline]

Say one branch taken.

Definition at line 180 of file CbcNodeInfo.hpp.

7.76.3.15 void CbcNodeInfo::throwAway ( ) [inline]

Say thrown away.

Definition at line 187 of file CbcNodeInfo.hpp.

7.76.3.16 CbcNodeInfo\* CbcNodeInfo::parent ( ) const [inline]

Parent of this.

Definition at line 193 of file CbcNodeInfo.hpp.

7.76.3.17 void CbcNodeInfo::nullParent ( ) [inline]

Set parent null.

Definition at line 197 of file CbcNodeInfo.hpp.

7.76.3.18 void CbcNodeInfo::addCuts ( OsiCuts & *cuts*, int *numberToBranch*, int *numberPointingToThis* )

7.76.3.19 void CbcNodeInfo::addCuts ( int *numberCuts*, CbcCountRowCut \*\* *cuts*, int *numberToBranch* )

7.76.3.20 void CbcNodeInfo::deleteCuts ( int *numberToDelete*, CbcCountRowCut \*\* *cuts* )

Delete cuts (decrements counts) Slow unless cuts in same order as saved.

7.76.3.21 void CbcNodeInfo::deleteCuts ( int *numberToDelete*, int \* *which* )

7.76.3.22 void CbcNodeInfo::deleteCut ( int *whichOne* )

Really delete a cut.

**7.76.3.23** void CbcNodeInfo::decrementCuts ( int *change* = 1 )

Decrement active cut counts.

**7.76.3.24** void CbcNodeInfo::incrementCuts ( int *change* = 1 )

Increment active cut counts.

**7.76.3.25** void CbcNodeInfo::decrementParentCuts ( CbcModel \* *model*, int *change* = 1 )

Decrement all active cut counts in chain starting at parent.

**7.76.3.26** void CbcNodeInfo::incrementParentCuts ( CbcModel \* *model*, int *change* = 1 )

Increment all active cut counts in parent chain.

**7.76.3.27** CbcCountRowCut\*\* CbcNodeInfo::cuts ( ) const [inline]

Array of pointers to cuts.

Definition at line 226 of file CbcNodeInfo.hpp.

**7.76.3.28** int CbcNodeInfo::numberCuts ( ) const [inline]

Number of row cuts (this node)

Definition at line 231 of file CbcNodeInfo.hpp.

**7.76.3.29** void CbcNodeInfo::setNumberCuts ( int *value* ) [inline]

Definition at line 234 of file CbcNodeInfo.hpp.

**7.76.3.30** void CbcNodeInfo::nullOwner ( ) [inline]

Set owner null.

Definition at line 239 of file CbcNodeInfo.hpp.

**7.76.3.31** const CbcNode\* CbcNodeInfo::owner ( ) const [inline]

Definition at line 242 of file CbcNodeInfo.hpp.

**7.76.3.32** CbcNode\* CbcNodeInfo::mutableOwner ( ) const [inline]

Definition at line 245 of file CbcNodeInfo.hpp.

**7.76.3.33** int CbcNodeInfo::nodeNumber ( ) const [inline]

The node number.

Definition at line 249 of file CbcNodeInfo.hpp.

**7.76.3.34** void CbcNodeInfo::setNodeNumber ( int *node* ) [inline]

Definition at line 252 of file CbcNodeInfo.hpp.

**7.76.3.35** void CbcNodeInfo::deactivate ( int *mode* = 3 )

Deactivate node information.



1 - bounds 2 - cuts 4 - basis!

**7.76.3.36** `bool CbcNodeInfo::allActivated ( ) const [inline]`

Say if normal.

Definition at line 262 of file CbcNodeInfo.hpp.

**7.76.3.37** `bool CbcNodeInfo::marked ( ) const [inline]`

Say if marked.

Definition at line 266 of file CbcNodeInfo.hpp.

**7.76.3.38** `void CbcNodeInfo::mark ( ) [inline]`

Mark.

Definition at line 270 of file CbcNodeInfo.hpp.

**7.76.3.39** `void CbcNodeInfo::unmark ( ) [inline]`

Unmark.

Definition at line 274 of file CbcNodeInfo.hpp.

**7.76.3.40** `const OsiBranchingObject* CbcNodeInfo::parentBranch ( ) const [inline]`

Branching object for the parent.

Definition at line 279 of file CbcNodeInfo.hpp.

**7.76.3.41** `void CbcNodeInfo::unsetParentBasedData ( )`

If we need to take off parent based data.

#### 7.76.4 Member Data Documentation

**7.76.4.1** `int CbcNodeInfo::numberPointingToThis_ [protected]`

Number of other nodes pointing to this node.

Number of existing and potential search tree nodes pointing to this node. 'Existing' means referenced by [parent\\_](#) of some other [CbcNodeInfo](#). 'Potential' means children still to be created ([numberBranchesLeft\\_](#) of this [CbcNodeInfo](#)).

Definition at line 293 of file CbcNodeInfo.hpp.

**7.76.4.2** `CbcNodeInfo* CbcNodeInfo::parent_ [protected]`

parent

Definition at line 296 of file CbcNodeInfo.hpp.

**7.76.4.3** `OsiBranchingObject* CbcNodeInfo::parentBranch_ [protected]`

Copy of the branching object of the parent when the node is created.

Definition at line 299 of file CbcNodeInfo.hpp.

**7.76.4.4 CbcNode\* CbcNodeInfo::owner\_ [protected]**

Owner.

Definition at line 302 of file CbcNodeInfo.hpp.

**7.76.4.5 int CbcNodeInfo::numberCuts\_ [protected]**

Number of row cuts (this node)

Definition at line 305 of file CbcNodeInfo.hpp.

**7.76.4.6 int CbcNodeInfo::nodeNumber\_ [protected]**

The node number.

Definition at line 308 of file CbcNodeInfo.hpp.

**7.76.4.7 CbcCountRowCut\*\* CbcNodeInfo::cuts\_ [protected]**

Array of pointers to cuts.

Definition at line 311 of file CbcNodeInfo.hpp.

**7.76.4.8 int CbcNodeInfo::numberOfRows\_ [protected]**

Number of rows in problem (before these cuts).

This means that for top of chain it must be rows at continuous

Definition at line 315 of file CbcNodeInfo.hpp.

**7.76.4.9 int CbcNodeInfo::numberOfBranchesLeft\_ [protected]**

Number of branch arms left to explore at this node.

**Todo** There seems to be redundancy between this field and CbcBranchingObject::numberOfBranchesLeft\_. It'd be good to sort out if both are necessary.

Definition at line 323 of file CbcNodeInfo.hpp.

**7.76.4.10 int CbcNodeInfo::active\_ [protected]**

Active node information.

1 - bounds 2 - cuts 4 - basis!

Definition at line 329 of file CbcNodeInfo.hpp.

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

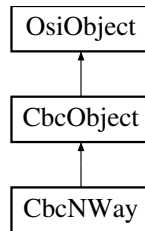
- /home/ted/COIN/trunk/Cbc/src/CbcNodeInfo.hpp

**7.77 CbcNWay Class Reference**

Define an n-way class for variables.

```
#include <CbcNWay.hpp>
```

Inheritance diagram for CbcNWay:



### Public Member Functions

- [CbcNWay](#) ()
- [CbcNWay](#) ([CbcModel](#) \**model*, int *numberMembers*, const int \**which*, int *identifier*)  
*Useful constructor (which are matrix indices)*
- [CbcNWay](#) (const [CbcNWay](#) &)
- virtual [CbcObject](#) \* [clone](#) () const  
*Clone.*
- [CbcNWay](#) & [operator=](#) (const [CbcNWay](#) &*rhs*)  
*Assignment operator.*
- virtual [~CbcNWay](#) ()  
*Destructor.*
- void [setConsequence](#) (int *iColumn*, const [CbcConsequence](#) &*consequence*)  
*Set up a consequence for a single member.*
- void [applyConsequence](#) (int *iSequence*, int *state*) const  
*Applies a consequence for a single member.*
- virtual double [infeasibility](#) (const [OsiBranchingInformation](#) \**info*, int &[preferredWay](#)) const  
*Infeasibility - large is 0.5 (and 0.5 will give this)*
- virtual void [feasibleRegion](#) ()  
*This looks at solution and sets bounds to contain solution.*
- virtual [CbcBranchingObject](#) \* [createCbcBranch](#) ([OsiSolverInterface](#) \**solver*, const [OsiBranchingInformation](#) \**info*, int *way*)  
*Creates a branching object.*
- int [numberMembers](#) () const  
*Number of members.*
- const int \* [members](#) () const  
*Members (indices in range 0 ... numberColumns-1)*
- virtual void [redoSequenceEtc](#) ([CbcModel](#) \**model*, int *numberColumns*, const int \**originalColumns*)  
*Redoes data when sequence numbers change.*

### Protected Attributes

- int [numberMembers\\_](#)  
*data Number of members*
- int \* [members\\_](#)  
*Members (indices in range 0 ... numberColumns-1)*
- [CbcConsequence](#) \*\* [consequence\\_](#)  
*Consequences (normally NULL)*

### 7.77.1 Detailed Description

Define an n-way class for variables.

Only valid value is one at UB others at LB Normally 0-1

Definition at line 15 of file CbcNWay.hpp.

### 7.77.2 Constructor & Destructor Documentation

#### 7.77.2.1 CbcNWay::CbcNWay ( )

#### 7.77.2.2 CbcNWay::CbcNWay ( CbcModel \* *model*, int *numberMembers*, const int \* *which*, int *identifier* )

Useful constructor (which are matrix indices)

#### 7.77.2.3 CbcNWay::CbcNWay ( const CbcNWay & )

#### 7.77.2.4 virtual CbcNWay::~~CbcNWay ( ) [virtual]

Destructor.

### 7.77.3 Member Function Documentation

#### 7.77.3.1 virtual CbcObject\* CbcNWay::clone ( ) const [virtual]

Clone.

Implements [CbcObject](#).

#### 7.77.3.2 CbcNWay& CbcNWay::operator= ( const CbcNWay & *rhs* )

Assignment operator.

#### 7.77.3.3 void CbcNWay::setConsequence ( int *iColumn*, const CbcConsequence & *consequence* )

Set up a consequence for a single member.

#### 7.77.3.4 void CbcNWay::applyConsequence ( int *iSequence*, int *state* ) const

Applies a consequence for a single member.

#### 7.77.3.5 virtual double CbcNWay::infeasibility ( const OsiBranchingInformation \* *info*, int & *preferredWay* ) const [virtual]

Infeasibility - large is 0.5 (and 0.5 will give this)

Reimplemented from [CbcObject](#).

#### 7.77.3.6 virtual void CbcNWay::feasibleRegion ( ) [virtual]

This looks at solution and sets bounds to contain solution.

Implements [CbcObject](#).

#### 7.77.3.7 virtual CbcBranchingObject\* CbcNWay::createCbcBranch ( OsiSolverInterface \* *solver*, const OsiBranchingInformation \* *info*, int *way* ) [virtual]

Creates a branching object.

Reimplemented from [CbcObject](#).

**7.77.3.8** `int CbcNWay::numberMembers ( ) const [inline]`

Number of members.

Definition at line 57 of file CbcNWay.hpp.

**7.77.3.9** `const int* CbcNWay::members ( ) const [inline]`

Members (indices in range 0 ... numberColumns-1)

Definition at line 62 of file CbcNWay.hpp.

**7.77.3.10** `virtual void CbcNWay::redoSequenceEtc ( CbcModel * model, int numberColumns, const int * originalColumns ) [virtual]`

Redoes data when sequence numbers change.

Reimplemented from [CbcObject](#).

#### 7.77.4 Member Data Documentation

**7.77.4.1** `int CbcNWay::numberMembers_ [protected]`

data Number of members

Definition at line 71 of file CbcNWay.hpp.

**7.77.4.2** `int* CbcNWay::members_ [protected]`

Members (indices in range 0 ... numberColumns-1)

Definition at line 74 of file CbcNWay.hpp.

**7.77.4.3** `CbcConsequence** CbcNWay::consequence_ [protected]`

Consequences (normally NULL)

Definition at line 76 of file CbcNWay.hpp.

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

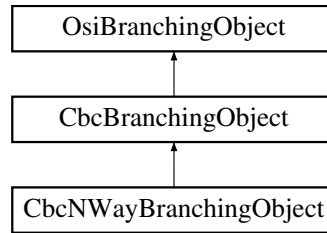
- [/home/ted/COIN/trunk/Cbc/src/CbcNWay.hpp](#)

## 7.78 CbcNWayBranchingObject Class Reference

N way branching Object class.

```
#include <CbcNWay.hpp>
```

Inheritance diagram for CbcNWayBranchingObject:



### Public Member Functions

- [CbcNWayBranchingObject](#) ()
- [CbcNWayBranchingObject](#) ([CbcModel](#) \**model*, const [CbcNWay](#) \**nway*, int *numberBranches*, const int \**order*)  
*Useful constructor - order had matrix indices way\_ -1 corresponds to setting first, +1 to second, +3 etc.*
- [CbcNWayBranchingObject](#) (const [CbcNWayBranchingObject](#) &)
- [CbcNWayBranchingObject](#) & *operator=* (const [CbcNWayBranchingObject](#) &*rhs*)
- virtual [CbcBranchingObject](#) \* *clone* () const  
*Clone.*
- virtual [~CbcNWayBranchingObject](#) ()
- virtual double *branch* ()  
*Does next branch and updates state.*
- virtual void *print* ()  
*Print something about branch - only if log level high.*
- virtual int *numberBranches* () const  
*The number of branch arms created for this branching object.*
- virtual bool *twoWay* () const  
*Is this a two way object (-1 down, +1 up)*
- virtual [CbcBranchObjType](#) *type* () const  
*Return the type (an integer identifier) of this.*
- virtual int *compareOriginalObject* (const [CbcBranchingObject](#) \**brObj*) const  
*Compare the original object of this with the original object of brObj.*
- virtual [CbcRangeCompare](#) *compareBranchingObject* (const [CbcBranchingObject](#) \**brObj*, const bool *replacelf-Overlap=false*)  
*Compare the this with brObj.*

### Additional Inherited Members

#### 7.78.1 Detailed Description

N way branching Object class.

Variable is number of set.

Definition at line 81 of file [CbcNWay.hpp](#).

## 7.78.2 Constructor &amp; Destructor Documentation

## 7.78.2.1 CbcNWayBranchingObject::CbcNWayBranchingObject ( )

7.78.2.2 CbcNWayBranchingObject::CbcNWayBranchingObject ( CbcModel \* *model*, const CbcNWay \* *nway*, int *numberBranches*, const int \* *order* )

Useful constructor - order had matrix indices way\_ -1 corresponds to setting first, +1 to second, +3 etc.  
this is so -1 and +1 have similarity to normal

## 7.78.2.3 CbcNWayBranchingObject::CbcNWayBranchingObject ( const CbcNWayBranchingObject &amp; )

## 7.78.2.4 virtual CbcNWayBranchingObject::~~CbcNWayBranchingObject ( ) [virtual]

## 7.78.3 Member Function Documentation

7.78.3.1 CbcNWayBranchingObject& CbcNWayBranchingObject::operator= ( const CbcNWayBranchingObject & *rhs* )

## 7.78.3.2 virtual CbcBranchingObject\* CbcNWayBranchingObject::clone ( ) const [virtual]

Clone.

Implements [CbcBranchingObject](#).

## 7.78.3.3 virtual double CbcNWayBranchingObject::branch ( ) [virtual]

Does next branch and updates state.

Implements [CbcBranchingObject](#).

## 7.78.3.4 virtual void CbcNWayBranchingObject::print ( ) [virtual]

Print something about branch - only if log level high.

## 7.78.3.5 virtual int CbcNWayBranchingObject::numberBranches ( ) const [inline],[virtual]

The number of branch arms created for this branching object.

Definition at line 125 of file CbcNWay.hpp.

## 7.78.3.6 virtual bool CbcNWayBranchingObject::twoWay ( ) const [inline],[virtual]

Is this a two way object (-1 down, +1 up)

Definition at line 129 of file CbcNWay.hpp.

## 7.78.3.7 virtual CbcBranchObjType CbcNWayBranchingObject::type ( ) const [inline],[virtual]

Return the type (an integer identifier) of `this`.

Implements [CbcBranchingObject](#).

Definition at line 134 of file CbcNWay.hpp.

7.78.3.8 virtual int CbcNWayBranchingObject::compareOriginalObject ( const CbcBranchingObject \* *brObj* ) const [virtual]

Compare the original object of `this` with the original object of `brObj`.

Assumes that there is an ordering of the original objects. This method should be invoked only if `this` and `brObj` are of

the same type. Return negative/0/positive depending on whether `this` is smaller/same/larger than the argument.

Reimplemented from [CbcBranchingObject](#).

**7.78.3.9** `virtual CbcRangeCompare CbcNWayBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false ) [virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

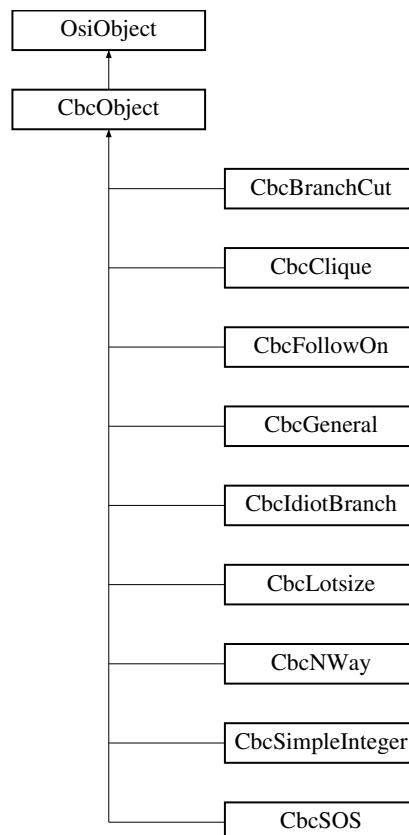
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcNWay.hpp](#)

## 7.79 CbcObject Class Reference

```
#include <CbcObject.hpp>
```

Inheritance diagram for `CbcObject`:





## Public Member Functions

- [CbcObject](#) ()
- [CbcObject](#) ([CbcModel](#) \*model)
- [CbcObject](#) (const [CbcObject](#) &)
- [CbcObject](#) & [operator=](#) (const [CbcObject](#) &rhs)
- virtual [CbcObject](#) \* [clone](#) () const =0  
*Clone.*
- virtual [~CbcObject](#) ()  
*Destructor.*
- virtual double [infeasibility](#) (const [OsiBranchingInformation](#) \*, int &[preferredWay](#)) const  
*Infeasibility of the object.*
- virtual double [infeasibility](#) (int &) const
- virtual void [feasibleRegion](#) ()=0  
*For the variable(s) referenced by the object, look at the current solution and set bounds to match the solution.*
- virtual double [feasibleRegion](#) ([OsiSolverInterface](#) \*solver, const [OsiBranchingInformation](#) \*info) const  
*Dummy one for compatibility.*
- virtual double [feasibleRegion](#) ([OsiSolverInterface](#) \*solver) const  
*For the variable(s) referenced by the object, look at the current solution and set bounds to match the solution.*
- virtual [CbcBranchingObject](#) \* [createCbcBranch](#) ([OsiSolverInterface](#) \*, const [OsiBranchingInformation](#) \*, int)  
*Create a branching object and indicate which way to branch first.*
- virtual [OsiBranchingObject](#) \* [createBranch](#) ([OsiSolverInterface](#) \*, const [OsiBranchingInformation](#) \*, int) const
- virtual [OsiBranchingObject](#) \* [createOsiBranch](#) ([OsiSolverInterface](#) \*solver, const [OsiBranchingInformation](#) \*info, int way) const  
*Create an Osibranching object and indicate which way to branch first.*
- virtual [OsiSolverBranch](#) \* [solverBranch](#) () const  
*Create an OsiSolverBranch object.*
- virtual [CbcBranchingObject](#) \* [preferredNewFeasible](#) () const  
*Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a good direction.*
- virtual [CbcBranchingObject](#) \* [notPreferredNewFeasible](#) () const  
*Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a bad direction.*
- virtual void [resetBounds](#) (const [OsiSolverInterface](#) \*)  
*Reset variable bounds to their original values.*
- virtual void [floorCeiling](#) (double &floorValue, double &ceilingValue, double value, double tolerance) const  
*Returns floor and ceiling i.e.*
- virtual [CbcObjectUpdateData](#) [createUpdateInformation](#) (const [OsiSolverInterface](#) \*solver, const [CbcNode](#) \*node, const [CbcBranchingObject](#) \*branchingObject)  
*Pass in information on branch just done and create CbcObjectUpdateData instance.*
- virtual void [updateInformation](#) (const [CbcObjectUpdateData](#) &)  
*Update object by CbcObjectUpdateData.*
- int [id](#) () const  
*Identifier (normally column number in matrix)*
- void [setId](#) (int value)  
*Set identifier (normally column number in matrix) but 1000000000 to 1100000000 means optional branching object i.e.*
- bool [optionalObject](#) () const  
*Return true if optional branching object i.e.*

- int [position](#) () const  
*Get position in object\_ list.*
- void [setPosition](#) (int [position](#))  
*Set position in object\_ list.*
- void [setModel](#) ([CbcModel](#) \*[model](#))  
*update model*
- [CbcModel](#) \* [model](#) () const  
*Return model.*
- int [preferredWay](#) () const  
*If -1 down always chosen first, +1 up always, 0 normal.*
- void [setPreferredWay](#) (int value)  
*Set -1 down always chosen first, +1 up always, 0 normal.*
- virtual void [redoSequenceEtc](#) ([CbcModel](#) \*, int, const int \*)  
*Redoes data when sequence numbers change.*
- virtual void [initializeForBranching](#) ([CbcModel](#) \*)  
*Initialize for branching.*

#### Protected Attributes

- [CbcModel](#) \* [model\\_](#)  
*data*
- int [id\\_](#)  
*Identifier (normally column number in matrix)*
- int [position\\_](#)  
*Position in object list.*
- int [preferredWay\\_](#)  
*If -1 down always chosen first, +1 up always, 0 normal.*

#### 7.79.1 Detailed Description

Definition at line 67 of file [CbcObject.hpp](#).

#### 7.79.2 Constructor & Destructor Documentation

##### 7.79.2.1 [CbcObject::CbcObject](#) ( )

##### 7.79.2.2 [CbcObject::CbcObject](#) ( [CbcModel](#) \* [model](#) )

##### 7.79.2.3 [CbcObject::CbcObject](#) ( const [CbcObject](#) & )

##### 7.79.2.4 virtual [CbcObject::~~CbcObject](#) ( ) [virtual]

Destructor.

## 7.79.3 Member Function Documentation

7.79.3.1 **CbcObject& CbcObject::operator= ( const CbcObject & rhs )**

7.79.3.2 **virtual CbcObject\* CbcObject::clone ( ) const** [pure virtual]

Clone.

Implemented in [CbcSimpleInteger](#), [CbcdiotBranch](#), [CbcClique](#), [CbcSimpleIntegerDynamicPseudoCost](#), [CbcSOS](#), [CbcBranchToFixLots](#), [CbcFollowOn](#), [CbcBranchAllDifferent](#), [CbcSimpleIntegerPseudoCost](#), [CbcGeneral](#), [CbcBranchCut](#), [CbcNWay](#), and [CbcLotsize](#).

7.79.3.3 **virtual double CbcObject::infeasibility ( const OsiBranchingInformation \*, int & preferredWay ) const** [inline], [virtual]

Infeasibility of the object.

This is some measure of the infeasibility of the object. It should be scaled to be in the range [0.0, 0.5], with 0.0 indicating the object is satisfied.

The preferred branching direction is returned in preferredWay,

This is used to prepare for strong branching but should also think of case when no strong branching

The object may also compute an estimate of cost of going "up" or "down". This will probably be based on pseudo-cost ideas

Reimplemented in [CbcSimpleInteger](#), [CbcdiotBranch](#), [CbcClique](#), [CbcSimpleIntegerDynamicPseudoCost](#), [CbcSOS](#), [CbcBranchToFixLots](#), [CbcFollowOn](#), [CbcBranchAllDifferent](#), [CbcNWay](#), [CbcSimpleIntegerPseudoCost](#), [CbcGeneral](#), [CbcBranchCut](#), and [CbcLotsize](#).

Definition at line 107 of file CbcObject.hpp.

7.79.3.4 **virtual double CbcObject::infeasibility ( int & ) const** [inline], [virtual]

Definition at line 111 of file CbcObject.hpp.

7.79.3.5 **virtual void CbcObject::feasibleRegion ( )** [pure virtual]

For the variable(s) referenced by the object, look at the current solution and set bounds to match the solution.

Implemented in [CbcSimpleInteger](#), [CbcdiotBranch](#), [CbcClique](#), [CbcSOS](#), [CbcFollowOn](#), [CbcBranchCut](#), [CbcNWay](#), [CbcLotsize](#), and [CbcGeneral](#).

7.79.3.6 **virtual double CbcObject::feasibleRegion ( OsiSolverInterface \* solver, const OsiBranchingInformation \* info ) const** [virtual]

Dummy one for compatibility.

Reimplemented in [CbcSimpleInteger](#).

7.79.3.7 **virtual double CbcObject::feasibleRegion ( OsiSolverInterface \* solver ) const** [virtual]

For the variable(s) referenced by the object, look at the current solution and set bounds to match the solution.

Returns measure of how much it had to move solution to make feasible

7.79.3.8 **virtual CbcBranchingObject\* CbcObject::createCbcBranch ( OsiSolverInterface \*, const OsiBranchingInformation \*, int )** [inline], [virtual]

Create a branching object and indicate which way to branch first.

The branching object has to know how to create branches (fix variables, etc.)

Reimplemented in [CbcSimpleInteger](#), [CbcIdiotBranch](#), [CbcClique](#), [CbcSOS](#), [CbcBranchToFixLots](#), [CbcSimpleIntegerDynamicPseudoCost](#), [CbcBranchCut](#), [CbcFollowOn](#), [CbcNWay](#), [CbcLotsize](#), [CbcBranchAllDifferent](#), [CbcGeneral](#), and [CbcSimpleIntegerPseudoCost](#).

Definition at line 137 of file CbcObject.hpp.

```
7.79.3.9  virtual OsiBranchingObject* CbcObject::createBranch ( OsiSolverInterface * , const OsiBranchingInformation * , int )
          const [inline],[virtual]
```

Definition at line 144 of file CbcObject.hpp.

```
7.79.3.10 virtual OsiBranchingObject* CbcObject::createOsiBranch ( OsiSolverInterface * solver, const OsiBranchingInformation *
          info, int way ) const [virtual]
```

Create an OsiBranching object and indicate which way to branch first.

The branching object has to know how to create branches (fix variables, etc.)

```
7.79.3.11 virtual OsiSolverBranch* CbcObject::solverBranch ( ) const [virtual]
```

Create an OsiSolverBranch object.

This returns NULL if branch not represented by bound changes

Reimplemented in [CbcSimpleIntegerDynamicPseudoCost](#), and [CbcSOS](#).

```
7.79.3.12 virtual CbcBranchingObject* CbcObject::preferredNewFeasible ( ) const [inline],[virtual]
```

Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a good direction.

If the method cannot generate a feasible point (because there aren't any, or because it isn't bright enough to find one), it should return null.

Reimplemented in [CbcBranchCut](#), and [CbcLotsize](#).

Definition at line 169 of file CbcObject.hpp.

```
7.79.3.13 virtual CbcBranchingObject* CbcObject::notPreferredNewFeasible ( ) const [inline],[virtual]
```

Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a bad direction.

If the method cannot generate a feasible point (because there aren't any, or because it isn't bright enough to find one), it should return null.

Reimplemented in [CbcBranchCut](#), and [CbcLotsize](#).

Definition at line 181 of file CbcObject.hpp.

```
7.79.3.14 virtual void CbcObject::resetBounds ( const OsiSolverInterface * ) [inline],[virtual]
```

Reset variable bounds to their original values.

Bounds may be tightened, so it may be good to be able to set this info in object.

Reimplemented in [CbcSimpleInteger](#), and [CbcLotsize](#).

Definition at line 189 of file CbcObject.hpp.

**7.79.3.15** `virtual void CbcObject::floorCeiling ( double & floorValue, double & ceilingValue, double value, double tolerance ) const`  
`[virtual]`

Returns floor and ceiling i.e.

closest valid points

Reimplemented in [CbcLotsize](#).

**7.79.3.16** `virtual CbcObjectUpdateData CbcObject::createUpdateInformation ( const OsiSolverInterface * solver, const CbcNode * node, const CbcBranchingObject * branchingObject )` `[virtual]`

Pass in information on branch just done and create [CbcObjectUpdateData](#) instance.

If object does not need data then backward pointer will be NULL. Assumes can get information from solver

Reimplemented in [CbcSimpleIntegerDynamicPseudoCost](#), and [CbcSOS](#).

**7.79.3.17** `virtual void CbcObject::updateInformation ( const CbcObjectUpdateData & )` `[inline], [virtual]`

Update object by [CbcObjectUpdateData](#).

Reimplemented in [CbcSimpleIntegerDynamicPseudoCost](#), and [CbcSOS](#).

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

**7.79.3.18** `int CbcObject::id ( ) const` `[inline]`

Identifier (normally column number in matrix)

Definition at line 207 of file `CbcObject.hpp`.

**7.79.3.19** `void CbcObject::setId ( int value )` `[inline]`

Set identifier (normally column number in matrix) but 1000000000 to 1100000000 means optional branching object i.e. code would work without it

Definition at line 214 of file `CbcObject.hpp`.

**7.79.3.20** `bool CbcObject::optionalObject ( ) const` `[inline]`

Return true if optional branching object i.e.

code would work without it

Definition at line 220 of file `CbcObject.hpp`.

**7.79.3.21** `int CbcObject::position ( ) const` `[inline]`

Get position in `object_list`.

Definition at line 225 of file `CbcObject.hpp`.

**7.79.3.22** `void CbcObject::setPosition ( int position )` `[inline]`

Set position in `object_list`.

Definition at line 230 of file `CbcObject.hpp`.

**7.79.3.23** `void CbcObject::setModel ( CbcModel * model )` `[inline]`

update model

Definition at line 235 of file CbcObject.hpp.

**7.79.3.24** `CbcModel* CbcObject::model ( ) const` `[inline]`

Return model.

Definition at line 240 of file CbcObject.hpp.

**7.79.3.25** `int CbcObject::preferredWay ( ) const` `[inline]`

If -1 down always chosen first, +1 up always, 0 normal.

Definition at line 245 of file CbcObject.hpp.

**7.79.3.26** `void CbcObject::setPreferredWay ( int value )` `[inline]`

Set -1 down always chosen first, +1 up always, 0 normal.

Definition at line 249 of file CbcObject.hpp.

**7.79.3.27** `virtual void CbcObject::redoSequenceEtc ( CbcModel *, int, const int * )` `[inline],[virtual]`

Redoes data when sequence numbers change.

Reimplemented in [CbcClique](#), [CbcSOS](#), [CbcBranchToFixLots](#), [CbcNWay](#), and [CbcGeneral](#).

Definition at line 253 of file CbcObject.hpp.

**7.79.3.28** `virtual void CbcObject::initializeForBranching ( CbcModel * )` `[inline],[virtual]`

Initialize for branching.

Reimplemented in [CbcIdiotBranch](#).

Definition at line 255 of file CbcObject.hpp.

## 7.79.4 Member Data Documentation

**7.79.4.1** `CbcModel* CbcObject::model_` `[protected]`

data

Model

Definition at line 261 of file CbcObject.hpp.

**7.79.4.2** `int CbcObject::id_` `[protected]`

Identifier (normally column number in matrix)

Definition at line 263 of file CbcObject.hpp.

**7.79.4.3** `int CbcObject::position_` `[protected]`

Position in object list.

Definition at line 265 of file CbcObject.hpp.

**7.79.4.4** `int CbcObject::preferredWay_` `[protected]`

If -1 down always chosen first, +1 up always, 0 normal.

Definition at line 267 of file CbcObject.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcObject.hpp](#)

## 7.80 CbcObjectUpdateData Class Reference

```
#include <CbcObjectUpdateData.hpp>
```

### Public Member Functions

- [CbcObjectUpdateData \(\)](#)  
*Default Constructor.*
- [CbcObjectUpdateData \(CbcObject \\*object, int way, double change, int status, int \[intDecrease\\\_\]\(#\), double branching-Value\)](#)  
*Useful constructor.*
- [CbcObjectUpdateData \(const CbcObjectUpdateData &\)](#)  
*Copy constructor.*
- [CbcObjectUpdateData & operator= \(const CbcObjectUpdateData &rhs\)](#)  
*Assignment operator.*
- [virtual ~CbcObjectUpdateData \(\)](#)  
*Destructor.*

### Public Attributes

- [CbcObject \\* \[object\\\_\]\(#\)](#)  
*data*
- [int \[way\\\_\]\(#\)](#)  
*Branch as defined by instance of [CbcObject](#).*
- [int \[objectNumber\\\_\]\(#\)](#)  
*Object number.*
- [double \[change\\\_\]\(#\)](#)  
*Change in objective.*
- [int \[status\\\_\]\(#\)](#)  
*Status 0 Optimal, 1 infeasible, 2 unknown.*
- [int \[intDecrease\\\_\]\(#\)](#)  
*Decrease in number unsatisfied.*
- [double \[branchingValue\\\_\]\(#\)](#)  
*Branching value.*
- [double \[originalObjective\\\_\]\(#\)](#)  
*Objective value before branching.*
- [double \[cutoff\\\_\]\(#\)](#)  
*Current cutoff.*

### 7.80.1 Detailed Description

Definition at line 14 of file CbcObjectUpdateData.hpp.

## 7.80.2 Constructor & Destructor Documentation

### 7.80.2.1 CbcObjectUpdateData::CbcObjectUpdateData ( )

Default Constructor.

### 7.80.2.2 CbcObjectUpdateData::CbcObjectUpdateData ( CbcObject \* *object*, int *way*, double *change*, int *status*, int *intDecrease\_*, double *branchingValue* )

Useful constructor.

### 7.80.2.3 CbcObjectUpdateData::CbcObjectUpdateData ( const CbcObjectUpdateData & )

Copy constructor.

### 7.80.2.4 virtual CbcObjectUpdateData::~~CbcObjectUpdateData ( ) [virtual]

Destructor.

## 7.80.3 Member Function Documentation

### 7.80.3.1 CbcObjectUpdateData& CbcObjectUpdateData::operator= ( const CbcObjectUpdateData & *rhs* )

Assignment operator.

## 7.80.4 Member Data Documentation

### 7.80.4.1 CbcObject\* CbcObjectUpdateData::object\_

data

Object

Definition at line 43 of file CbcObjectUpdateData.hpp.

### 7.80.4.2 int CbcObjectUpdateData::way\_

Branch as defined by instance of [CbcObject](#).

Definition at line 45 of file CbcObjectUpdateData.hpp.

### 7.80.4.3 int CbcObjectUpdateData::objectNumber\_

Object number.

Definition at line 47 of file CbcObjectUpdateData.hpp.

### 7.80.4.4 double CbcObjectUpdateData::change\_

Change in objective.

Definition at line 49 of file CbcObjectUpdateData.hpp.

### 7.80.4.5 int CbcObjectUpdateData::status\_

Status 0 Optimal, 1 infeasible, 2 unknown.

Definition at line 51 of file CbcObjectUpdateData.hpp.



**7.80.4.6** `int CbcObjectUpdateData::intDecrease_`

Decrease in number unsatisfied.

Definition at line 53 of file CbcObjectUpdateData.hpp.

**7.80.4.7** `double CbcObjectUpdateData::branchingValue_`

Branching value.

Definition at line 55 of file CbcObjectUpdateData.hpp.

**7.80.4.8** `double CbcObjectUpdateData::originalObjective_`

Objective value before branching.

Definition at line 57 of file CbcObjectUpdateData.hpp.

**7.80.4.9** `double CbcObjectUpdateData::cutoff_`

Current cutoff.

Definition at line 59 of file CbcObjectUpdateData.hpp.

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

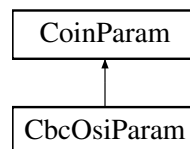
- [/home/ted/COIN/trunk/Cbc/src/CbcObjectUpdateData.hpp](#)

**7.81 CbcOsiParam Class Reference**

Class for control parameters that act on a OsiSolverInterface object.

```
#include <CbcGenOsiParam.hpp>
```

Inheritance diagram for CbcOsiParam:

**Public Types****Subtypes**

- enum [CbcOsiParamCode](#) {  
`CBCOSI_FIRSTPARAM` = CbcCbcParam::CBCCBC\_LASTPARAM + 1, [ALGORITHM](#), [ALLSLACK](#), [AUTOS-](#)

```

CALE,
BARRIER, BARRIERSCALE, BASISIN, BASISOUT,
BIASLU, CHOLESKY, CRASH, CROSSOVER,
DUALBOUND, DUALPIVOT, DUALSIMPLEX, DUALTOLERANCE,
FAKEBOUND, GAMMA, IDIOT, KEEPNames,
KKT, MAXITERATION, MAXHOTITS, NETLIB_BARRIER,
NETLIB_DUAL, NETLIB_PRIMAL, NETWORK, OBJSCALE,
PERTURBATION, PERTVALUE, PFI, PLUSMINUS,
PRESOLVE, PRESOLVEOPTIONS, PRESOLVEPASS, PRIMALPIVOT,
PRIMALSIMPLEX, PRIMALTOLERANCE, REALLY_SCALE, RESTORE,
REVERSE, RHSSCALE, SAVE, SCALING,
SLPVALUE, SOLVERLOGLEVEL, SPARSEFACTOR, SPECIALOPTIONS,
SPRINT, TIGHTEN, CBCOSI_LASTPARAM }

```

*Enumeration for parameters that control an OsiSolverInterface object.*

## Public Member Functions

### Constructors and Destructors

*Be careful how you specify parameters for the constructors! There's great potential for confusion.*

- `CbcOsiParam ()`  
*Default constructor.*
- `CbcOsiParam (CbcOsiParamCode code, 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.*
- `CbcOsiParam (CbcOsiParamCode code, std::string name, std::string help, int lower, int upper, int dflt=0, bool display=true)`  
*Constructor for a parameter with an integer value.*
- `CbcOsiParam (CbcOsiParamCode code, std::string name, std::string help, std::string firstValue, int dflt, bool display=true)`  
*Constructor for a parameter with keyword values.*
- `CbcOsiParam (CbcOsiParamCode code, std::string name, std::string help, std::string dflt, bool display=true)`  
*Constructor for a string parameter.*
- `CbcOsiParam (CbcOsiParamCode code, std::string name, std::string help, bool display=true)`  
*Constructor for an action parameter.*
- `CbcOsiParam (const CbcOsiParam &orig)`  
*Copy constructor.*
- `CbcOsiParam * clone ()`  
*Clone.*
- `CbcOsiParam & operator= (const CbcOsiParam &rhs)`  
*Assignment.*
- `~CbcOsiParam ()`  
*Destructor.*

### Methods to query and manipulate a parameter object

- `CbcOsiParamCode paramCode () const`  
*Get the parameter code.*
- `void setParamCode (CbcOsiParamCode code)`  
*Set the parameter code.*
- `OsiSolverInterface * obj () const`  
*Get the underlying OsiSolverInterface object.*
- `void setObj (OsiSolverInterface *obj)`  
*Set the underlying OsiSolverInterface object.*

### 7.81.1 Detailed Description

Class for control parameters that act on a OsiSolverInterface object.

Adds parameter type codes and push/pull functions to the generic parameter object.

Definition at line 31 of file CbcGenOsiParam.hpp.

### 7.81.2 Member Enumeration Documentation

#### 7.81.2.1 enum CbcOsiParam::CbcOsiParamCode

Enumeration for parameters that control an OsiSolverInterface object.

These are parameters that control the operation of an OsiSolverInterface object. CBCOSI\_FIRSTPARAM and CBCOSI\_LASTPARAM are markers to allow convenient separation of parameter groups.

Enumerator

***CBCOSI\_FIRSTPARAM***  
***ALGORITHM***  
***ALLSLACK***  
***AUTOSCALE***  
***BARRIER***  
***BARRIERSCALE***  
***BASISIN***  
***BASISOUT***  
***BIASLU***  
***CHOLESKY***  
***CRASH***  
***CROSSOVER***  
***DUALBOUND***  
***DUALPIVOT***  
***DUALSIMPLEX***  
***DUALTOLERANCE***  
***FAKEBOUND***  
***GAMMA***  
***IDIOT***  
***KEEPNAMES***  
***KKT***  
***MAXITERATION***  
***MAXHOTITS***  
***NETLIB\_BARRIER***  
***NETLIB\_DUAL***  
***NETLIB\_PRIMAL***  
***NETWORK***  
***OBJSCALE***

***PERTURBATION***  
***PERTVALUE***  
***PFI***  
***PLUSMINUS***  
***PRESOLVE***  
***PRESOLVEOPTIONS***  
***PRESOLVEPASS***  
***PRIMALPIVOT***  
***PRIMALSIMPLEX***  
***PRIMALTOLERANCE***  
***REALLY\_SCALE***  
***RESTORE***  
***REVERSE***  
***RHSSCALE***  
***SAVE***  
***SCALING***  
***SLPVALUE***  
***SOLVERLOGLEVEL***  
***SPARSEFACTOR***  
***SPECIALOPTIONS***  
***SPRINT***  
***TIGHTEN***  
***CBCOSI\_LASTPARAM***

Definition at line 46 of file CbcGenOsiParam.hpp.

### 7.81.3 Constructor & Destructor Documentation

#### 7.81.3.1 CbcOsiParam::CbcOsiParam ( )

Default constructor.

#### 7.81.3.2 CbcOsiParam::CbcOsiParam ( CbcOsiParamCode code, 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.

#### 7.81.3.3 CbcOsiParam::CbcOsiParam ( CbcOsiParamCode code, 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.

**7.81.3.4** `CbcOsiParam::CbcOsiParam ( CbcOsiParamCode code, 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 keyword. Additional keywords can be added using `appendKwd()`. Keywords are numbered from zero. It's necessary to specify both the first keyword (`firstValue`) and the default keyword index (`dflt`) in order to distinguish this constructor from the string and action parameter constructors.

**7.81.3.5** `CbcOsiParam::CbcOsiParam ( CbcOsiParamCode code, std::string name, std::string help, std::string dflt, bool display = true )`

Constructor for a string parameter.

The default string value must be specified explicitly to distinguish a string constructor from an action parameter constructor.

**7.81.3.6** `CbcOsiParam::CbcOsiParam ( CbcOsiParamCode code, std::string name, std::string help, bool display = true )`

Constructor for an action parameter.

**7.81.3.7** `CbcOsiParam::CbcOsiParam ( const CbcOsiParam & orig )`

Copy constructor.

**7.81.3.8** `CbcOsiParam::~~CbcOsiParam ( )`

Destructor.

#### 7.81.4 Member Function Documentation

**7.81.4.1** `CbcOsiParam* CbcOsiParam::clone ( )`

Clone.

**7.81.4.2** `CbcOsiParam& CbcOsiParam::operator= ( const CbcOsiParam & rhs )`

Assignment.

**7.81.4.3** `CbcOsiParamCode CbcOsiParam::paramCode ( ) const [inline]`

Get the parameter code.

Definition at line 141 of file `CbcGenOsiParam.hpp`.

**7.81.4.4** `void CbcOsiParam::setParamCode ( CbcOsiParamCode code ) [inline]`

Set the parameter code.

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

**7.81.4.5** `OsiSolverInterface* CbcOsiParam::obj ( ) const [inline]`

Get the underlying `OsiSolverInterface` object.

Definition at line 153 of file `CbcGenOsiParam.hpp`.

7.81.4.6 `void CbcOsiParam::setObj ( OsiSolverInterface * obj ) [inline]`

Set the underlying OsiSolverInterface object.

Definition at line 159 of file CbcGenOsiParam.hpp.

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

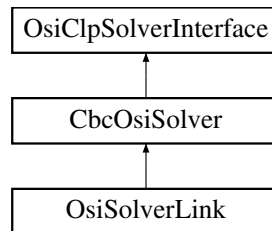
- </home/ted/COIN/trunk/Cbc/src/CbcGenOsiParam.hpp>

## 7.82 CbcOsiSolver Class Reference

This is for codes where solver needs to know about [CbcModel](#) Seems to provide only one value-added feature, a [CbcModel](#) object.

```
#include <CbcFathom.hpp>
```

Inheritance diagram for CbcOsiSolver:



### Public Member Functions

#### Constructors and destructors

- [CbcOsiSolver](#) ()  
*Default Constructor.*
- virtual OsiSolverInterface \* [clone](#) (bool copyData=true) const  
*Clone.*
- [CbcOsiSolver](#) (const [CbcOsiSolver](#) &)  
*Copy constructor.*
- [CbcOsiSolver](#) & [operator=](#) (const [CbcOsiSolver](#) &rhs)  
*Assignment operator.*
- virtual [~CbcOsiSolver](#) ()  
*Destructor.*

#### Sets and Gets

- void [setCbcModel](#) ([CbcModel](#) \*model)  
*Set Cbc Model.*
- [CbcModel](#) \* [cbcModel](#) () const  
*Return Cbc Model.*

### Protected Attributes

#### Private member data

- [CbcModel](#) \* [cbcModel\\_](#)  
*Pointer back to [CbcModel](#).*

### 7.82.1 Detailed Description

This is for codes where solver needs to know about [CbcModel](#) Seems to provide only one value-added feature, a [CbcModel](#) object.

Definition at line 90 of file CbcFathom.hpp.

### 7.82.2 Constructor & Destructor Documentation

#### 7.82.2.1 CbcOsiSolver::CbcOsiSolver ( )

Default Constructor.

#### 7.82.2.2 CbcOsiSolver::CbcOsiSolver ( const CbcOsiSolver & )

Copy constructor.

#### 7.82.2.3 virtual CbcOsiSolver::~~CbcOsiSolver ( ) [virtual]

Destructor.

### 7.82.3 Member Function Documentation

#### 7.82.3.1 virtual OsiSolverInterface\* CbcOsiSolver::clone ( bool *copyData* = true ) const [virtual]

Clone.

Reimplemented in [OsiSolverLink](#).

#### 7.82.3.2 CbcOsiSolver& CbcOsiSolver::operator= ( const CbcOsiSolver & *rhs* )

Assignment operator.

#### 7.82.3.3 void CbcOsiSolver::setCbcModel ( CbcModel \* *model* ) [inline]

Set Cbc Model.

Definition at line 117 of file CbcFathom.hpp.

#### 7.82.3.4 CbcModel\* CbcOsiSolver::cbcModel ( ) const [inline]

Return Cbc Model.

Definition at line 121 of file CbcFathom.hpp.

### 7.82.4 Member Data Documentation

#### 7.82.4.1 CbcModel\* CbcOsiSolver::cbcModel\_ [protected]

Pointer back to [CbcModel](#).

Definition at line 134 of file CbcFathom.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcFathom.hpp](#)

### 7.83 CbcParam Class Reference

Very simple class for setting parameters.

```
#include <CbcParam.hpp>
```

#### Public Member Functions

##### Constructor and destructor

- [CbcParam](#) ()  
*Constructors.*
- [CbcParam](#) (std::string [name](#), std::string help, double lower, double upper, [CbcParameterType](#) [type](#), bool display=true)
- [CbcParam](#) (std::string [name](#), std::string help, int lower, int upper, [CbcParameterType](#) [type](#), bool display=true)
- [CbcParam](#) (std::string [name](#), std::string help, std::string firstValue, [CbcParameterType](#) [type](#), int defaultIndex=0, bool display=true)
- [CbcParam](#) (std::string [name](#), std::string help, [CbcParameterType](#) [type](#), int [indexNumber](#)=-1, bool display=true)
- [CbcParam](#) (const [CbcParam](#) &)
- [CbcParam](#) & [operator=](#) (const [CbcParam](#) &rhs)  
*Copy constructor.*
- [CbcParam](#) & [operator=](#) (const [CbcParam](#) &rhs)  
*Assignment operator. This copies the data.*
- [~CbcParam](#) ()  
*Destructor.*

#### stuff

- void [append](#) (std::string keyWord)  
*Insert string (only valid for keywords)*
- void [addHelp](#) (std::string keyWord)  
*Adds one help line.*
- std::string [name](#) () const  
*Returns name.*
- std::string [shortHelp](#) () const  
*Returns short help.*
- int [setDoubleParameter](#) ([CbcModel](#) &model, double value) const  
*Sets a double parameter (nonzero code if error)*
- double [doubleParameter](#) ([CbcModel](#) &model) const  
*Gets a double parameter.*
- int [setIntParameter](#) ([CbcModel](#) &model, int value) const  
*Sets a int parameter (nonzero code if error)*
- int [intParameter](#) ([CbcModel](#) &model) const  
*Gets a int parameter.*
- int [setDoubleParameter](#) (ClpSimplex \*model, double value) const  
*Sets a double parameter (nonzero code if error)*
- double [doubleParameter](#) (ClpSimplex \*model) const  
*Gets a double parameter.*
- int [setIntParameter](#) (ClpSimplex \*model, int value) const  
*Sets a int parameter (nonzero code if error)*
- int [intParameter](#) (ClpSimplex \*model) const



- Gets a int parameter.*

  - int [setDoubleParameter](#) (OsiSolverInterface \*model, double value) const

*Sets a double parameter (nonzero code if error)*
- double [doubleParameter](#) (OsiSolverInterface \*model) const

*Gets a double parameter.*
- int [setIntParameter](#) (OsiSolverInterface \*model, int value) const

*Sets a int parameter (nonzero code if error)*
- int [intParameter](#) (OsiSolverInterface \*model) const

*Gets a int parameter.*
- int [checkDoubleParameter](#) (double value) const

*Checks a double parameter (nonzero code if error)*
- std::string [matchName](#) () const

*Returns name which could match.*
- int [parameterOption](#) (std::string check) const

*Returns parameter option which matches (-1 if none)*
- void [printOptions](#) () const

*Prints parameter options.*
- std::string [currentOption](#) () const

*Returns current parameter option.*
- void [setCurrentOption](#) (int value)

*Sets current parameter option.*
- void [setIntValue](#) (int value)

*Sets int value.*
- int [intValue](#) () const
- void [setDoubleValue](#) (double value)

*Sets double value.*
- double [doubleValue](#) () const
- void [setStringValue](#) (std::string value)

*Sets string value.*
- std::string [stringValue](#) () const
- int [matches](#) (std::string input) const

*Returns 1 if matches minimum, 2 if matches less, 0 if not matched.*
- [CbcParameterType](#) [type](#) () const

*type*
- bool [displayThis](#) () const

*whether to display*
- void [setLonghelp](#) (const std::string help)

*Set Long help.*
- void [printLongHelp](#) () const

*Print Long help.*
- void [printString](#) () const

*Print action and string.*
- int [indexNumber](#) () const

*type for classification*

### 7.83.1 Detailed Description

Very simple class for setting parameters.

Definition at line 153 of file CbcParam.hpp.

### 7.83.2 Constructor & Destructor Documentation

#### 7.83.2.1 CbcParam::CbcParam ( )

Constructors.

**7.83.2.2 CbcParam::CbcParam ( std::string *name*, std::string *help*, double *lower*, double *upper*, CbcParameterType *type*, bool *display* = true )**

**7.83.2.3 CbcParam::CbcParam ( std::string *name*, std::string *help*, int *lower*, int *upper*, CbcParameterType *type*, bool *display* = true )**

**7.83.2.4 CbcParam::CbcParam ( std::string *name*, std::string *help*, std::string *firstValue*, CbcParameterType *type*, int *defaultIndex* = 0, bool *display* = true )**

**7.83.2.5 CbcParam::CbcParam ( std::string *name*, std::string *help*, CbcParameterType *type*, int *indexNumber* = -1, bool *display* = true )**

**7.83.2.6 CbcParam::CbcParam ( const CbcParam & )**

Copy constructor.

**7.83.2.7 CbcParam::~~CbcParam ( )**

Destructor.

### 7.83.3 Member Function Documentation

**7.83.3.1 CbcParam& CbcParam::operator=( const CbcParam & *rhs* )**

Assignment operator. This copies the data.

**7.83.3.2 void CbcParam::append ( std::string *keyWord* )**

Insert string (only valid for keywords)

**7.83.3.3 void CbcParam::addHelp ( std::string *keyWord* )**

Adds one help line.

**7.83.3.4 std::string CbcParam::name ( ) const [inline]**

Returns name.

Definition at line 186 of file CbcParam.hpp.

**7.83.3.5 std::string CbcParam::shortHelp ( ) const [inline]**

Returns short help.

Definition at line 190 of file CbcParam.hpp.

7.83.3.6 `int CbcParam::setDoubleParameter ( CbcModel & model, double value ) const`

Sets a double parameter (nonzero code if error)

7.83.3.7 `double CbcParam::doubleParameter ( CbcModel & model ) const`

Gets a double parameter.

7.83.3.8 `int CbcParam::setIntParameter ( CbcModel & model, int value ) const`

Sets a int parameter (nonzero code if error)

7.83.3.9 `int CbcParam::intParameter ( CbcModel & model ) const`

Gets a int parameter.

7.83.3.10 `int CbcParam::setDoubleParameter ( ClpSimplex * model, double value ) const`

Sets a double parameter (nonzero code if error)

7.83.3.11 `double CbcParam::doubleParameter ( ClpSimplex * model ) const`

Gets a double parameter.

7.83.3.12 `int CbcParam::setIntParameter ( ClpSimplex * model, int value ) const`

Sets a int parameter (nonzero code if error)

7.83.3.13 `int CbcParam::intParameter ( ClpSimplex * model ) const`

Gets a int parameter.

7.83.3.14 `int CbcParam::setDoubleParameter ( OsiSolverInterface * model, double value ) const`

Sets a double parameter (nonzero code if error)

7.83.3.15 `double CbcParam::doubleParameter ( OsiSolverInterface * model ) const`

Gets a double parameter.

7.83.3.16 `int CbcParam::setIntParameter ( OsiSolverInterface * model, int value ) const`

Sets a int parameter (nonzero code if error)

7.83.3.17 `int CbcParam::intParameter ( OsiSolverInterface * model ) const`

Gets a int parameter.

7.83.3.18 `int CbcParam::checkDoubleParameter ( double value ) const`

Checks a double parameter (nonzero code if error)

7.83.3.19 `std::string CbcParam::matchName ( ) const`

Returns name which could match.

**7.83.3.20** `int CbcParam::parameterOption ( std::string check ) const`

Returns parameter option which matches (-1 if none)

**7.83.3.21** `void CbcParam::printOptions ( ) const`

Prints parameter options.

**7.83.3.22** `std::string CbcParam::currentOption ( ) const` `[inline]`

Returns current parameter option.

Definition at line 226 of file CbcParam.hpp.

**7.83.3.23** `void CbcParam::setCurrentOption ( int value )` `[inline]`

Sets current parameter option.

Definition at line 230 of file CbcParam.hpp.

**7.83.3.24** `void CbcParam::setIntValue ( int value )` `[inline]`

Sets int value.

Definition at line 234 of file CbcParam.hpp.

**7.83.3.25** `int CbcParam::intValue ( ) const` `[inline]`

Definition at line 237 of file CbcParam.hpp.

**7.83.3.26** `void CbcParam::setDoubleValue ( double value )` `[inline]`

Sets double value.

Definition at line 241 of file CbcParam.hpp.

**7.83.3.27** `double CbcParam::doubleValue ( ) const` `[inline]`

Definition at line 244 of file CbcParam.hpp.

**7.83.3.28** `void CbcParam::setStringValue ( std::string value )` `[inline]`

Sets string value.

Definition at line 248 of file CbcParam.hpp.

**7.83.3.29** `std::string CbcParam::stringValue ( ) const` `[inline]`

Definition at line 251 of file CbcParam.hpp.

**7.83.3.30** `int CbcParam::matches ( std::string input ) const`

Returns 1 if matches minimum, 2 if matches less, 0 if not matched.

**7.83.3.31** `CbcParameterType CbcParam::type ( ) const` `[inline]`

type

Definition at line 257 of file CbcParam.hpp.

7.83.3.32 `bool CbcParam::displayThis ( ) const` `[inline]`

whether to display

Definition at line 261 of file CbcParam.hpp.

7.83.3.33 `void CbcParam::setLonghelp ( const std::string help )` `[inline]`

Set Long help.

Definition at line 265 of file CbcParam.hpp.

7.83.3.34 `void CbcParam::printLongHelp ( ) const`

Print Long help.

7.83.3.35 `void CbcParam::printString ( ) const`

Print action and string.

7.83.3.36 `int CbcParam::indexNumber ( ) const` `[inline]`

type for classification

Definition at line 273 of file CbcParam.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcParam.hpp](#)

## 7.84 CbcGenCtlBlk::cbcParamsInfo\_struct Struct Reference

Start and end of [CbcModel](#) parameters in parameter vector.

```
#include <CbcGenCtlBlk.hpp>
```

### Public Attributes

- int [first\\_](#)
- int [last\\_](#)

#### 7.84.1 Detailed Description

Start and end of [CbcModel](#) parameters in parameter vector.

Definition at line 605 of file CbcGenCtlBlk.hpp.

#### 7.84.2 Member Data Documentation

7.84.2.1 `int CbcGenCtlBlk::cbcParamsInfo_struct::first_`

Definition at line 606 of file CbcGenCtlBlk.hpp.

7.84.2.2 `int CbcGenCtlBlk::cbcParamsInfo_struct::last_`

Definition at line 607 of file CbcGenCtlBlk.hpp.

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

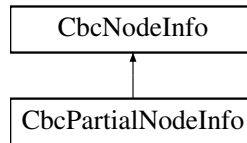
- /home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp

## 7.85 CbcPartialNodeInfo Class Reference

Holds information for recreating a subproblem by incremental change from the parent.

```
#include <CbcPartialNodeInfo.hpp>
```

Inheritance diagram for CbcPartialNodeInfo:



### Public Member Functions

- virtual void [applyToModel](#) ([CbcModel](#) \*model, [CoinWarmStartBasis](#) \*&basis, [CbcCountRowCut](#) \*\*addCuts, int &currentNumberCuts) const  
*Modify model according to information at node.*
- virtual int [applyBounds](#) (int iColumn, double &lower, double &upper, int force)  
*Just apply bounds to one variable - force means overwrite by lower, upper (1=>infeasible)*
- virtual [CbcNodeInfo](#) \* [buildRowBasis](#) ([CoinWarmStartBasis](#) &basis) const  
*Builds up row basis backwards (until original model).*
- [CbcPartialNodeInfo](#) ()
- [CbcPartialNodeInfo](#) ([CbcNodeInfo](#) \*parent, [CbcNode](#) \*owner, int numberChangedBounds, const int \*variables, const double \*boundChanges, const [CoinWarmStartDiff](#) \*basisDiff)
- [CbcPartialNodeInfo](#) (const [CbcPartialNodeInfo](#) &)
- ~[CbcPartialNodeInfo](#) ()
- virtual [CbcNodeInfo](#) \* [clone](#) () const  
*Clone.*
- const [CoinWarmStartDiff](#) \* [basisDiff](#) () const  
*Basis diff information.*
- const int \* [variables](#) () const  
*Which variable (top bit if upper bound changing)*
- const double \* [newBounds](#) () const
- int [numberChangedBounds](#) () const  
*Number of bound changes.*

### Protected Attributes

- [CoinWarmStartDiff](#) \* [basisDiff\\_](#)  
*Basis diff information.*
- int \* [variables\\_](#)  
*Which variable (top bit if upper bound changing)*
- double \* [newBounds\\_](#)
- int [numberChangedBounds\\_](#)  
*Number of bound changes.*

## 7.85.1 Detailed Description

Holds information for recreating a subproblem by incremental change from the parent.

A [CbcPartialNodeInfo](#) object contains changes to the bounds and basis, and additional cuts, required to recreate a subproblem by modifying and augmenting the parent subproblem.

Definition at line 39 of file CbcPartialNodeInfo.hpp.

## 7.85.2 Constructor &amp; Destructor Documentation

7.85.2.1 `CbcPartialNodeInfo::CbcPartialNodeInfo ( )`

7.85.2.2 `CbcPartialNodeInfo::CbcPartialNodeInfo ( CbcNodeInfo * parent, CbcNode * owner, int numberChangedBounds, const int * variables, const double * boundChanges, const CoinWarmStartDiff * basisDiff )`

7.85.2.3 `CbcPartialNodeInfo::CbcPartialNodeInfo ( const CbcPartialNodeInfo & )`

7.85.2.4 `CbcPartialNodeInfo::~~CbcPartialNodeInfo ( )`

## 7.85.3 Member Function Documentation

7.85.3.1 `virtual void CbcPartialNodeInfo::applyToModel ( CbcModel * model, CoinWarmStartBasis *& basis, CbcCountRowCut ** addCuts, int & currentNumberCuts ) const [virtual]`

Modify model according to information at node.

The routine modifies the model according to bound and basis change information at node and adds any cuts to the addCuts array.

Implements [CbcNodeInfo](#).

7.85.3.2 `virtual int CbcPartialNodeInfo::applyBounds ( int iColumn, double & lower, double & upper, int force ) [virtual]`

Just apply bounds to one variable - force means overwrite by lower,upper (1=>infeasible)

Implements [CbcNodeInfo](#).

7.85.3.3 `virtual CbcNodeInfo* CbcPartialNodeInfo::buildRowBasis ( CoinWarmStartBasis & basis ) const [virtual]`

Builds up row basis backwards (until original model).

Returns NULL or previous one to apply . Depends on Free being 0 and impossible for cuts

Implements [CbcNodeInfo](#).

7.85.3.4 `virtual CbcNodeInfo* CbcPartialNodeInfo::clone ( ) const [virtual]`

Clone.

Implements [CbcNodeInfo](#).

7.85.3.5 `const CoinWarmStartDiff* CbcPartialNodeInfo::basisDiff ( ) const [inline]`

Basis diff information.

Definition at line 77 of file CbcPartialNodeInfo.hpp.

**7.85.3.6** `const int* CbcPartialNodeInfo::variables ( ) const [inline]`

Which variable (top bit if upper bound changing)

Definition at line 81 of file CbcPartialNodeInfo.hpp.

**7.85.3.7** `const double* CbcPartialNodeInfo::newBounds ( ) const [inline]`

Definition at line 85 of file CbcPartialNodeInfo.hpp.

**7.85.3.8** `int CbcPartialNodeInfo::numberChangedBounds ( ) const [inline]`

Number of bound changes.

Definition at line 89 of file CbcPartialNodeInfo.hpp.

#### 7.85.4 Member Data Documentation

**7.85.4.1** `CoinWarmStartDiff* CbcPartialNodeInfo::basisDiff_ [protected]`

Basis diff information.

Definition at line 96 of file CbcPartialNodeInfo.hpp.

**7.85.4.2** `int* CbcPartialNodeInfo::variables_ [protected]`

Which variable (top bit if upper bound changing)

Definition at line 98 of file CbcPartialNodeInfo.hpp.

**7.85.4.3** `double* CbcPartialNodeInfo::newBounds_ [protected]`

Definition at line 100 of file CbcPartialNodeInfo.hpp.

**7.85.4.4** `int CbcPartialNodeInfo::numberChangedBounds_ [protected]`

Number of bound changes.

Definition at line 102 of file CbcPartialNodeInfo.hpp.

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

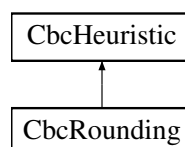
- /home/ted/COIN/trunk/Cbc/src/CbcPartialNodeInfo.hpp

## 7.86 CbcRounding Class Reference

Rounding class.

```
#include <CbcHeuristic.hpp>
```

Inheritance diagram for CbcRounding:





## Public Member Functions

- [CbcRounding](#) ()
- [CbcRounding](#) ([CbcModel](#) &model)
- [CbcRounding](#) (const [CbcRounding](#) &)
- [~CbcRounding](#) ()
- [CbcRounding](#) & [operator=](#) (const [CbcRounding](#) &rhs)  
*Assignment operator.*
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model (This is needed if cliques update matrix etc)*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution, double solutionValue)  
*returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts Use solutionValue rather than solvers one*
- virtual void [validate](#) ()  
*Validate model i.e. sets when\_ to 0 if necessary (may be NULL)*
- void [setSeed](#) (int value)  
*Set seed.*

## Protected Attributes

- CoinPackedMatrix [matrix\\_](#)
- CoinPackedMatrix [matrixByRow\\_](#)
- unsigned short \* [down\\_](#)
- unsigned short \* [up\\_](#)
- unsigned short \* [equal\\_](#)
- int [seed\\_](#)

## 7.86.1 Detailed Description

Rounding class.

Definition at line 407 of file CbcHeuristic.hpp.

## 7.86.2 Constructor &amp; Destructor Documentation

## 7.86.2.1 CbcRounding::CbcRounding ( )

7.86.2.2 CbcRounding::CbcRounding ( [CbcModel](#) & *model* )

7.86.2.3 `CbcRounding::CbcRounding ( const CbcRounding & )`

7.86.2.4 `CbcRounding::~~CbcRounding ( )`

7.86.3 Member Function Documentation

7.86.3.1 `CbcRounding& CbcRounding::operator= ( const CbcRounding & rhs )`

Assignment operator.

7.86.3.2 `virtual CbcHeuristic* CbcRounding::clone ( ) const [virtual]`

Clone.

Implements [CbcHeuristic](#).

7.86.3.3 `virtual void CbcRounding::generateCpp ( FILE * fp ) [virtual]`

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.86.3.4 `virtual void CbcRounding::resetModel ( CbcModel * model ) [virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

7.86.3.5 `virtual void CbcRounding::setModel ( CbcModel * model ) [virtual]`

update model (This is needed if cliques update matrix etc)

Reimplemented from [CbcHeuristic](#).

7.86.3.6 `virtual int CbcRounding::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts

Implements [CbcHeuristic](#).

7.86.3.7 `virtual int CbcRounding::solution ( double & objectiveValue, double * newSolution, double solutionValue ) [virtual]`

returns 0 if no solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value (only if good) This is called after cuts have been added - so can not add cuts Use solutionValue rather than solvers one

7.86.3.8 `virtual void CbcRounding::validate ( ) [virtual]`

Validate model i.e. sets when\_ to 0 if necessary (may be NULL)

Reimplemented from [CbcHeuristic](#).

7.86.3.9 `void CbcRounding::setSeed ( int value ) [inline]`

Set seed.

Definition at line 458 of file CbcHeuristic.hpp.

## 7.86.4 Member Data Documentation

## 7.86.4.1 CoinPackedMatrix CbcRounding::matrix\_ [protected]

Definition at line 466 of file CbcHeuristic.hpp.

## 7.86.4.2 CoinPackedMatrix CbcRounding::matrixByRow\_ [protected]

Definition at line 469 of file CbcHeuristic.hpp.

## 7.86.4.3 unsigned short\* CbcRounding::down\_ [protected]

Definition at line 472 of file CbcHeuristic.hpp.

## 7.86.4.4 unsigned short\* CbcRounding::up\_ [protected]

Definition at line 475 of file CbcHeuristic.hpp.

## 7.86.4.5 unsigned short\* CbcRounding::equal\_ [protected]

Definition at line 478 of file CbcHeuristic.hpp.

## 7.86.4.6 int CbcRounding::seed\_ [protected]

Definition at line 481 of file CbcHeuristic.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp](#)

## 7.87 CbcRowCuts Class Reference

```
#include <CbcCountRowCut.hpp>
```

## Public Member Functions

- [CbcRowCuts](#) (int initialMaxSize=0, int hashMultiplier=4)
- [~CbcRowCuts](#) ()
- [CbcRowCuts](#) (const [CbcRowCuts](#) &rhs)
- [CbcRowCuts](#) & [operator=](#) (const [CbcRowCuts](#) &rhs)
- [OsiRowCut2](#) \* [cut](#) (int sequence) const
- int [numberCuts](#) () const
- int [sizeRowCuts](#) () const
- [OsiRowCut](#) \* [rowCutPtr](#) (int sequence)
- void [eraseRowCut](#) (int sequence)
- int [addCutIfNotDuplicate](#) (const [OsiRowCut](#) &[cut](#), int whichType=0)
- int [addCutIfNotDuplicateWhenGreedy](#) (const [OsiRowCut](#) &[cut](#), int whichType=0)
- void [addCuts](#) ([OsiCuts](#) &cs)

## 7.87.1 Detailed Description

Definition at line 134 of file CbcCountRowCut.hpp.

### 7.87.2 Constructor & Destructor Documentation

7.87.2.1 `CbcRowCuts::CbcRowCuts ( int initialMaxSize = 0, int hashMultiplier = 4 )`

7.87.2.2 `CbcRowCuts::~~CbcRowCuts ( )`

7.87.2.3 `CbcRowCuts::CbcRowCuts ( const CbcRowCuts & rhs )`

### 7.87.3 Member Function Documentation

7.87.3.1 `CbcRowCuts& CbcRowCuts::operator= ( const CbcRowCuts & rhs )`

7.87.3.2 `OsiRowCut2* CbcRowCuts::cut ( int sequence ) const` `[inline]`

Definition at line 141 of file `CbcCountRowCut.hpp`.

7.87.3.3 `int CbcRowCuts::numberCuts ( ) const` `[inline]`

Definition at line 143 of file `CbcCountRowCut.hpp`.

7.87.3.4 `int CbcRowCuts::sizeRowCuts ( ) const` `[inline]`

Definition at line 145 of file `CbcCountRowCut.hpp`.

7.87.3.5 `OsiRowCut* CbcRowCuts::rowCutPtr ( int sequence )` `[inline]`

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

7.87.3.6 `void CbcRowCuts::eraseRowCut ( int sequence )`

7.87.3.7 `int CbcRowCuts::addCutIfNotDuplicate ( const OsiRowCut & cut, int whichType = 0 )`

7.87.3.8 `int CbcRowCuts::addCutIfNotDuplicateWhenGreedy ( const OsiRowCut & cut, int whichType = 0 )`

7.87.3.9 `void CbcRowCuts::addCuts ( OsiCuts & cs )`

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

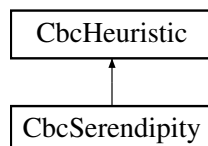
- [/home/ted/COIN/trunk/Cbc/src/CbcCountRowCut.hpp](#)

## 7.88 CbcSerendipity Class Reference

heuristic - just picks up any good solution found by solver - see `OsiBabSolver`

```
#include <CbcHeuristic.hpp>
```

Inheritance diagram for `CbcSerendipity`:



## Public Member Functions

- [CbcSerendipity](#) ()
- [CbcSerendipity](#) ([CbcModel](#) &model)
- [CbcSerendipity](#) (const [CbcSerendipity](#) &)
- [~CbcSerendipity](#) ()
- [CbcSerendipity](#) & [operator=](#) (const [CbcSerendipity](#) &rhs)  
*Assignment operator.*
- virtual [CbcHeuristic](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [setModel](#) ([CbcModel](#) \*model)  
*update model*
- virtual int [solution](#) (double &objectiveValue, double \*newSolution)  
*returns 0 if no solution, 1 if valid solution.*
- virtual void [resetModel](#) ([CbcModel](#) \*model)  
*Resets stuff if model changes.*

## Additional Inherited Members

## 7.88.1 Detailed Description

heuristic - just picks up any good solution found by solver - see OsiBabSolver

Definition at line 552 of file CbcHeuristic.hpp.

## 7.88.2 Constructor &amp; Destructor Documentation

7.88.2.1 [CbcSerendipity::CbcSerendipity](#) ( )

7.88.2.2 [CbcSerendipity::CbcSerendipity](#) ( [CbcModel](#) & *model* )

7.88.2.3 [CbcSerendipity::CbcSerendipity](#) ( const [CbcSerendipity](#) & )

7.88.2.4 [CbcSerendipity::~~CbcSerendipity](#) ( )

## 7.88.3 Member Function Documentation

7.88.3.1 [CbcSerendipity&](#) [CbcSerendipity::operator=](#) ( const [CbcSerendipity](#) & *rhs* )

Assignment operator.

7.88.3.2 virtual [CbcHeuristic](#)\* [CbcSerendipity::clone](#) ( ) const [virtual]

Clone.

Implements [CbcHeuristic](#).

7.88.3.3 virtual void [CbcSerendipity::generateCpp](#) ( FILE \* *fp* ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcHeuristic](#).

7.88.3.4 `virtual void CbcSerendipity::setModel ( CbcModel * model ) [virtual]`

update model

Reimplemented from [CbcHeuristic](#).

7.88.3.5 `virtual int CbcSerendipity::solution ( double & objectiveValue, double * newSolution ) [virtual]`

returns 0 if no solution, 1 if valid solution.

Sets solution values if good, sets objective value (only if good) We leave all variables which are at one at this node of the tree to that value and will initially set all others to zero. We then sort all variables in order of their cost divided by the number of entries in rows which are not yet covered. We randomize that value a bit so that ties will be broken in different ways on different runs of the heuristic. We then choose the best one and set it to one and repeat the exercise.

Implements [CbcHeuristic](#).

7.88.3.6 `virtual void CbcSerendipity::resetModel ( CbcModel * model ) [virtual]`

Resets stuff if model changes.

Implements [CbcHeuristic](#).

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

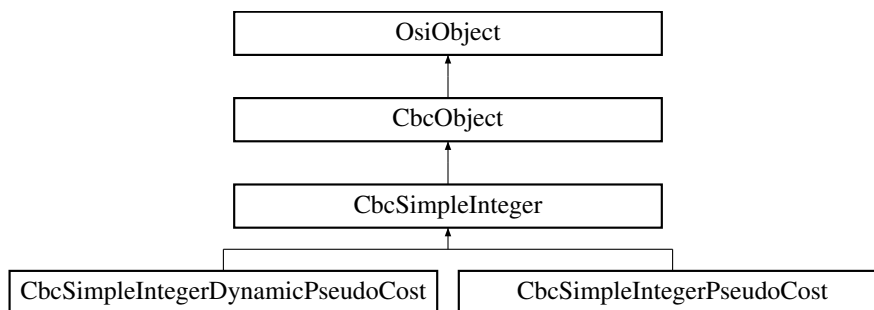
- `/home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp`

## 7.89 CbcSimpleInteger Class Reference

Define a single integer class.

```
#include <CbcSimpleInteger.hpp>
```

Inheritance diagram for CbcSimpleInteger:



### Public Member Functions

- [CbcSimpleInteger](#) ()
- [CbcSimpleInteger](#) (CbcModel \*model, int iColumn, double breakEven=0.5)
- [CbcSimpleInteger](#) (CbcModel \*model, const OsiSimpleInteger \*object)
- [CbcSimpleInteger](#) (const [CbcSimpleInteger](#) &)
- virtual [CbcObject](#) \* clone () const
- *Clone.*
- [CbcSimpleInteger](#) & operator= (const [CbcSimpleInteger](#) &rhs)
- virtual [~CbcSimpleInteger](#) ()

- OsiSimpleInteger \* [osiObject](#) () const  
*Construct an OsiSimpleInteger object.*
- virtual double [infeasibility](#) (const OsiBranchingInformation \*info, int &[preferredWay](#)) const  
*Infeasibility - large is 0.5.*
- virtual double [feasibleRegion](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info) const  
*Set bounds to fix the variable at the current (integer) value.*
- virtual [CbcBranchingObject](#) \* [createCbcBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way)  
*Create a branching object and indicate which way to branch first.*
- void [fillCreateBranch](#) ([CbcIntegerBranchingObject](#) \*branching, const OsiBranchingInformation \*info, int way)  
*Fills in a created branching object.*
- virtual OsiSolverBranch \* [solverBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info) const  
*Create an OsiSolverBranch object.*
- virtual void [feasibleRegion](#) ()  
*Set bounds to fix the variable at the current (integer) value.*
- virtual int [columnNumber](#) () const  
*Column number if single column object -1 otherwise, so returns  $\geq 0$  Used by heuristics.*
- void [setColumnNumber](#) (int value)  
*Set column number.*
- virtual void [resetBounds](#) (const OsiSolverInterface \*solver)  
*Reset variable bounds to their original values.*
- virtual void [resetSequenceEtc](#) (int numberColumns, const int \*originalColumns)  
*Change column numbers after preprocessing.*
- double [originalLowerBound](#) () const  
*Original bounds.*
- void [setOriginalLowerBound](#) (double value)
- double [originalUpperBound](#) () const
- void [setOriginalUpperBound](#) (double value)
- double [breakEven](#) () const  
*Breakeven e.g 0.7 ->  $\geq 0.7$  go up first.*
- void [setBreakEven](#) (double value)  
*Set breakeven e.g 0.7 ->  $\geq 0.7$  go up first.*

#### Protected Attributes

- double [originalLower\\_](#)  
*data*
- double [originalUpper\\_](#)  
*Original upper bound.*
- double [breakEven\\_](#)  
*Breakeven i.e.  $\geq$  this preferred is up.*
- int [columnNumber\\_](#)  
*Column number in model.*
- int [preferredWay\\_](#)  
*If -1 down always chosen first, +1 up always, 0 normal.*

### 7.89.1 Detailed Description

Define a single integer class.

Definition at line 167 of file CbcSimpleInteger.hpp.

### 7.89.2 Constructor & Destructor Documentation

7.89.2.1 `CbcSimpleInteger::CbcSimpleInteger ( )`

7.89.2.2 `CbcSimpleInteger::CbcSimpleInteger ( CbcModel * model, int iColumn, double breakEven = 0.5 )`

7.89.2.3 `CbcSimpleInteger::CbcSimpleInteger ( CbcModel * model, const OsiSimpleInteger * object )`

7.89.2.4 `CbcSimpleInteger::CbcSimpleInteger ( const CbcSimpleInteger & )`

7.89.2.5 `virtual CbcSimpleInteger::~CbcSimpleInteger ( ) [virtual]`

### 7.89.3 Member Function Documentation

7.89.3.1 `virtual CbcObject* CbcSimpleInteger::clone ( ) const [virtual]`

Clone.

Implements [CbcObject](#).

Reimplemented in [CbcSimpleIntegerDynamicPseudoCost](#), and [CbcSimpleIntegerPseudoCost](#).

7.89.3.2 `CbcSimpleInteger& CbcSimpleInteger::operator= ( const CbcSimpleInteger & rhs )`

7.89.3.3 `OsiSimpleInteger* CbcSimpleInteger::osiObject ( ) const`

Construct an OsiSimpleInteger object.

7.89.3.4 `virtual double CbcSimpleInteger::infeasibility ( const OsiBranchingInformation * info, int & preferredWay ) const [virtual]`

Infeasibility - large is 0.5.

Reimplemented from [CbcObject](#).

Reimplemented in [CbcSimpleIntegerDynamicPseudoCost](#), and [CbcSimpleIntegerPseudoCost](#).

7.89.3.5 `virtual double CbcSimpleInteger::feasibleRegion ( OsiSolverInterface * solver, const OsiBranchingInformation * info ) const [virtual]`

Set bounds to fix the variable at the current (integer) value.

Given an integer value, set the lower and upper bounds to fix the variable. Returns amount it had to move variable.

Reimplemented from [CbcObject](#).

7.89.3.6 `virtual CbcBranchingObject* CbcSimpleInteger::createCbcBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way ) [virtual]`

Create a branching object and indicate which way to branch first.

The branching object has to know how to create branches (fix variables, etc.)

Reimplemented from [CbcObject](#).



Reimplemented in [CbcSimpleIntegerDynamicPseudoCost](#), and [CbcSimpleIntegerPseudoCost](#).

**7.89.3.7** `void CbcSimpleInteger::fillCreateBranch ( CbcIntegerBranchingObject * branching, const OsiBranchingInformation * info, int way )`

Fills in a created branching object.

**7.89.3.8** `virtual OsiSolverBranch* CbcSimpleInteger::solverBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info ) const` `[virtual]`

Create an OsiSolverBranch object.

This returns NULL if branch not represented by bound changes

**7.89.3.9** `virtual void CbcSimpleInteger::feasibleRegion ( )` `[virtual]`

Set bounds to fix the variable at the current (integer) value.

Given an integer value, set the lower and upper bounds to fix the variable. The algorithm takes a bit of care in order to compensate for minor numerical inaccuracy.

Implements [CbcObject](#).

**7.89.3.10** `virtual int CbcSimpleInteger::columnNumber ( ) const` `[virtual]`

Column number if single column object -1 otherwise, so returns  $\geq 0$  Used by heuristics.

**7.89.3.11** `void CbcSimpleInteger::setColumnNumber ( int value )` `[inline]`

Set column number.

Definition at line 235 of file CbcSimpleInteger.hpp.

**7.89.3.12** `virtual void CbcSimpleInteger::resetBounds ( const OsiSolverInterface * solver )` `[virtual]`

Reset variable bounds to their original values.

Bounds may be tightened, so it may be good to be able to set this info in object.

Reimplemented from [CbcObject](#).

**7.89.3.13** `virtual void CbcSimpleInteger::resetSequenceEtc ( int numberColumns, const int * originalColumns )` `[virtual]`

Change column numbers after preprocessing.

**7.89.3.14** `double CbcSimpleInteger::originalLowerBound ( ) const` `[inline]`

Original bounds.

Definition at line 249 of file CbcSimpleInteger.hpp.

**7.89.3.15** `void CbcSimpleInteger::setOriginalLowerBound ( double value )` `[inline]`

Definition at line 252 of file CbcSimpleInteger.hpp.

**7.89.3.16** `double CbcSimpleInteger::originalUpperBound ( ) const` `[inline]`

Definition at line 255 of file CbcSimpleInteger.hpp.

**7.89.3.17** `void CbcSimpleInteger::setOriginalUpperBound ( double value ) [inline]`

Definition at line 258 of file CbcSimpleInteger.hpp.

**7.89.3.18** `double CbcSimpleInteger::breakEven ( ) const [inline]`

Breakeven e.g 0.7 ->  $\geq 0.7$  go up first.

Definition at line 262 of file CbcSimpleInteger.hpp.

**7.89.3.19** `void CbcSimpleInteger::setBreakEven ( double value ) [inline]`

Set breakeven e.g 0.7 ->  $\geq 0.7$  go up first.

Definition at line 266 of file CbcSimpleInteger.hpp.

## 7.89.4 Member Data Documentation

**7.89.4.1** `double CbcSimpleInteger::originalLower_ [protected]`

data

Original lower bound

Definition at line 275 of file CbcSimpleInteger.hpp.

**7.89.4.2** `double CbcSimpleInteger::originalUpper_ [protected]`

Original upper bound.

Definition at line 277 of file CbcSimpleInteger.hpp.

**7.89.4.3** `double CbcSimpleInteger::breakEven_ [protected]`

Breakeven i.e.  $\geq$  this preferred is up.

Definition at line 279 of file CbcSimpleInteger.hpp.

**7.89.4.4** `int CbcSimpleInteger::columnNumber_ [protected]`

Column number in model.

Definition at line 281 of file CbcSimpleInteger.hpp.

**7.89.4.5** `int CbcSimpleInteger::preferredWay_ [protected]`

If -1 down always chosen first, +1 up always, 0 normal.

Definition at line 283 of file CbcSimpleInteger.hpp.

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

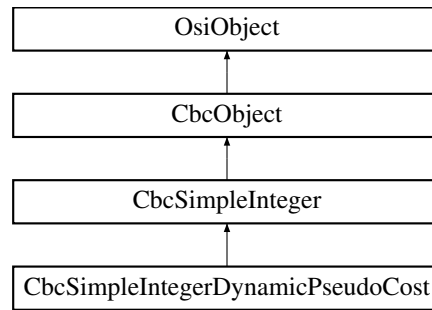
- [/home/ted/COIN/trunk/Cbc/src/CbcSimpleInteger.hpp](#)

## 7.90 CbcSimpleIntegerDynamicPseudoCost Class Reference

Define a single integer class but with dynamic pseudo costs.

```
#include <CbcSimpleIntegerDynamicPseudoCost.hpp>
```

Inheritance diagram for CbcSimpleIntegerDynamicPseudoCost:



#### Public Member Functions

- [CbcSimpleIntegerDynamicPseudoCost](#) ()
- [CbcSimpleIntegerDynamicPseudoCost](#) ([CbcModel](#) \*model, int iColumn, double [breakEven](#)=0.5)
- [CbcSimpleIntegerDynamicPseudoCost](#) ([CbcModel](#) \*model, int iColumn, double [downDynamicPseudoCost](#), double [upDynamicPseudoCost](#))
- [CbcSimpleIntegerDynamicPseudoCost](#) ([CbcModel](#) \*model, int dummy, int iColumn, double [downDynamicPseudoCost](#), double [upDynamicPseudoCost](#))
- [CbcSimpleIntegerDynamicPseudoCost](#) (const [CbcSimpleIntegerDynamicPseudoCost](#) &)
- virtual [CbcObject](#) \* [clone](#) () const  
*Clone.*
- [CbcSimpleIntegerDynamicPseudoCost](#) & [operator=](#) (const [CbcSimpleIntegerDynamicPseudoCost](#) &rhs)
- virtual [~CbcSimpleIntegerDynamicPseudoCost](#) ()
- virtual double [infeasibility](#) (const [OsiBranchingInformation](#) \*info, int &[preferredWay](#)) const  
*Infeasibility - large is 0.5.*
- virtual [CbcBranchingObject](#) \* [createCbcBranch](#) ([OsiSolverInterface](#) \*solver, const [OsiBranchingInformation](#) \*info, int way)  
*Creates a branching object.*
- virtual [CbcObjectUpdateData](#) [createUpdateInformation](#) (const [OsiSolverInterface](#) \*solver, const [CbcNode](#) \*node, const [CbcBranchingObject](#) \*branchingObject)  
*Fills in a created branching object.*
- virtual void [updateInformation](#) (const [CbcObjectUpdateData](#) &data)  
*Update object by CbcObjectUpdateData.*
- void [copySome](#) (const [CbcSimpleIntegerDynamicPseudoCost](#) \*otherObject)  
*Copy some information i.e. just variable stuff.*
- virtual void [updateBefore](#) (const [OsiObject](#) \*rhs)  
*Updates stuff like pseudocosts before threads.*
- virtual void [updateAfter](#) (const [OsiObject](#) \*rhs, const [OsiObject](#) \*baseObject)  
*Updates stuff like pseudocosts after threads finished.*
- void [updateAfterMini](#) (int numberDown, int numberDownInfeasible, double sumDown, int numberUp, int numberUpInfeasible, double sumUp)  
*Updates stuff like pseudocosts after mini branch and bound.*
- virtual [OsiSolverBranch](#) \* [solverBranch](#) () const  
*Create an OsiSolverBranch object.*
- double [downDynamicPseudoCost](#) () const  
*Down pseudo cost.*

- void [setDownDynamicPseudoCost](#) (double value)  
*Set down pseudo cost.*
- void [updateDownDynamicPseudoCost](#) (double value)  
*Modify down pseudo cost in a slightly different way.*
- double [upDynamicPseudoCost](#) () const  
*Up pseudo cost.*
- void [setUpDynamicPseudoCost](#) (double value)  
*Set up pseudo cost.*
- void [updateUpDynamicPseudoCost](#) (double value)  
*Modify up pseudo cost in a slightly different way.*
- double [downShadowPrice](#) () const  
*Down pseudo shadow price cost.*
- void [setDownShadowPrice](#) (double value)  
*Set down pseudo shadow price cost.*
- double [upShadowPrice](#) () const  
*Up pseudo shadow price cost.*
- void [setUpShadowPrice](#) (double value)  
*Set up pseudo shadow price cost.*
- double [upDownSeparator](#) () const  
*Up down separator.*
- void [setUpDownSeparator](#) (double value)  
*Set up down separator.*
- double [sumDownCost](#) () const  
*Down sum cost.*
- void [setSumDownCost](#) (double value)  
*Set down sum cost.*
- void [addToSumDownCost](#) (double value)  
*Add to down sum cost and set last and square.*
- double [sumUpCost](#) () const  
*Up sum cost.*
- void [setSumUpCost](#) (double value)  
*Set up sum cost.*
- void [addToSumUpCost](#) (double value)  
*Add to up sum cost and set last and square.*
- double [sumDownChange](#) () const  
*Down sum change.*
- void [setSumDownChange](#) (double value)  
*Set down sum change.*
- void [addToSumDownChange](#) (double value)  
*Add to down sum change.*
- double [sumUpChange](#) () const  
*Up sum change.*
- void [setSumUpChange](#) (double value)  
*Set up sum change.*
- void [addToSumUpChange](#) (double value)  
*Add to up sum change and set last and square.*
- double [sumDownDecrease](#) () const

- Sum down decrease number infeasibilities from strong or actual.*

  - void [setSumDownDecrease](#) (double value)

*Set sum down decrease number infeasibilities from strong or actual.*

  - void [addToSumDownDecrease](#) (double value)

*Add to sum down decrease number infeasibilities from strong or actual.*

  - double [sumUpDecrease](#) () const

*Sum up decrease number infeasibilities from strong or actual.*

  - void [setSumUpDecrease](#) (double value)

*Set sum up decrease number infeasibilities from strong or actual.*

  - void [addToSumUpDecrease](#) (double value)

*Add to sum up decrease number infeasibilities from strong or actual.*

  - int [numberTimesDown](#) () const

*Down number times.*

  - void [setNumberTimesDown](#) (int value)

*Set down number times.*

  - void [incrementNumberTimesDown](#) ()

*Increment down number times.*

  - int [numberTimesUp](#) () const

*Up number times.*

  - void [setNumberTimesUp](#) (int value)

*Set up number times.*

  - void [incrementNumberTimesUp](#) ()

*Increment up number times.*

  - int [numberTimesDownInfeasible](#) () const

*Down number times infeasible.*

  - void [setNumberTimesDownInfeasible](#) (int value)

*Set down number times infeasible.*

  - void [incrementNumberTimesDownInfeasible](#) ()

*Increment down number times infeasible.*

  - int [numberTimesUpInfeasible](#) () const

*Up number times infeasible.*

  - void [setNumberTimesUpInfeasible](#) (int value)

*Set up number times infeasible.*

  - void [incrementNumberTimesUpInfeasible](#) ()

*Increment up number times infeasible.*

  - int [numberBeforeTrust](#) () const

*Number of times before trusted.*

  - void [setNumberBeforeTrust](#) (int value)

*Set number of times before trusted.*

  - void [incrementNumberBeforeTrust](#) ()

*Increment number of times before trusted.*

  - virtual double [upEstimate](#) () const

*Return "up" estimate.*

  - virtual double [downEstimate](#) () const

*Return "down" estimate (default 1.0e-5)*

  - int [method](#) () const

*method - see below for details*

- void `setMethod` (int value)  
*Set method.*
- void `setDownInformation` (double changeObjectiveDown, int changeInfeasibilityDown)  
*Pass in information on a down branch.*
- void `setUpInformation` (double changeObjectiveUp, int changeInfeasibilityUp)  
*Pass in information on a up branch.*
- void `setProbingInformation` (int fixedDown, int fixedUp)  
*Pass in probing information.*
- void `print` (int type=0, double value=0.0) const  
*Print - 0 -summary, 1 just before strong.*
- bool `same` (const `CbcSimpleIntegerDynamicPseudoCost` \*obj) const  
*Same - returns true if contents match(ish)*

#### Protected Attributes

- double `downDynamicPseudoCost_`  
*data*
- double `upDynamicPseudoCost_`  
*Up pseudo cost.*
- double `upDownSeparator_`  
*Up/down separator If >0.0 then do first branch up if value-floor(value) >= this value.*
- double `sumDownCost_`  
*Sum down cost from strong or actual.*
- double `sumUpCost_`  
*Sum up cost from strong or actual.*
- double `sumDownChange_`  
*Sum of all changes to x when going down.*
- double `sumUpChange_`  
*Sum of all changes to x when going up.*
- double `downShadowPrice_`  
*Current pseudo-shadow price estimate down.*
- double `upShadowPrice_`  
*Current pseudo-shadow price estimate up.*
- double `sumDownDecrease_`  
*Sum down decrease number infeasibilities from strong or actual.*
- double `sumUpDecrease_`  
*Sum up decrease number infeasibilities from strong or actual.*
- double `lastDownCost_`  
*Last down cost from strong (i.e. as computed by last strong)*
- double `lastUpCost_`  
*Last up cost from strong (i.e. as computed by last strong)*
- int `lastDownDecrease_`  
*Last down decrease number infeasibilities from strong (i.e. as computed by last strong)*
- int `lastUpDecrease_`  
*Last up decrease number infeasibilities from strong (i.e. as computed by last strong)*
- int `numberOfTimesDown_`  
*Number of times we have gone down.*

- int [numberTimesUp\\_](#)  
*Number of times we have gone up.*
- int [numberTimesDownInfeasible\\_](#)  
*Number of times we have been infeasible going down.*
- int [numberTimesUpInfeasible\\_](#)  
*Number of times we have been infeasible going up.*
- int [numberBeforeTrust\\_](#)  
*Number of branches before we trust.*
- int [numberTimesDownLocalFixed\\_](#)  
*Number of local probing fixings going down.*
- int [numberTimesUpLocalFixed\\_](#)  
*Number of local probing fixings going up.*
- double [numberTimesDownTotalFixed\\_](#)  
*Number of total probing fixings going down.*
- double [numberTimesUpTotalFixed\\_](#)  
*Number of total probing fixings going up.*
- int [numberTimesProbingTotal\\_](#)  
*Number of times probing done.*
- int [method\\_](#)  
*Number of times infeasible when tested.*

#### 7.90.1 Detailed Description

Define a single integer class but with dynamic pseudo costs.

Based on work by Achterberg, Koch and Martin.

It is wild overkill but to keep design all twiddly things are in each. This could be used for fine tuning.

Definition at line 35 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

#### 7.90.2 Constructor & Destructor Documentation

7.90.2.1 `CbcSimpleIntegerDynamicPseudoCost::CbcSimpleIntegerDynamicPseudoCost ( )`

7.90.2.2 `CbcSimpleIntegerDynamicPseudoCost::CbcSimpleIntegerDynamicPseudoCost ( CbcModel * model, int iColumn, double breakEven = 0.5 )`

7.90.2.3 `CbcSimpleIntegerDynamicPseudoCost::CbcSimpleIntegerDynamicPseudoCost ( CbcModel * model, int iColumn, double downDynamicPseudoCost, double upDynamicPseudoCost )`

7.90.2.4 `CbcSimpleIntegerDynamicPseudoCost::CbcSimpleIntegerDynamicPseudoCost ( CbcModel * model, int dummy, int iColumn, double downDynamicPseudoCost, double upDynamicPseudoCost )`

7.90.2.5 `CbcSimpleIntegerDynamicPseudoCost::CbcSimpleIntegerDynamicPseudoCost ( const CbcSimpleIntegerDynamicPseudoCost & )`

7.90.2.6 `virtual CbcSimpleIntegerDynamicPseudoCost::~CbcSimpleIntegerDynamicPseudoCost ( ) [virtual]`

#### 7.90.3 Member Function Documentation

7.90.3.1 `virtual CbcObject* CbcSimpleIntegerDynamicPseudoCost::clone ( ) const` [virtual]

Clone.

Reimplemented from [CbcSimpleInteger](#).

7.90.3.2 `CbcSimpleIntegerDynamicPseudoCost& CbcSimpleIntegerDynamicPseudoCost::operator= ( const CbcSimpleIntegerDynamicPseudoCost & rhs )`

7.90.3.3 `virtual double CbcSimpleIntegerDynamicPseudoCost::infeasibility ( const OsiBranchingInformation * info, int & preferredWay ) const` [virtual]

Infeasibility - large is 0.5.

Reimplemented from [CbcSimpleInteger](#).

7.90.3.4 `virtual CbcBranchingObject* CbcSimpleIntegerDynamicPseudoCost::createCbcBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way )` [virtual]

Creates a branching object.

Reimplemented from [CbcSimpleInteger](#).

7.90.3.5 `virtual CbcObjectUpdateData CbcSimpleIntegerDynamicPseudoCost::createUpdateInformation ( const OsiSolverInterface * solver, const CbcNode * node, const CbcBranchingObject * branchingObject )` [virtual]

Fills in a created branching object.

Pass in information on branch just done and create [CbcObjectUpdateData](#) instance. If object does not need data then backward pointer will be NULL. Assumes can get information from solver

Reimplemented from [CbcObject](#).

7.90.3.6 `virtual void CbcSimpleIntegerDynamicPseudoCost::updateInformation ( const CbcObjectUpdateData & data )` [virtual]

Update object by [CbcObjectUpdateData](#).

Reimplemented from [CbcObject](#).

7.90.3.7 `void CbcSimpleIntegerDynamicPseudoCost::copySome ( const CbcSimpleIntegerDynamicPseudoCost * otherObject )`

Copy some information i.e. just variable stuff.

7.90.3.8 `virtual void CbcSimpleIntegerDynamicPseudoCost::updateBefore ( const OsiObject * rhs )` [virtual]

Updates stuff like pseudocosts before threads.

7.90.3.9 `virtual void CbcSimpleIntegerDynamicPseudoCost::updateAfter ( const OsiObject * rhs, const OsiObject * baseObject )` [virtual]

Updates stuff like pseudocosts after threads finished.

7.90.3.10 `void CbcSimpleIntegerDynamicPseudoCost::updateAfterMini ( int numberDown, int numberDownInfeasible, double sumDown, int numberUp, int numberUpInfeasible, double sumUp )`

Updates stuff like pseudocosts after mini branch and bound.



7.90.3.11 `virtual OsiSolverBranch* CbcSimpleIntegerDynamicPseudoCost::solverBranch ( ) const` `[virtual]`

Create an OsiSolverBranch object.

This returns NULL if branch not represented by bound changes

Reimplemented from [CbcObject](#).

7.90.3.12 `double CbcSimpleIntegerDynamicPseudoCost::downDynamicPseudoCost ( ) const` `[inline]`

Down pseudo cost.

Definition at line 103 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.13 `void CbcSimpleIntegerDynamicPseudoCost::setDownDynamicPseudoCost ( double value )`

Set down pseudo cost.

7.90.3.14 `void CbcSimpleIntegerDynamicPseudoCost::updateDownDynamicPseudoCost ( double value )`

Modify down pseudo cost in a slightly different way.

7.90.3.15 `double CbcSimpleIntegerDynamicPseudoCost::upDynamicPseudoCost ( ) const` `[inline]`

Up pseudo cost.

Definition at line 112 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.16 `void CbcSimpleIntegerDynamicPseudoCost::setUpDynamicPseudoCost ( double value )`

Set up pseudo cost.

7.90.3.17 `void CbcSimpleIntegerDynamicPseudoCost::updateUpDynamicPseudoCost ( double value )`

Modify up pseudo cost in a slightly different way.

7.90.3.18 `double CbcSimpleIntegerDynamicPseudoCost::downShadowPrice ( ) const` `[inline]`

Down pseudo shadow price cost.

Definition at line 121 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.19 `void CbcSimpleIntegerDynamicPseudoCost::setDownShadowPrice ( double value )` `[inline]`

Set down pseudo shadow price cost.

Definition at line 125 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.20 `double CbcSimpleIntegerDynamicPseudoCost::upShadowPrice ( ) const` `[inline]`

Up pseudo shadow price cost.

Definition at line 129 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.21 `void CbcSimpleIntegerDynamicPseudoCost::setUpShadowPrice ( double value )` `[inline]`

Set up pseudo shadow price cost.

Definition at line 133 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.22** `double CbcSimpleIntegerDynamicPseudoCost::upDownSeparator ( ) const` `[inline]`

Up down separator.

Definition at line 138 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.23** `void CbcSimpleIntegerDynamicPseudoCost::setUpDownSeparator ( double value )` `[inline]`

Set up down separator.

Definition at line 142 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.24** `double CbcSimpleIntegerDynamicPseudoCost::sumDownCost ( ) const` `[inline]`

Down sum cost.

Definition at line 147 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.25** `void CbcSimpleIntegerDynamicPseudoCost::setSumDownCost ( double value )` `[inline]`

Set down sum cost.

Definition at line 151 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.26** `void CbcSimpleIntegerDynamicPseudoCost::addToSumDownCost ( double value )` `[inline]`

Add to down sum cost and set last and square.

Definition at line 155 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.27** `double CbcSimpleIntegerDynamicPseudoCost::sumUpCost ( ) const` `[inline]`

Up sum cost.

Definition at line 161 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.28** `void CbcSimpleIntegerDynamicPseudoCost::setSumUpCost ( double value )` `[inline]`

Set up sum cost.

Definition at line 165 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.29** `void CbcSimpleIntegerDynamicPseudoCost::addToSumUpCost ( double value )` `[inline]`

Add to up sum cost and set last and square.

Definition at line 169 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.30** `double CbcSimpleIntegerDynamicPseudoCost::sumDownChange ( ) const` `[inline]`

Down sum change.

Definition at line 175 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.3.31** `void CbcSimpleIntegerDynamicPseudoCost::setSumDownChange ( double value )` `[inline]`

Set down sum change.

Definition at line 179 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.32 `void CbcSimpleIntegerDynamicPseudoCost::addToSumDownChange ( double value ) [inline]`

Add to down sum change.

Definition at line 183 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.33 `double CbcSimpleIntegerDynamicPseudoCost::sumUpChange ( ) const [inline]`

Up sum change.

Definition at line 188 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.34 `void CbcSimpleIntegerDynamicPseudoCost::setSumUpChange ( double value ) [inline]`

Set up sum change.

Definition at line 192 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.35 `void CbcSimpleIntegerDynamicPseudoCost::addToSumUpChange ( double value ) [inline]`

Add to up sum change and set last and square.

Definition at line 196 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.36 `double CbcSimpleIntegerDynamicPseudoCost::sumDownDecrease ( ) const [inline]`

Sum down decrease number infeasibilities from strong or actual.

Definition at line 201 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.37 `void CbcSimpleIntegerDynamicPseudoCost::setSumDownDecrease ( double value ) [inline]`

Set sum down decrease number infeasibilities from strong or actual.

Definition at line 205 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.38 `void CbcSimpleIntegerDynamicPseudoCost::addToSumDownDecrease ( double value ) [inline]`

Add to sum down decrease number infeasibilities from strong or actual.

Definition at line 209 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.39 `double CbcSimpleIntegerDynamicPseudoCost::sumUpDecrease ( ) const [inline]`

Sum up decrease number infeasibilities from strong or actual.

Definition at line 214 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.40 `void CbcSimpleIntegerDynamicPseudoCost::setSumUpDecrease ( double value ) [inline]`

Set sum up decrease number infeasibilities from strong or actual.

Definition at line 218 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.41 `void CbcSimpleIntegerDynamicPseudoCost::addToSumUpDecrease ( double value ) [inline]`

Add to sum up decrease number infeasibilities from strong or actual.

Definition at line 222 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.42 `int CbcSimpleIntegerDynamicPseudoCost::numberTimesDown ( ) const [inline]`

Down number times.

Definition at line 227 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.43 `void CbcSimpleIntegerDynamicPseudoCost::setNumberTimesDown ( int value ) [inline]`

Set down number times.

Definition at line 231 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.44 `void CbcSimpleIntegerDynamicPseudoCost::incrementNumberTimesDown ( ) [inline]`

Increment down number times.

Definition at line 235 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.45 `int CbcSimpleIntegerDynamicPseudoCost::numberTimesUp ( ) const [inline]`

Up number times.

Definition at line 240 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.46 `void CbcSimpleIntegerDynamicPseudoCost::setNumberTimesUp ( int value ) [inline]`

Set up number times.

Definition at line 244 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.47 `void CbcSimpleIntegerDynamicPseudoCost::incrementNumberTimesUp ( ) [inline]`

Increment up number times.

Definition at line 248 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.48 `int CbcSimpleIntegerDynamicPseudoCost::numberTimesDownInfeasible ( ) const [inline]`

Down number times infeasible.

Definition at line 253 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.49 `void CbcSimpleIntegerDynamicPseudoCost::setNumberTimesDownInfeasible ( int value ) [inline]`

Set down number times infeasible.

Definition at line 257 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.50 `void CbcSimpleIntegerDynamicPseudoCost::incrementNumberTimesDownInfeasible ( ) [inline]`

Increment down number times infeasible.

Definition at line 261 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.51 `int CbcSimpleIntegerDynamicPseudoCost::numberTimesUpInfeasible ( ) const [inline]`

Up number times infeasible.

Definition at line 266 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

7.90.3.52 `void CbcSimpleIntegerDynamicPseudoCost::setNumberTimesUpInfeasible ( int value ) [inline]`

Set up number times infeasible.

Definition at line 270 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.53 `void CbcSimpleIntegerDynamicPseudoCost::incrementNumberTimesUpInfeasible ( ) [inline]`

Increment up number times infeasible.

Definition at line 274 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.54 `int CbcSimpleIntegerDynamicPseudoCost::numberBeforeTrust ( ) const [inline]`

Number of times before trusted.

Definition at line 279 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.55 `void CbcSimpleIntegerDynamicPseudoCost::setNumberBeforeTrust ( int value ) [inline]`

Set number of times before trusted.

Definition at line 283 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.56 `void CbcSimpleIntegerDynamicPseudoCost::incrementNumberBeforeTrust ( ) [inline]`

Increment number of times before trusted.

Definition at line 287 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.57 `virtual double CbcSimpleIntegerDynamicPseudoCost::upEstimate ( ) const [virtual]`

Return "up" estimate.

7.90.3.58 `virtual double CbcSimpleIntegerDynamicPseudoCost::downEstimate ( ) const [virtual]`

Return "down" estimate (default 1.0e-5)

7.90.3.59 `int CbcSimpleIntegerDynamicPseudoCost::method ( ) const [inline]`

method - see below for details

Definition at line 297 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.60 `void CbcSimpleIntegerDynamicPseudoCost::setMethod ( int value ) [inline]`

Set method.

Definition at line 301 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.3.61 `void CbcSimpleIntegerDynamicPseudoCost::setDownInformation ( double changeObjectiveDown, int changeInfeasibilityDown )`

Pass in information on a down branch.

7.90.3.62 `void CbcSimpleIntegerDynamicPseudoCost::setUpInformation ( double changeObjectiveUp, int changeInfeasibilityUp )`

Pass in information on a up branch.

7.90.3.63 `void CbcSimpleIntegerDynamicPseudoCost::setProbingInformation ( int fixedDown, int fixedUp )`

Pass in probing information.

7.90.3.64 `void CbcSimpleIntegerDynamicPseudoCost::print ( int type = 0, double value = 0.0 ) const`

Print - 0 -summary, 1 just before strong.

7.90.3.65 `bool CbcSimpleIntegerDynamicPseudoCost::same ( const CbcSimpleIntegerDynamicPseudoCost * obj ) const`

Same - returns true if contents match(ish)

#### 7.90.4 Member Data Documentation

7.90.4.1 `double CbcSimpleIntegerDynamicPseudoCost::downDynamicPseudoCost_ [protected]`

data

Down pseudo cost

Definition at line 320 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.4.2 `double CbcSimpleIntegerDynamicPseudoCost::upDynamicPseudoCost_ [protected]`

Up pseudo cost.

Definition at line 322 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.4.3 `double CbcSimpleIntegerDynamicPseudoCost::upDownSeparator_ [protected]`

Up/down separator If >0.0 then do first branch up if value-floor(value) >= this value.

Definition at line 327 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.4.4 `double CbcSimpleIntegerDynamicPseudoCost::sumDownCost_ [protected]`

Sum down cost from strong or actual.

Definition at line 329 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.4.5 `double CbcSimpleIntegerDynamicPseudoCost::sumUpCost_ [protected]`

Sum up cost from strong or actual.

Definition at line 331 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.4.6 `double CbcSimpleIntegerDynamicPseudoCost::sumDownChange_ [protected]`

Sum of all changes to x when going down.

Definition at line 333 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

7.90.4.7 `double CbcSimpleIntegerDynamicPseudoCost::sumUpChange_ [protected]`

Sum of all changes to x when going up.

Definition at line 335 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.8** `double CbcSimpleIntegerDynamicPseudoCost::downShadowPrice_` [mutable], [protected]

Current pseudo-shadow price estimate down.

Definition at line 337 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.9** `double CbcSimpleIntegerDynamicPseudoCost::upShadowPrice_` [mutable], [protected]

Current pseudo-shadow price estimate up.

Definition at line 339 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.10** `double CbcSimpleIntegerDynamicPseudoCost::sumDownDecrease_` [protected]

Sum down decrease number infeasibilities from strong or actual.

Definition at line 341 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.11** `double CbcSimpleIntegerDynamicPseudoCost::sumUpDecrease_` [protected]

Sum up decrease number infeasibilities from strong or actual.

Definition at line 343 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.12** `double CbcSimpleIntegerDynamicPseudoCost::lastDownCost_` [protected]

Last down cost from strong (i.e. as computed by last strong)

Definition at line 345 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.13** `double CbcSimpleIntegerDynamicPseudoCost::lastUpCost_` [protected]

Last up cost from strong (i.e. as computed by last strong)

Definition at line 347 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.14** `int CbcSimpleIntegerDynamicPseudoCost::lastDownDecrease_` [mutable], [protected]

Last down decrease number infeasibilities from strong (i.e. as computed by last strong)

Definition at line 349 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.15** `int CbcSimpleIntegerDynamicPseudoCost::lastUpDecrease_` [mutable], [protected]

Last up decrease number infeasibilities from strong (i.e. as computed by last strong)

Definition at line 351 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.16** `int CbcSimpleIntegerDynamicPseudoCost::numberOfTimesDown_` [protected]

Number of times we have gone down.

Definition at line 353 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.17** `int CbcSimpleIntegerDynamicPseudoCost::numberOfTimesUp_` [protected]

Number of times we have gone up.

Definition at line 355 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

**7.90.4.18** `int CbcSimpleIntegerDynamicPseudoCost::numberOfTimesDownInfeasible_` `[protected]`

Number of times we have been infeasible going down.

Definition at line 357 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.90.4.19** `int CbcSimpleIntegerDynamicPseudoCost::numberOfTimesUpInfeasible_` `[protected]`

Number of times we have been infeasible going up.

Definition at line 359 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.90.4.20** `int CbcSimpleIntegerDynamicPseudoCost::numberOfBranchesBeforeTrust_` `[protected]`

Number of branches before we trust.

Definition at line 361 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.90.4.21** `int CbcSimpleIntegerDynamicPseudoCost::numberOfLocalProbingFixingsDown_` `[protected]`

Number of local probing fixings going down.

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

**7.90.4.22** `int CbcSimpleIntegerDynamicPseudoCost::numberOfLocalProbingFixingsUp_` `[protected]`

Number of local probing fixings going up.

Definition at line 365 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.90.4.23** `double CbcSimpleIntegerDynamicPseudoCost::numberOfTotalProbingFixingsDown_` `[protected]`

Number of total probing fixings going down.

Definition at line 367 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.90.4.24** `double CbcSimpleIntegerDynamicPseudoCost::numberOfTotalProbingFixingsUp_` `[protected]`

Number of total probing fixings going up.

Definition at line 369 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.90.4.25** `int CbcSimpleIntegerDynamicPseudoCost::numberOfTimesProbingDone_` `[protected]`

Number of times probing done.

Definition at line 371 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

**7.90.4.26** `int CbcSimpleIntegerDynamicPseudoCost::method_` `[protected]`

Number of times infeasible when tested.

Method - 0 - pseudo costs 1 - probing

Definition at line 377 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerDynamicPseudoCost.hpp](#)

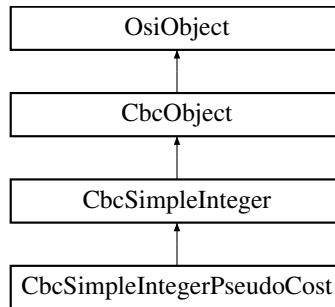


## 7.91 CbcSimpleIntegerPseudoCost Class Reference

Define a single integer class but with pseudo costs.

```
#include <CbcSimpleIntegerPseudoCost.hpp>
```

Inheritance diagram for CbcSimpleIntegerPseudoCost:



### Public Member Functions

- [CbcSimpleIntegerPseudoCost](#) ()
- [CbcSimpleIntegerPseudoCost](#) ([CbcModel](#) \**model*, int *iColumn*, double *breakEven*=0.5)
- [CbcSimpleIntegerPseudoCost](#) ([CbcModel](#) \**model*, int *iColumn*, double *downPseudoCost*, double *upPseudoCost*)
- [CbcSimpleIntegerPseudoCost](#) ([CbcModel](#) \**model*, int *dummy*, int *iColumn*, double *downPseudoCost*, double *upPseudoCost*)
- [CbcSimpleIntegerPseudoCost](#) (const [CbcSimpleIntegerPseudoCost](#) &)
- virtual [CbcObject](#) \* *clone* () const  
*Clone.*
- [CbcSimpleIntegerPseudoCost](#) & *operator=* (const [CbcSimpleIntegerPseudoCost](#) &*rhs*)
- virtual [~CbcSimpleIntegerPseudoCost](#) ()
- virtual double *infeasibility* (const [OsiBranchingInformation](#) \**info*, int &*preferredWay*) const  
*Infeasibility - large is 0.5.*
- virtual [CbcBranchingObject](#) \* *createCbcBranch* ([OsiSolverInterface](#) \**solver*, const [OsiBranchingInformation](#) \**info*, int *way*)  
*Creates a branching object.*
- double *downPseudoCost* () const  
*Down pseudo cost.*
- void *setDownPseudoCost* (double *value*)  
*Set down pseudo cost.*
- double *upPseudoCost* () const  
*Up pseudo cost.*
- void *setUpPseudoCost* (double *value*)  
*Set up pseudo cost.*
- double *upDownSeparator* () const  
*Up down separator.*
- void *setUpDownSeparator* (double *value*)  
*Set up down separator.*
- virtual double *upEstimate* () const  
*Return "up" estimate.*

- virtual double [downEstimate](#) () const  
*Return "down" estimate (default 1.0e-5)*
- int [method](#) () const  
*method - see below for details*
- void [setMethod](#) (int value)  
*Set method.*

#### Protected Attributes

- double [downPseudoCost\\_](#)  
*data*
- double [upPseudoCost\\_](#)  
*Up pseudo cost.*
- double [upDownSeparator\\_](#)  
*Up/down separator If >0.0 then do first branch up if value-floor(value) >= this value.*
- int [method\\_](#)  
*Method - 0 - normal - return min (up,down) 1 - if before any solution return CoinMax(up,down) 2 - if before branched solution return CoinMax(up,down) 3 - always return CoinMax(up,down)*

#### 7.91.1 Detailed Description

Define a single integer class but with pseudo costs.

Definition at line 14 of file `CbcSimpleIntegerPseudoCost.hpp`.

#### 7.91.2 Constructor & Destructor Documentation

- 7.91.2.1 `CbcSimpleIntegerPseudoCost::CbcSimpleIntegerPseudoCost ( )`
- 7.91.2.2 `CbcSimpleIntegerPseudoCost::CbcSimpleIntegerPseudoCost ( CbcModel * model, int iColumn, double breakEven = 0.5 )`
- 7.91.2.3 `CbcSimpleIntegerPseudoCost::CbcSimpleIntegerPseudoCost ( CbcModel * model, int iColumn, double downPseudoCost, double upPseudoCost )`
- 7.91.2.4 `CbcSimpleIntegerPseudoCost::CbcSimpleIntegerPseudoCost ( CbcModel * model, int dummy, int iColumn, double downPseudoCost, double upPseudoCost )`
- 7.91.2.5 `CbcSimpleIntegerPseudoCost::CbcSimpleIntegerPseudoCost ( const CbcSimpleIntegerPseudoCost & )`
- 7.91.2.6 `virtual CbcSimpleIntegerPseudoCost::~~CbcSimpleIntegerPseudoCost ( ) [virtual]`

#### 7.91.3 Member Function Documentation

- 7.91.3.1 `virtual CbcObject* CbcSimpleIntegerPseudoCost::clone ( ) const [virtual]`

Clone.

Reimplemented from [CbcSimpleInteger](#).

7.91.3.2 **CbcSimpleIntegerPseudoCost& CbcSimpleIntegerPseudoCost::operator= ( const CbcSimpleIntegerPseudoCost & rhs )**

7.91.3.3 **virtual double CbcSimpleIntegerPseudoCost::infeasibility ( const OsiBranchingInformation \* info, int & preferredWay ) const [virtual]**

Infeasibility - large is 0.5.

Reimplemented from [CbcSimpleInteger](#).

7.91.3.4 **virtual CbcBranchingObject\* CbcSimpleIntegerPseudoCost::createCbcBranch ( OsiSolverInterface \* solver, const OsiBranchingInformation \* info, int way ) [virtual]**

Creates a branching object.

Reimplemented from [CbcSimpleInteger](#).

7.91.3.5 **double CbcSimpleIntegerPseudoCost::downPseudoCost ( ) const [inline]**

Down pseudo cost.

Definition at line 51 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.3.6 **void CbcSimpleIntegerPseudoCost::setDownPseudoCost ( double value ) [inline]**

Set down pseudo cost.

Definition at line 55 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.3.7 **double CbcSimpleIntegerPseudoCost::upPseudoCost ( ) const [inline]**

Up pseudo cost.

Definition at line 60 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.3.8 **void CbcSimpleIntegerPseudoCost::setUpPseudoCost ( double value ) [inline]**

Set up pseudo cost.

Definition at line 64 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.3.9 **double CbcSimpleIntegerPseudoCost::upDownSeparator ( ) const [inline]**

Up down separator.

Definition at line 69 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.3.10 **void CbcSimpleIntegerPseudoCost::setUpDownSeparator ( double value ) [inline]**

Set up down separator.

Definition at line 73 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.3.11 **virtual double CbcSimpleIntegerPseudoCost::upEstimate ( ) const [virtual]**

Return "up" estimate.

7.91.3.12 **virtual double CbcSimpleIntegerPseudoCost::downEstimate ( ) const [virtual]**

Return "down" estimate (default 1.0e-5)

7.91.3.13 `int CbcSimpleIntegerPseudoCost::method ( ) const [inline]`

method - see below for details

Definition at line 83 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.3.14 `void CbcSimpleIntegerPseudoCost::setMethod ( int value ) [inline]`

Set method.

Definition at line 87 of file CbcSimpleIntegerPseudoCost.hpp.

#### 7.91.4 Member Data Documentation

7.91.4.1 `double CbcSimpleIntegerPseudoCost::downPseudoCost_ [protected]`

data

Down pseudo cost

Definition at line 95 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.4.2 `double CbcSimpleIntegerPseudoCost::upPseudoCost_ [protected]`

Up pseudo cost.

Definition at line 97 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.4.3 `double CbcSimpleIntegerPseudoCost::upDownSeparator_ [protected]`

Up/down separator If  $>0.0$  then do first branch up if  $\text{value-floor}(\text{value}) \geq$  this value.

Definition at line 102 of file CbcSimpleIntegerPseudoCost.hpp.

7.91.4.4 `int CbcSimpleIntegerPseudoCost::method_ [protected]`

Method - 0 - normal - return min (up,down) 1 - if before any solution return CoinMax(up,down) 2 - if before branched solution return CoinMax(up,down) 3 - always return CoinMax(up,down)

Definition at line 109 of file CbcSimpleIntegerPseudoCost.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerPseudoCost.hpp](#)

## 7.92 CbcSolver Class Reference

This allows the use of the standalone solver in a flexible manner.

```
#include <CbcSolver.hpp>
```

### Public Member Functions

#### Solve method

- `int solve (int argc, const char *argv[], int returnMode)`

*This takes a list of commands, does "stuff" and returns returnMode - 0 model and solver untouched - babModel updated 1 model updated - just with solution basis etc 2 model updated i.e.*

- `int solve (const char *input, int returnMode)`  
*This takes a list of commands, does "stuff" and returns returnMode - 0 model and solver untouched - babModel updated  
1 model updated - just with solution basis etc 2 model updated i.e.*

### Constructors and destructors etc

- `CbcSolver ()`  
*Default Constructor.*
- `CbcSolver (const OsiClpSolverInterface &)`  
*Constructor from solver.*
- `CbcSolver (const CbcModel &)`  
*Constructor from model.*
- `CbcSolver (const CbcSolver &rhs)`  
*Copy constructor .*
- `CbcSolver & operator= (const CbcSolver &rhs)`  
*Assignment operator.*
- `~CbcSolver ()`  
*Destructor.*
- `void fillParameters ()`  
*Fill with standard parameters.*
- `void fillValuesInSolver ()`  
*Set default values in solvers from parameters.*
- `void addUserFunction (CbcUser *function)`  
*Add user function.*
- `void setUserCallBack (CbcStopNow *function)`  
*Set user call back.*
- `void addCutGenerator (CglCutGenerator *generator)`  
*Add cut generator.*

### miscellaneous methods to line up with old

- `int * analyze (OsiClpSolverInterface *solverMod, int &numberChanged, double &increment, bool changeInt, CoinMessageHandler *generalMessageHandler)`
- `void updateModel (ClpSimplex *model2, int returnMode)`  
*1 - add heuristics to model 2 - do heuristics (and set cutoff and best solution) 3 - for miplib test so skip some (out model later)*

### useful stuff

- `int intValue (CbcOrClpParameterType type) const`  
*Get int value.*
- `void setIntValue (CbcOrClpParameterType type, int value)`  
*Set int value.*
- `double doubleValue (CbcOrClpParameterType type) const`  
*Get double value.*
- `void setDoubleValue (CbcOrClpParameterType type, double value)`  
*Set double value.*
- `CbcUser * userFunction (const char *name) const`  
*User function (NULL if no match)*
- `CbcModel * model ()`  
*Return original Cbc model.*
- `CbcModel * babModel ()`  
*Return updated Cbc model.*
- `int numberUserFunctions () const`

- *Number of userFunctions.*  
 • `CbcUser ** userFunctionArray () const`  
*User function array.*
- `OsiClpSolverInterface * originalSolver () const`  
*Copy of model on initial load (will contain output solutions)*
- `CoinModel * originalCoinModel () const`  
*Copy of model on initial load.*
- `void setOriginalSolver (OsiClpSolverInterface *originalSolver)`  
*Copy of model on initial load (will contain output solutions)*
- `void setOriginalCoinModel (CoinModel *originalCoinModel)`  
*Copy of model on initial load.*
- `int numberCutGenerators () const`  
*Number of cutgenerators.*
- `CglCutGenerator ** cutGeneratorArray () const`  
*Cut generator array.*
- `double startTime () const`  
*Start time.*
- `void setPrinting (bool onOff)`  
*Whether to print to std::cout.*
- `void setReadMode (int value)`  
*Where to start reading commands.*

### 7.92.1 Detailed Description

This allows the use of the standalone solver in a flexible manner.

It has an original `OsiClpSolverInterface` and `CbcModel` which it can use repeatedly, e.g., to get a heuristic solution and then start again.

So I [jjf] will need a primitive scripting language which can then call solve and manipulate solution value and solution arrays.

Also provides for user callback functions. Currently two ideas in gestation, `CbcUser` and `CbcStopNow`. The latter seems limited to deciding whether or not to stop. The former seems completely general, with a notion of importing and exporting, and a 'solve', which should be interpreted as 'do whatever this user function does'.

Parameter initialisation is at last centralised in `fillParameters()`.

Definition at line 56 of file `CbcSolver.hpp`.

### 7.92.2 Constructor & Destructor Documentation

#### 7.92.2.1 `CbcSolver::CbcSolver ( )`

Default Constructor.

#### 7.92.2.2 `CbcSolver::CbcSolver ( const OsiClpSolverInterface & )`

Constructor from solver.

#### 7.92.2.3 `CbcSolver::CbcSolver ( const CbcModel & )`

Constructor from model.

#### 7.92.2.4 `CbcSolver::CbcSolver ( const CbcSolver & rhs )`

Copy constructor .

## 7.92.2.5 CbcSolver::~~CbcSolver ( )

Destructor.

## 7.92.3 Member Function Documentation

## 7.92.3.1 int CbcSolver::solve ( int argc, const char \* argv[], int returnMode )

This takes a list of commands, does "stuff" and returns returnMode - 0 model and solver untouched - babModel updated  
1 model updated - just with solution basis etc 2 model updated i.e.

as babModel (babModel NULL) (only use without preprocessing)

## 7.92.3.2 int CbcSolver::solve ( const char \* input, int returnMode )

This takes a list of commands, does "stuff" and returns returnMode - 0 model and solver untouched - babModel updated  
1 model updated - just with solution basis etc 2 model updated i.e.

as babModel (babModel NULL) (only use without preprocessing)

## 7.92.3.3 CbcSolver&amp; CbcSolver::operator= ( const CbcSolver &amp; rhs )

Assignment operator.

## 7.92.3.4 void CbcSolver::fillParameters ( )

Fill with standard parameters.

## 7.92.3.5 void CbcSolver::fillValuesInSolver ( )

Set default values in solvers from parameters.

Misleading. The current code actually reads default values from the underlying solvers and installs them as default values for a subset of parameters in #parameters\_.

## 7.92.3.6 void CbcSolver::addUserFunction ( CbcUser \* function )

Add user function.

## 7.92.3.7 void CbcSolver::setUserCallBack ( CbcStopNow \* function )

Set user call back.

## 7.92.3.8 void CbcSolver::addCutGenerator ( CglCutGenerator \* generator )

Add cut generator.

## 7.92.3.9 int\* CbcSolver::analyze ( OsiClpSolverInterface \* solverMod, int &amp; numberChanged, double &amp; increment, bool changeInt, CoinMessageHandler \* generalMessageHandler )

## 7.92.3.10 void CbcSolver::updateModel ( ClpSimplex \* model2, int returnMode )

1 - add heuristics to model 2 - do heuristics (and set cutoff and best solution) 3 - for miplib test so skip some (out model later)

Updates model\_ from babModel\_ according to returnMode returnMode - 0 model and solver untouched - babModel updated  
1 model updated - just with solution basis etc 2 model updated i.e. as babModel (babModel NULL) (only use without preprocessing)

7.92.3.11 `int CbcSolver::intValue ( CbcOrClpParameterType type ) const`

Get int value.

7.92.3.12 `void CbcSolver::setIntValue ( CbcOrClpParameterType type, int value )`

Set int value.

7.92.3.13 `double CbcSolver::doubleValue ( CbcOrClpParameterType type ) const`

Get double value.

7.92.3.14 `void CbcSolver::setDoubleValue ( CbcOrClpParameterType type, double value )`

Set double value.

7.92.3.15 `CbcUser* CbcSolver::userFunction ( const char * name ) const`

User function (NULL if no match)

7.92.3.16 `CbcModel* CbcSolver::model ( ) [inline]`

Return original Cbc model.

Definition at line 144 of file CbcSolver.hpp.

7.92.3.17 `CbcModel* CbcSolver::babModel ( ) [inline]`

Return updated Cbc model.

Definition at line 148 of file CbcSolver.hpp.

7.92.3.18 `int CbcSolver::numberUserFunctions ( ) const [inline]`

Number of userFunctions.

Definition at line 152 of file CbcSolver.hpp.

7.92.3.19 `CbcUser** CbcSolver::userFunctionArray ( ) const [inline]`

User function array.

Definition at line 156 of file CbcSolver.hpp.

7.92.3.20 `OsiClpSolverInterface* CbcSolver::originalSolver ( ) const [inline]`

Copy of model on initial load (will contain output solutions)

Definition at line 160 of file CbcSolver.hpp.

7.92.3.21 `CoinModel* CbcSolver::originalCoinModel ( ) const [inline]`

Copy of model on initial load.

Definition at line 164 of file CbcSolver.hpp.

7.92.3.22 `void CbcSolver::setOriginalSolver ( OsiClpSolverInterface * originalSolver )`

Copy of model on initial load (will contain output solutions)



7.92.3.23 `void CbcSolver::setOriginalCoinModel ( CoinModel * originalCoinModel )`

Copy of model on initial load.

7.92.3.24 `int CbcSolver::numberCutGenerators ( ) const [inline]`

Number of cutgenerators.

Definition at line 172 of file CbcSolver.hpp.

7.92.3.25 `CglCutGenerator** CbcSolver::cutGeneratorArray ( ) const [inline]`

Cut generator array.

Definition at line 176 of file CbcSolver.hpp.

7.92.3.26 `double CbcSolver::startTime ( ) const [inline]`

Start time.

Definition at line 180 of file CbcSolver.hpp.

7.92.3.27 `void CbcSolver::setPrinting ( bool onOff ) [inline]`

Whether to print to std::cout.

Definition at line 184 of file CbcSolver.hpp.

7.92.3.28 `void CbcSolver::setReadMode ( int value ) [inline]`

Where to start reading commands.

Definition at line 188 of file CbcSolver.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcSolver.hpp](#)

## 7.93 CbcSolverUsefulData Struct Reference

Structure to hold useful arrays.

```
#include <CbcSolver.hpp>
```

### Public Attributes

- `int * priorities\_`
- `int * sosPriority\_`
- `int * branchDirection\_`
- `double * primalSolution\_`
- `double * pseudoDown\_`
- `double * pseudoUp\_`

### 7.93.1 Detailed Description

Structure to hold useful arrays.

Definition at line 240 of file CbcSolver.hpp.

### 7.93.2 Member Data Documentation

#### 7.93.2.1 `int* CbcSolverUsefulData::priorities_`

Definition at line 242 of file CbcSolver.hpp.

#### 7.93.2.2 `int* CbcSolverUsefulData::sosPriority_`

Definition at line 244 of file CbcSolver.hpp.

#### 7.93.2.3 `int* CbcSolverUsefulData::branchDirection_`

Definition at line 246 of file CbcSolver.hpp.

#### 7.93.2.4 `double* CbcSolverUsefulData::primalSolution_`

Definition at line 248 of file CbcSolver.hpp.

#### 7.93.2.5 `double* CbcSolverUsefulData::pseudoDown_`

Definition at line 250 of file CbcSolver.hpp.

#### 7.93.2.6 `double* CbcSolverUsefulData::pseudoUp_`

Definition at line 252 of file CbcSolver.hpp.

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

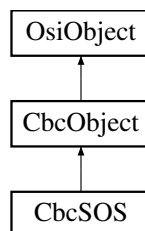
- </home/ted/COIN/trunk/Cbc/src/CbcSolver.hpp>

## 7.94 CbcSOS Class Reference

Branching object for Special Ordered Sets of type 1 and 2.

```
#include <CbcSOS.hpp>
```

Inheritance diagram for CbcSOS:



### Public Member Functions

- `CbcSOS ()`
- `CbcSOS (CbcModel *model, int numberMembers, const int *which, const double *weights, int identifier, int type=1)`  
*Constructor with SOS type and member information.*
- `CbcSOS (const CbcSOS &)`
- `virtual CbcObject * clone () const`

*Clone.*

- `CbcSOS & operator=` (const `CbcSOS` &rhs)
- virtual `~CbcSOS` ()
- virtual double `infeasibility` (const `OsiBranchingInformation` \*info, int &`preferredWay`) const

*Infeasibility - large is 0.5.*

- virtual void `feasibleRegion` ()

*This looks at solution and sets bounds to contain solution.*

- virtual `CbcBranchingObject` \* `createCbcBranch` (`OsiSolverInterface` \*solver, const `OsiBranchingInformation` \*info, int way)

*Creates a branching object.*

- virtual `CbcObjectUpdateData` `createUpdateInformation` (const `OsiSolverInterface` \*solver, const `CbcNode` \*node, const `CbcBranchingObject` \*branchingObject)

*Pass in information on branch just done and create `CbcObjectUpdateData` instance.*

- virtual void `updateInformation` (const `CbcObjectUpdateData` &data)

*Update object by `CbcObjectUpdateData`.*

- virtual `OsiSolverBranch` \* `solverBranch` () const

*Create an `OsiSolverBranch` object.*

- virtual void `redoSequenceEtc` (`CbcModel` \*model, int numberColumns, const int \*originalColumns)

*Redoes data when sequence numbers change.*

- `OsiSOS` \* `osiObject` (const `OsiSolverInterface` \*solver) const

*Construct an `OsiSOS` object.*

- int `numberMembers` () const

*Number of members.*

- const int \* `members` () const

*Members (indices in range 0 ... numberColumns-1)*

- int `sosType` () const

*SOS type.*

- int `numberTimesDown` () const

*Down number times.*

- int `numberTimesUp` () const

*Up number times.*

- const double \* `weights` () const

*Array of weights.*

- void `setNumberMembers` (int n)

*Set number of members.*

- int \* `mutableMembers` () const

*Members (indices in range 0 ... numberColumns-1)*

- double \* `mutableWeights` () const

*Array of weights.*

- virtual bool `canDoHeuristics` () const

*Return true if object can take part in normal heuristics.*

- void `setIntegerValued` (bool yesNo)

*Set whether set is integer valued or not.*

## Additional Inherited Members

### 7.94.1 Detailed Description

Branching object for Special Ordered Sets of type 1 and 2.

SOS1 are an ordered set of variables where at most one variable can be non-zero. SOS1 are commonly defined with binary variables (interpreted as selection between alternatives) but this is not necessary. An SOS1 with all binary variables is a special case of a clique (setting any one variable to 1 forces all others to 0).

In theory, the implementation makes no assumptions about integrality in Type 1 sets. In practice, there are places where the code seems to have been written with a binary SOS mindset. Current development of SOS branching objects is proceeding in OsiSOS.

SOS2 are an ordered set of variables in which at most two consecutive variables can be non-zero and must sum to 1 (interpreted as interpolation between two discrete values). By definition the variables are non-integer.

Definition at line 29 of file CbcSOS.hpp.

### 7.94.2 Constructor & Destructor Documentation

#### 7.94.2.1 CbcSOS::CbcSOS ( )

**7.94.2.2** CbcSOS::CbcSOS ( CbcModel \* *model*, int *numberMembers*, const int \* *which*, const double \* *weights*, int *identifier*, int *type* = 1 )

Constructor with SOS type and member information.

Type specifies SOS 1 or 2. Identifier is an arbitrary value.

Which should be an array of variable indices with numberMembers entries. Weights can be used to assign arbitrary weights to variables, in the order they are specified in which. If no weights are provided, a default array of 0, 1, 2, ... is generated.

#### 7.94.2.3 CbcSOS::CbcSOS ( const CbcSOS & )

#### 7.94.2.4 virtual CbcSOS::~~CbcSOS ( ) [virtual]

### 7.94.3 Member Function Documentation

#### 7.94.3.1 virtual CbcObject\* CbcSOS::clone ( ) const [virtual]

Clone.

Implements [CbcObject](#).

#### 7.94.3.2 CbcSOS& CbcSOS::operator= ( const CbcSOS & *rhs* )

#### 7.94.3.3 virtual double CbcSOS::infeasibility ( const OsiBranchingInformation \* *info*, int & *preferredWay* ) const [virtual]

Infeasibility - large is 0.5.

Reimplemented from [CbcObject](#).

#### 7.94.3.4 virtual void CbcSOS::feasibleRegion ( ) [virtual]

This looks at solution and sets bounds to contain solution.

Implements [CbcObject](#).

7.94.3.5 `virtual CbcBranchingObject* CbcSOS::createCbcBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way ) [virtual]`

Creates a branching object.

Reimplemented from [CbcObject](#).

7.94.3.6 `virtual CbcObjectUpdateData CbcSOS::createUpdateInformation ( const OsiSolverInterface * solver, const CbcNode * node, const CbcBranchingObject * branchingObject ) [virtual]`

Pass in information on branch just done and create [CbcObjectUpdateData](#) instance.

If object does not need data then backward pointer will be NULL. Assumes can get information from solver

Reimplemented from [CbcObject](#).

7.94.3.7 `virtual void CbcSOS::updateInformation ( const CbcObjectUpdateData & data ) [virtual]`

Update object by [CbcObjectUpdateData](#).

Reimplemented from [CbcObject](#).

7.94.3.8 `virtual OsiSolverBranch* CbcSOS::solverBranch ( ) const [virtual]`

Create an OsiSolverBranch object.

This returns NULL if branch not represented by bound changes

Reimplemented from [CbcObject](#).

7.94.3.9 `virtual void CbcSOS::redoSequenceEtc ( CbcModel * model, int numberColumns, const int * originalColumns ) [virtual]`

Redoes data when sequence numbers change.

Reimplemented from [CbcObject](#).

7.94.3.10 `OsiSOS* CbcSOS::osiObject ( const OsiSolverInterface * solver ) const`

Construct an OsiSOS object.

7.94.3.11 `int CbcSOS::numberMembers ( ) const [inline]`

Number of members.

Definition at line 95 of file CbcSOS.hpp.

7.94.3.12 `const int* CbcSOS::members ( ) const [inline]`

Members (indices in range 0 ... numberColumns-1)

Definition at line 100 of file CbcSOS.hpp.

7.94.3.13 `int CbcSOS::sosType ( ) const [inline]`

SOS type.

Definition at line 105 of file CbcSOS.hpp.

7.94.3.14 `int CbcSOS::numberTimesDown ( ) const [inline]`

Down number times.

Definition at line 109 of file CbcSOS.hpp.

**7.94.3.15** `int CbcSOS::numberOfTimesUp ( ) const [inline]`

Up number times.

Definition at line 113 of file CbcSOS.hpp.

**7.94.3.16** `const double* CbcSOS::weights ( ) const [inline]`

Array of weights.

Definition at line 118 of file CbcSOS.hpp.

**7.94.3.17** `void CbcSOS::setNumberMembers ( int n ) [inline]`

Set number of members.

Definition at line 123 of file CbcSOS.hpp.

**7.94.3.18** `int* CbcSOS::mutableMembers ( ) const [inline]`

Members (indices in range 0 ... numberColumns-1)

Definition at line 128 of file CbcSOS.hpp.

**7.94.3.19** `double* CbcSOS::mutableWeights ( ) const [inline]`

Array of weights.

Definition at line 133 of file CbcSOS.hpp.

**7.94.3.20** `virtual bool CbcSOS::canDoHeuristics ( ) const [inline],[virtual]`

Return true if object can take part in normal heuristics.

Definition at line 139 of file CbcSOS.hpp.

**7.94.3.21** `void CbcSOS::setIntegerValued ( bool yesNo ) [inline]`

Set whether set is integer valued or not.

Definition at line 143 of file CbcSOS.hpp.

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

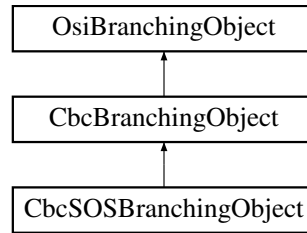
- [/home/ted/COIN/trunk/Cbc/src/CbcSOS.hpp](#)

## 7.95 CbcSOSBranchingObject Class Reference

Branching object for Special ordered sets.

```
#include <CbcSOS.hpp>
```

Inheritance diagram for CbcSOSBranchingObject:



### Public Member Functions

- [CbcSOSBranchingObject](#) ()
- [CbcSOSBranchingObject](#) ([CbcModel](#) \*model, const [CbcSOS](#) \*clique, int way, double separator)
- [CbcSOSBranchingObject](#) (const [CbcSOSBranchingObject](#) &)
- [CbcSOSBranchingObject](#) & operator= (const [CbcSOSBranchingObject](#) &rhs)
- virtual [CbcBranchingObject](#) \* clone () const  
*Clone.*
- virtual ~[CbcSOSBranchingObject](#) ()
- virtual double [branch](#) ()  
*Does next branch and updates state.*
- virtual void [fix](#) ([OsiSolverInterface](#) \*solver, double \*lower, double \*upper, int branchState) const  
*Update bounds in solver as in 'branch' and update given bounds.*
- virtual void [previousBranch](#) ()  
*Reset every information so that the branching object appears to point to the previous child.*
- virtual void [print](#) ()  
*Print something about branch - only if log level high.*
- virtual [CbcBranchObjType](#) type () const  
*Return the type (an integer identifier) of this.*
- virtual int [compareOriginalObject](#) (const [CbcBranchingObject](#) \*brObj) const  
*Compare the original object of this with the original object of brObj.*
- virtual [CbcRangeCompare](#) [compareBranchingObject](#) (const [CbcBranchingObject](#) \*brObj, const bool replaceIfOverlap=false)  
*Compare the this with brObj.*
- void [computeNonzeroRange](#) ()  
*Fill out the firstNonzero\_ and lastNonzero\_ data members.*

### Additional Inherited Members

#### 7.95.1 Detailed Description

Branching object for Special ordered sets.

Variable\_ is the set id number (redundant, as the object also holds a pointer to the set).

Definition at line 189 of file CbcSOS.hpp.

## 7.95.2 Constructor & Destructor Documentation

7.95.2.1 `CbcSOSBranchingObject::CbcSOSBranchingObject ( )`

7.95.2.2 `CbcSOSBranchingObject::CbcSOSBranchingObject ( CbcModel * model, const CbcSOS * clique, int way, double separator )`

7.95.2.3 `CbcSOSBranchingObject::CbcSOSBranchingObject ( const CbcSOSBranchingObject & )`

7.95.2.4 `virtual CbcSOSBranchingObject::~~CbcSOSBranchingObject ( )` [virtual]

## 7.95.3 Member Function Documentation

7.95.3.1 `CbcSOSBranchingObject& CbcSOSBranchingObject::operator= ( const CbcSOSBranchingObject & rhs )`

7.95.3.2 `virtual CbcBranchingObject* CbcSOSBranchingObject::clone ( ) const` [virtual]

Clone.

Implements [CbcBranchingObject](#).

7.95.3.3 `virtual double CbcSOSBranchingObject::branch ( )` [virtual]

Does next branch and updates state.

Implements [CbcBranchingObject](#).

7.95.3.4 `virtual void CbcSOSBranchingObject::fix ( OsiSolverInterface * solver, double * lower, double * upper, int branchState ) const` [virtual]

Update bounds in solver as in 'branch' and update given bounds.

branchState is -1 for 'down' +1 for 'up'

Reimplemented from [CbcBranchingObject](#).

7.95.3.5 `virtual void CbcSOSBranchingObject::previousBranch ( )` [inline],[virtual]

Reset every information so that the branching object appears to point to the previous child.

This method does not need to modify anything in any solver.

Reimplemented from [CbcBranchingObject](#).

Definition at line 225 of file CbcSOS.hpp.

7.95.3.6 `virtual void CbcSOSBranchingObject::print ( )` [virtual]

Print something about branch - only if log level high.

7.95.3.7 `virtual CbcBranchObjType CbcSOSBranchingObject::type ( ) const` [inline],[virtual]

Return the type (an integer identifier) of this.

Implements [CbcBranchingObject](#).

Definition at line 236 of file CbcSOS.hpp.



**7.95.3.8** `virtual int CbcSOSBranchingObject::compareOriginalObject ( const CbcBranchingObject * brObj ) const`  
`[virtual]`

Compare the original object of `this` with the original object of `brObj`.

Assumes that there is an ordering of the original objects. This method should be invoked only if `this` and `brObj` are of the same type. Return negative/0/positive depending on whether `this` is smaller/same/larger than the argument.

Reimplemented from [CbcBranchingObject](#).

**7.95.3.9** `virtual CbcRangeCompare CbcSOSBranchingObject::compareBranchingObject ( const CbcBranchingObject * brObj, const bool replaceIfOverlap = false )` `[virtual]`

Compare the `this` with `brObj`.

`this` and `brObj` must be of the same type and must have the same original object, but they may have different feasible regions. Return the appropriate `CbcRangeCompare` value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the current branching object with one whose feasible region is the overlap.

Implements [CbcBranchingObject](#).

**7.95.3.10** `void CbcSOSBranchingObject::computeNonzeroRange ( )`

Fill out the `firstNonzero_` and `lastNonzero_` data members.

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

- `/home/ted/COIN/trunk/Cbc/src/CbcSOS.hpp`

## 7.96 CbcStatistics Class Reference

For gathering statistics.

```
#include <CbcStatistics.hpp>
```

### Public Member Functions

- [CbcStatistics](#) ()
- [CbcStatistics](#) ([CbcNode](#) \*node, [CbcModel](#) \*model)
- [~CbcStatistics](#) ()
- [CbcStatistics](#) (const [CbcStatistics](#) &rhs)
- [CbcStatistics](#) & [operator=](#) (const [CbcStatistics](#) &rhs)
- void [endOfBranch](#) (int [numberIterations](#), double [objectiveValue](#))
- void [updateInfeasibility](#) (int [numberInfeasibilities](#))
- void [sayInfeasible](#) ()
- void [print](#) (const int \*sequenceLookup=NULL) const
- int [node](#) () const
- int [parentNode](#) () const
- int [depth](#) () const
- int [way](#) () const
- double [value](#) () const
- double [startingObjective](#) () const
- int [startingInfeasibility](#) () const
- double [endingObjective](#) () const
- int [endingInfeasibility](#) () const
- int [numberIterations](#) () const

## Protected Attributes

- double [value\\_](#)  
*Value.*
- double [startingObjective\\_](#)  
*Starting objective.*
- double [endingObjective\\_](#)  
*Ending objective.*
- int [id\\_](#)  
*id*
- int [parentId\\_](#)  
*parent id*
- int [way\\_](#)  
*way -1 or +1 is first branch -10 or +10 is second branch*
- int [sequence\\_](#)  
*sequence number branched on*
- int [depth\\_](#)  
*depth*
- int [startingInfeasibility\\_](#)  
*starting number of integer infeasibilities*
- int [endingInfeasibility\\_](#)  
*ending number of integer infeasibilities*
- int [numberIterations\\_](#)  
*number of iterations*

### 7.96.1 Detailed Description

For gathering statistics.

Definition at line 13 of file CbcStatistics.hpp.

### 7.96.2 Constructor & Destructor Documentation

7.96.2.1 `CbcStatistics::CbcStatistics ( )`

7.96.2.2 `CbcStatistics::CbcStatistics ( CbcNode * node, CbcModel * model )`

7.96.2.3 `CbcStatistics::~~CbcStatistics ( )`

7.96.2.4 `CbcStatistics::CbcStatistics ( const CbcStatistics & rhs )`

### 7.96.3 Member Function Documentation

7.96.3.1 `CbcStatistics& CbcStatistics::operator= ( const CbcStatistics & rhs )`

7.96.3.2 `void CbcStatistics::endOfBranch ( int numberIterations, double objectiveValue )`

7.96.3.3 `void CbcStatistics::updateInfeasibility ( int numberInfeasibilities )`

7.96.3.4 `void CbcStatistics::sayInfeasible ( )`

7.96.3.5 `void CbcStatistics::print ( const int * sequenceLookup = NULL ) const`

7.96.3.6 `int CbcStatistics::node ( ) const [inline]`

Definition at line 34 of file CbcStatistics.hpp.

7.96.3.7 `int CbcStatistics::parentNode ( ) const [inline]`

Definition at line 38 of file CbcStatistics.hpp.

7.96.3.8 `int CbcStatistics::depth ( ) const [inline]`

Definition at line 42 of file CbcStatistics.hpp.

7.96.3.9 `int CbcStatistics::way ( ) const [inline]`

Definition at line 46 of file CbcStatistics.hpp.

7.96.3.10 `double CbcStatistics::value ( ) const [inline]`

Definition at line 50 of file CbcStatistics.hpp.

7.96.3.11 `double CbcStatistics::startingObjective ( ) const [inline]`

Definition at line 54 of file CbcStatistics.hpp.

7.96.3.12 `int CbcStatistics::startingInfeasibility ( ) const [inline]`

Definition at line 58 of file CbcStatistics.hpp.

7.96.3.13 `double CbcStatistics::endingObjective ( ) const [inline]`

Definition at line 62 of file CbcStatistics.hpp.

7.96.3.14 `int CbcStatistics::endingInfeasibility ( ) const [inline]`

Definition at line 66 of file CbcStatistics.hpp.

7.96.3.15 `int CbcStatistics::numberIterations ( ) const [inline]`

Definition at line 70 of file CbcStatistics.hpp.

#### 7.96.4 Member Data Documentation

7.96.4.1 `double CbcStatistics::value_ [protected]`

Value.

Definition at line 77 of file CbcStatistics.hpp.

7.96.4.2 `double CbcStatistics::startingObjective_ [protected]`

Starting objective.

Definition at line 79 of file CbcStatistics.hpp.

#### 7.96.4.3 `double CbcStatistics::endingObjective_` [protected]

Ending objective.

Definition at line 81 of file CbcStatistics.hpp.

#### 7.96.4.4 `int CbcStatistics::id_` [protected]

id

Definition at line 83 of file CbcStatistics.hpp.

#### 7.96.4.5 `int CbcStatistics::parentId_` [protected]

parent id

Definition at line 85 of file CbcStatistics.hpp.

#### 7.96.4.6 `int CbcStatistics::way_` [protected]

way -1 or +1 is first branch -10 or +10 is second branch

Definition at line 87 of file CbcStatistics.hpp.

#### 7.96.4.7 `int CbcStatistics::sequence_` [protected]

sequence number branched on

Definition at line 89 of file CbcStatistics.hpp.

#### 7.96.4.8 `int CbcStatistics::depth_` [protected]

depth

Definition at line 91 of file CbcStatistics.hpp.

#### 7.96.4.9 `int CbcStatistics::startingInfeasibility_` [protected]

starting number of integer infeasibilities

Definition at line 93 of file CbcStatistics.hpp.

#### 7.96.4.10 `int CbcStatistics::endingInfeasibility_` [protected]

ending number of integer infeasibilities

Definition at line 95 of file CbcStatistics.hpp.

#### 7.96.4.11 `int CbcStatistics::numberIterations_` [protected]

number of iterations

Definition at line 97 of file CbcStatistics.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcStatistics.hpp](#)

## 7.97 CbcStopNow Class Reference

Support the use of a call back class to decide whether to stop.

```
#include <CbcSolver.hpp>
```

## Public Member Functions

### Decision methods

- virtual int [callBack](#) ([CbcModel](#) \*, int)  
*Import.*

### Constructors and destructors etc

- [CbcStopNow](#) ()  
*Default Constructor.*
- [CbcStopNow](#) (const [CbcStopNow](#) &rhs)  
*Copy constructor .*
- [CbcStopNow](#) & [operator=](#) (const [CbcStopNow](#) &rhs)  
*Assignment operator.*
- virtual [CbcStopNow](#) \* [clone](#) () const  
*Clone.*
- virtual [~CbcStopNow](#) ()  
*Destructor.*

#### 7.97.1 Detailed Description

Support the use of a call back class to decide whether to stop.

Definitely under construction.

Definition at line 351 of file CbcSolver.hpp.

#### 7.97.2 Constructor & Destructor Documentation

##### 7.97.2.1 CbcStopNow::CbcStopNow ( )

Default Constructor.

##### 7.97.2.2 CbcStopNow::CbcStopNow ( const CbcStopNow & rhs )

Copy constructor .

##### 7.97.2.3 virtual CbcStopNow::~~CbcStopNow ( ) [virtual]

Destructor.

#### 7.97.3 Member Function Documentation

##### 7.97.3.1 virtual int CbcStopNow::callBack ( CbcModel \*, int ) [inline],[virtual]

Import.

Values for whereFrom:

- 1 after initial solve by dualsimplex etc

- 2 after preprocessing
- 3 just before branchAndBound (so user can override)
- 4 just after branchAndBound (before postprocessing)
- 5 after postprocessing
- 6 after a user called heuristic phase

#### Returns

0 if good nonzero return code to stop

Definition at line 369 of file CbcSolver.hpp.

#### 7.97.3.2 CbcStopNow& CbcStopNow::operator= ( const CbcStopNow & rhs )

Assignment operator.

#### 7.97.3.3 virtual CbcStopNow\* CbcStopNow::clone ( ) const [virtual]

Clone.

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

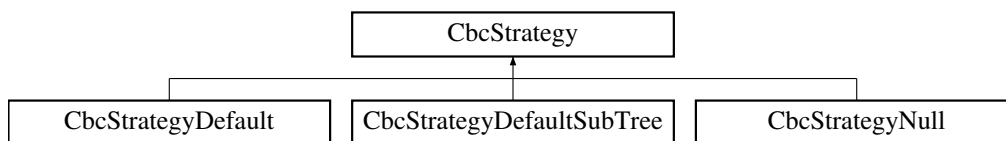
- </home/ted/COIN/trunk/Cbc/src/CbcSolver.hpp>

## 7.98 CbcStrategy Class Reference

Strategy base class.

```
#include <CbcStrategy.hpp>
```

Inheritance diagram for CbcStrategy:



#### Public Member Functions

- [CbcStrategy](#) ()
- virtual [~CbcStrategy](#) ()
- virtual [CbcStrategy](#) \* [clone](#) () const =0  
*Clone.*
- virtual void [setupCutGenerators](#) ([CbcModel](#) &model)=0  
*Setup cut generators.*
- virtual void [setupHeuristics](#) ([CbcModel](#) &model)=0  
*Setup heuristics.*
- virtual void [setupPrinting](#) ([CbcModel](#) &model, int modelLogLevel)=0

- *Do printing stuff.*
- virtual void `setupOther` (`CbcModel` &model)=0  
*Other stuff e.g. strong branching and preprocessing.*
- void `setNested` (int depth)  
*Set model depth (i.e. how nested)*
- int `getNested` () const  
*Get model depth (i.e. how nested)*
- void `setPreProcessState` (int state)  
*Say preProcessing done.*
- int `preProcessState` () const  
*See what sort of preprocessing was done.*
- `CglPreProcess` \* `process` () const  
*Pre-processing object.*
- void `deletePreProcess` ()  
*Delete pre-processing object to save memory.*
- virtual `CbcNodeInfo` \* `fullNodeInfo` (`CbcModel` \*model, int numberOfRowsAtContinuous) const  
*Return a new Full node information pointer (descendant of `CbcFullNodeInfo`)*
- virtual `CbcNodeInfo` \* `partialNodeInfo` (`CbcModel` \*model, `CbcNodeInfo` \*parent, `CbcNode` \*owner, int number-ChangedBounds, const int \*variables, const double \*boundChanges, const `CoinWarmStartDiff` \*basisDiff) const  
*Return a new Partial node information pointer (descendant of `CbcPartialNodeInfo`)*
- virtual void `generateCpp` (FILE \*)  
*Create C++ lines to get to current state.*
- virtual int `status` (`CbcModel` \*model, `CbcNodeInfo` \*parent, int whereFrom)  
*After a `CbcModel::resolve` this can return a status -1 no effect 0 treat as optimal 1 as 0 but do not do any more resolves (i.e.*

#### Protected Attributes

- int `depth_`  
*Model depth.*
- int `preProcessState_`  
*PreProcessing state - -1 infeasible 0 off 1 was done (so need post-processing)*
- `CglPreProcess` \* `process_`  
*If preprocessing then this is object.*

#### 7.98.1 Detailed Description

Strategy base class.

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

#### 7.98.2 Constructor & Destructor Documentation

##### 7.98.2.1 `CbcStrategy::CbcStrategy ( )`

##### 7.98.2.2 `virtual CbcStrategy::~~CbcStrategy ( )` `[virtual]`

#### 7.98.3 Member Function Documentation

**7.98.3.1** `virtual CbcStrategy* CbcStrategy::clone ( ) const` `[pure virtual]`

Clone.

Implemented in [CbcStrategyDefaultSubTree](#), [CbcStrategyDefault](#), and [CbcStrategyNull](#).

**7.98.3.2** `virtual void CbcStrategy::setupCutGenerators ( CbcModel & model )` `[pure virtual]`

Setup cut generators.

Implemented in [CbcStrategyDefaultSubTree](#), [CbcStrategyDefault](#), and [CbcStrategyNull](#).

**7.98.3.3** `virtual void CbcStrategy::setupHeuristics ( CbcModel & model )` `[pure virtual]`

Setup heuristics.

Implemented in [CbcStrategyDefaultSubTree](#), [CbcStrategyDefault](#), and [CbcStrategyNull](#).

**7.98.3.4** `virtual void CbcStrategy::setupPrinting ( CbcModel & model, int modelLogLevel )` `[pure virtual]`

Do printing stuff.

Implemented in [CbcStrategyDefaultSubTree](#), [CbcStrategyDefault](#), and [CbcStrategyNull](#).

**7.98.3.5** `virtual void CbcStrategy::setupOther ( CbcModel & model )` `[pure virtual]`

Other stuff e.g. strong branching and preprocessing.

Implemented in [CbcStrategyDefaultSubTree](#), [CbcStrategyDefault](#), and [CbcStrategyNull](#).

**7.98.3.6** `void CbcStrategy::setNested ( int depth )` `[inline]`

Set model depth (i.e. how nested)

Definition at line 37 of file [CbcStrategy.hpp](#).

**7.98.3.7** `int CbcStrategy::getNested ( ) const` `[inline]`

Get model depth (i.e. how nested)

Definition at line 41 of file [CbcStrategy.hpp](#).

**7.98.3.8** `void CbcStrategy::setPreProcessState ( int state )` `[inline]`

Say preProcessing done.

Definition at line 45 of file [CbcStrategy.hpp](#).

**7.98.3.9** `int CbcStrategy::preProcessState ( ) const` `[inline]`

See what sort of preprocessing was done.

Definition at line 49 of file [CbcStrategy.hpp](#).

**7.98.3.10** `CglPreProcess* CbcStrategy::process ( ) const` `[inline]`

Pre-processing object.

Definition at line 53 of file [CbcStrategy.hpp](#).



7.98.3.11 void CbcStrategy::deletePreProcess ( )

Delete pre-processing object to save memory.

7.98.3.12 virtual CbcNodeInfo\* CbcStrategy::fullNodeInfo ( CbcModel \* model, int numberOfRowsAtContinuous ) const  
[virtual]

Return a new Full node information pointer (descendant of CbcFullNodeInfo)

7.98.3.13 virtual CbcNodeInfo\* CbcStrategy::partialNodeInfo ( CbcModel \* model, CbcNodeInfo \* parent, CbcNode \* owner, int numberChangedBounds, const int \* variables, const double \* boundChanges, const CoinWarmStartDiff \* basisDiff ) const [virtual]

Return a new Partial node information pointer (descendant of CbcPartialNodeInfo)

7.98.3.14 virtual void CbcStrategy::generateCpp ( FILE \* ) [inline],[virtual]

Create C++ lines to get to current state.

Reimplemented in CbcStrategyDefault.

Definition at line 66 of file CbcStrategy.hpp.

7.98.3.15 virtual int CbcStrategy::status ( CbcModel \* model, CbcNodeInfo \* parent, int whereFrom ) [virtual]

After a CbcModel::resolve this can return a status -1 no effect 0 treat as optimal 1 as 0 but do not do any more resolves (i.e.

no more cuts) 2 treat as infeasible

## 7.98.4 Member Data Documentation

7.98.4.1 int CbcStrategy::depth\_ [protected]

Model depth.

Definition at line 81 of file CbcStrategy.hpp.

7.98.4.2 int CbcStrategy::preProcessState\_ [protected]

PreProcessing state - -1 infeasible 0 off 1 was done (so need post-processing)

Definition at line 87 of file CbcStrategy.hpp.

7.98.4.3 CglPreProcess\* CbcStrategy::process\_ [protected]

If preprocessing then this is object.

Definition at line 89 of file CbcStrategy.hpp.

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

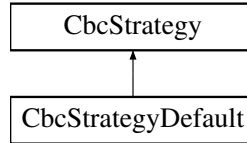
- /home/ted/COIN/trunk/Cbc/src/CbcStrategy.hpp

## 7.99 CbcStrategyDefault Class Reference

Default class.

```
#include <CbcStrategy.hpp>
```

Inheritance diagram for CbcStrategyDefault:



#### Public Member Functions

- [CbcStrategyDefault](#) (int cutsOnlyAtRoot=1, int numberStrong=5, int numberBeforeTrust=0, int printLevel=0)
- [CbcStrategyDefault](#) (const [CbcStrategyDefault](#) &)
- [~CbcStrategyDefault](#) ()
- virtual [CbcStrategy](#) \* [clone](#) () const  
*Clone.*
- virtual void [setupCutGenerators](#) ([CbcModel](#) &model)  
*Setup cut generators.*
- virtual void [setupHeuristics](#) ([CbcModel](#) &model)  
*Setup heuristics.*
- virtual void [setupPrinting](#) ([CbcModel](#) &model, int modelLogLevel)  
*Do printing stuff.*
- virtual void [setupOther](#) ([CbcModel](#) &model)  
*Other stuff e.g. strong branching.*
- void [setupPreProcessing](#) (int desired=1, int passes=10)  
*Set up preProcessing - see below.*
- int [desiredPreProcess](#) () const  
*See what sort of preprocessing wanted.*
- int [preProcessPasses](#) () const  
*See how many passes wanted.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

#### Protected Attributes

- int [cutsOnlyAtRoot\\_](#)
- int [numberStrong\\_](#)
- int [numberBeforeTrust\\_](#)
- int [printLevel\\_](#)
- int [desiredPreProcess\\_](#)  
*Desired pre-processing 0 - none 1 - ordinary 2 - find sos 3 - find cliques 4 - more aggressive sos 5 - add integer slacks.*
- int [preProcessPasses\\_](#)  
*Number of pre-processing passes.*

#### 7.99.1 Detailed Description

Default class.

Definition at line 131 of file CbcStrategy.hpp.

## 7.99.2 Constructor & Destructor Documentation

7.99.2.1 `CbcStrategyDefault::CbcStrategyDefault ( int cutsOnlyAtRoot = 1, int numberStrong = 5, int numberBeforeTrust = 0, int printLevel = 0 )`

7.99.2.2 `CbcStrategyDefault::CbcStrategyDefault ( const CbcStrategyDefault & )`

7.99.2.3 `CbcStrategyDefault::~~CbcStrategyDefault ( )`

## 7.99.3 Member Function Documentation

7.99.3.1 `virtual CbcStrategy* CbcStrategyDefault::clone ( ) const` [virtual]

Clone.

Implements [CbcStrategy](#).

7.99.3.2 `virtual void CbcStrategyDefault::setupCutGenerators ( CbcModel & model )` [virtual]

Setup cut generators.

Implements [CbcStrategy](#).

7.99.3.3 `virtual void CbcStrategyDefault::setupHeuristics ( CbcModel & model )` [virtual]

Setup heuristics.

Implements [CbcStrategy](#).

7.99.3.4 `virtual void CbcStrategyDefault::setupPrinting ( CbcModel & model, int modelLogLevel )` [virtual]

Do printing stuff.

Implements [CbcStrategy](#).

7.99.3.5 `virtual void CbcStrategyDefault::setupOther ( CbcModel & model )` [virtual]

Other stuff e.g. strong branching.

Implements [CbcStrategy](#).

7.99.3.6 `void CbcStrategyDefault::setupPreProcessing ( int desired = 1, int passes = 10 )` [inline]

Set up preProcessing - see below.

Definition at line 158 of file `CbcStrategy.hpp`.

7.99.3.7 `int CbcStrategyDefault::desiredPreProcess ( ) const` [inline]

See what sort of preprocessing wanted.

Definition at line 163 of file `CbcStrategy.hpp`.

7.99.3.8 `int CbcStrategyDefault::preProcessPasses ( ) const` [inline]

See how many passes wanted.

Definition at line 167 of file `CbcStrategy.hpp`.

7.99.3.9 `virtual void CbcStrategyDefault::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcStrategy](#).

#### 7.99.4 Member Data Documentation

7.99.4.1 `int CbcStrategyDefault::cutsOnlyAtRoot_` [protected]

Definition at line 177 of file `CbcStrategy.hpp`.

7.99.4.2 `int CbcStrategyDefault::numberStrong_` [protected]

Definition at line 180 of file `CbcStrategy.hpp`.

7.99.4.3 `int CbcStrategyDefault::numberBeforeTrust_` [protected]

Definition at line 183 of file `CbcStrategy.hpp`.

7.99.4.4 `int CbcStrategyDefault::printLevel_` [protected]

Definition at line 186 of file `CbcStrategy.hpp`.

7.99.4.5 `int CbcStrategyDefault::desiredPreProcess_` [protected]

Desired pre-processing 0 - none 1 - ordinary 2 - find sos 3 - find cliques 4 - more aggressive sos 5 - add integer slacks.

Definition at line 196 of file `CbcStrategy.hpp`.

7.99.4.6 `int CbcStrategyDefault::preProcessPasses_` [protected]

Number of pre-processing passes.

Definition at line 198 of file `CbcStrategy.hpp`.

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

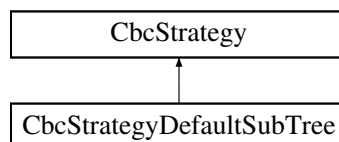
- [/home/ted/COIN/trunk/Cbc/src/CbcStrategy.hpp](#)

## 7.100 CbcStrategyDefaultSubTree Class Reference

Default class for sub trees.

```
#include <CbcStrategy.hpp>
```

Inheritance diagram for `CbcStrategyDefaultSubTree`:



## Public Member Functions

- [CbcStrategyDefaultSubTree](#) ([CbcModel](#) \*parent=NULL, int cutsOnlyAtRoot=1, int numberStrong=5, int numberBeforeTrust=0, int printLevel=0)
- [CbcStrategyDefaultSubTree](#) (const [CbcStrategyDefaultSubTree](#) &)
- [~CbcStrategyDefaultSubTree](#) ()
- virtual [CbcStrategy](#) \* [clone](#) () const  
*Clone.*
- virtual void [setupCutGenerators](#) ([CbcModel](#) &model)  
*Setup cut generators.*
- virtual void [setupHeuristics](#) ([CbcModel](#) &model)  
*Setup heuristics.*
- virtual void [setupPrinting](#) ([CbcModel](#) &model, int modelLogLevel)  
*Do printing stuff.*
- virtual void [setupOther](#) ([CbcModel](#) &model)  
*Other stuff e.g. strong branching.*

## Protected Attributes

- [CbcModel](#) \* [parentModel\\_](#)
- int [cutsOnlyAtRoot\\_](#)
- int [numberStrong\\_](#)
- int [numberBeforeTrust\\_](#)
- int [printLevel\\_](#)

## 7.100.1 Detailed Description

Default class for sub trees.

Definition at line 209 of file CbcStrategy.hpp.

## 7.100.2 Constructor &amp; Destructor Documentation

7.100.2.1 [CbcStrategyDefaultSubTree::CbcStrategyDefaultSubTree](#) ( [CbcModel](#) \* *parent* = NULL, int *cutsOnlyAtRoot* = 1, int *numberStrong* = 5, int *numberBeforeTrust* = 0, int *printLevel* = 0 )

7.100.2.2 [CbcStrategyDefaultSubTree::CbcStrategyDefaultSubTree](#) ( const [CbcStrategyDefaultSubTree](#) & )

7.100.2.3 [CbcStrategyDefaultSubTree::~~CbcStrategyDefaultSubTree](#) ( )

## 7.100.3 Member Function Documentation

7.100.3.1 virtual [CbcStrategy](#)\* [CbcStrategyDefaultSubTree::clone](#) ( ) const [virtual]

Clone.

Implements [CbcStrategy](#).

7.100.3.2 virtual void [CbcStrategyDefaultSubTree::setupCutGenerators](#) ( [CbcModel](#) & *model* ) [virtual]

Setup cut generators.

Implements [CbcStrategy](#).

7.100.3.3 `virtual void CbcStrategyDefaultSubTree::setupHeuristics ( CbcModel & model ) [virtual]`

Setup heuristics.

Implements [CbcStrategy](#).

7.100.3.4 `virtual void CbcStrategyDefaultSubTree::setupPrinting ( CbcModel & model, int modelLogLevel ) [virtual]`

Do printing stuff.

Implements [CbcStrategy](#).

7.100.3.5 `virtual void CbcStrategyDefaultSubTree::setupOther ( CbcModel & model ) [virtual]`

Other stuff e.g. strong branching.

Implements [CbcStrategy](#).

#### 7.100.4 Member Data Documentation

7.100.4.1 `CbcModel* CbcStrategyDefaultSubTree::parentModel_ [protected]`

Definition at line 238 of file `CbcStrategy.hpp`.

7.100.4.2 `int CbcStrategyDefaultSubTree::cutsOnlyAtRoot_ [protected]`

Definition at line 240 of file `CbcStrategy.hpp`.

7.100.4.3 `int CbcStrategyDefaultSubTree::numberStrong_ [protected]`

Definition at line 243 of file `CbcStrategy.hpp`.

7.100.4.4 `int CbcStrategyDefaultSubTree::numberBeforeTrust_ [protected]`

Definition at line 246 of file `CbcStrategy.hpp`.

7.100.4.5 `int CbcStrategyDefaultSubTree::printLevel_ [protected]`

Definition at line 249 of file `CbcStrategy.hpp`.

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

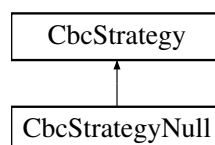
- `/home/ted/COIN/trunk/Cbc/src/CbcStrategy.hpp`

### 7.101 CbcStrategyNull Class Reference

Null class.

```
#include <CbcStrategy.hpp>
```

Inheritance diagram for `CbcStrategyNull`:



## Public Member Functions

- [CbcStrategyNull](#) ( )
- [CbcStrategyNull](#) (const [CbcStrategyNull](#) &rhs)
- [~CbcStrategyNull](#) ( )
- virtual [CbcStrategy](#) \* [clone](#) ( ) const  
*Clone.*
- virtual void [setupCutGenerators](#) ([CbcModel](#) &)  
*Setup cut generators.*
- virtual void [setupHeuristics](#) ([CbcModel](#) &)  
*Setup heuristics.*
- virtual void [setupPrinting](#) ([CbcModel](#) &, int)  
*Do printing stuff.*
- virtual void [setupOther](#) ([CbcModel](#) &)  
*Other stuff e.g. strong branching.*

## Additional Inherited Members

## 7.101.1 Detailed Description

Null class.

Definition at line 95 of file [CbcStrategy.hpp](#).

## 7.101.2 Constructor &amp; Destructor Documentation

7.101.2.1 [CbcStrategyNull::CbcStrategyNull](#) ( ) [\[inline\]](#)

Definition at line 99 of file [CbcStrategy.hpp](#).

7.101.2.2 [CbcStrategyNull::CbcStrategyNull](#) ( const [CbcStrategyNull](#) & *rhs* ) [\[inline\]](#)

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

7.101.2.3 [CbcStrategyNull::~~CbcStrategyNull](#) ( ) [\[inline\]](#)

Definition at line 105 of file [CbcStrategy.hpp](#).

## 7.101.3 Member Function Documentation

7.101.3.1 virtual [CbcStrategy](#)\* [CbcStrategyNull::clone](#) ( ) const [\[inline\]](#),[\[virtual\]](#)

Clone.

Implements [CbcStrategy](#).

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

7.101.3.2 virtual void [CbcStrategyNull::setupCutGenerators](#) ( [CbcModel](#) & ) [\[inline\]](#),[\[virtual\]](#)

Setup cut generators.

Implements [CbcStrategy](#).

Definition at line 113 of file [CbcStrategy.hpp](#).

7.101.3.3 `virtual void CbcStrategyNull::setupHeuristics ( CbcModel & ) [inline],[virtual]`

Setup heuristics.

Implements [CbcStrategy](#).

Definition at line 115 of file CbcStrategy.hpp.

7.101.3.4 `virtual void CbcStrategyNull::setupPrinting ( CbcModel &, int ) [inline],[virtual]`

Do printing stuff.

Implements [CbcStrategy](#).

Definition at line 117 of file CbcStrategy.hpp.

7.101.3.5 `virtual void CbcStrategyNull::setupOther ( CbcModel & ) [inline],[virtual]`

Other stuff e.g. strong branching.

Implements [CbcStrategy](#).

Definition at line 119 of file CbcStrategy.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcStrategy.hpp](#)

## 7.102 CbcStrongInfo Struct Reference

Abstract base class for 'objects'.

```
#include <CbcObject.hpp>
```

### Public Attributes

- [CbcBranchingObject](#) \* [possibleBranch](#)
- double [upMovement](#)
- double [downMovement](#)
- int [numIntInfeasUp](#)
- int [numObjInfeasUp](#)
- bool [finishedUp](#)
- int [numItersUp](#)
- int [numIntInfeasDown](#)
- int [numObjInfeasDown](#)
- bool [finishedDown](#)
- int [numItersDown](#)
- int [objectNumber](#)
- int [fix](#)

### 7.102.1 Detailed Description

Abstract base class for 'objects'.

It now just has stuff that `OsiObject` does not have



The branching model used in Cbc is based on the idea of an *object*. In the abstract, an object is something that has a feasible region, can be evaluated for infeasibility, can be branched on (*i.e.*, there's some constructive action to be taken to move toward feasibility), and allows comparison of the effect of branching.

This class ([CbcObject](#)) is the base class for an object. To round out the branching model, the class [CbcBranchingObject](#) describes how to perform a branch, and the class [CbcBranchDecision](#) describes how to compare two CbcBranching-Objects.

To create a new type of object you need to provide three methods: `#infeasibility()`, `#feasibleRegion()`, and `#createCbcBranch()`, described below.

This base class is primarily virtual to allow for any form of structure. Any form of discontinuity is allowed.

**Todo** The notion that all branches are binary (two arms) is wired into the implementation of [CbcObject](#), [CbcBranchingObject](#), and [CbcBranchDecision](#). Changing this will require a moderate amount of recoding.

Definition at line 51 of file CbcObject.hpp.

## 7.102.2 Member Data Documentation

### 7.102.2.1 CbcBranchingObject\* CbcStrongInfo::possibleBranch

Definition at line 52 of file CbcObject.hpp.

### 7.102.2.2 double CbcStrongInfo::upMovement

Definition at line 53 of file CbcObject.hpp.

### 7.102.2.3 double CbcStrongInfo::downMovement

Definition at line 54 of file CbcObject.hpp.

### 7.102.2.4 int CbcStrongInfo::numIntInfeasUp

Definition at line 55 of file CbcObject.hpp.

### 7.102.2.5 int CbcStrongInfo::numObjInfeasUp

Definition at line 56 of file CbcObject.hpp.

### 7.102.2.6 bool CbcStrongInfo::finishedUp

Definition at line 57 of file CbcObject.hpp.

### 7.102.2.7 int CbcStrongInfo::numItersUp

Definition at line 58 of file CbcObject.hpp.

### 7.102.2.8 int CbcStrongInfo::numIntInfeasDown

Definition at line 59 of file CbcObject.hpp.

### 7.102.2.9 int CbcStrongInfo::numObjInfeasDown

Definition at line 60 of file CbcObject.hpp.

#### 7.102.2.10 `bool CbcStrongInfo::finishedDown`

Definition at line 61 of file `CbcObject.hpp`.

#### 7.102.2.11 `int CbcStrongInfo::numItersDown`

Definition at line 62 of file `CbcObject.hpp`.

#### 7.102.2.12 `int CbcStrongInfo::objectNumber`

Definition at line 63 of file `CbcObject.hpp`.

#### 7.102.2.13 `int CbcStrongInfo::fix`

Definition at line 64 of file `CbcObject.hpp`.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcObject.hpp](#)

### 7.103 `CbcThread` Class Reference

A class to encapsulate thread stuff.

```
#include <CbcThread.hpp>
```

#### Public Member Functions

- [CbcThread\(\)](#)
- `virtual ~CbcThread()`

#### 7.103.1 Detailed Description

A class to encapsulate thread stuff.

Definition at line 418 of file `CbcThread.hpp`.

#### 7.103.2 Constructor & Destructor Documentation

##### 7.103.2.1 `CbcThread::CbcThread( )` `[inline]`

Definition at line 421 of file `CbcThread.hpp`.

##### 7.103.2.2 `virtual CbcThread::~~CbcThread( )` `[inline]`, `[virtual]`

Definition at line 423 of file `CbcThread.hpp`.

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

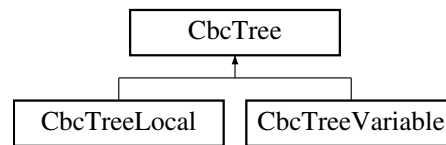
- [/home/ted/COIN/trunk/Cbc/src/CbcThread.hpp](#)

### 7.104 `CbcTree` Class Reference

Using MS heap implementation.

```
#include <CbcTree.hpp>
```

Inheritance diagram for CbcTree:



## Public Member Functions

### Constructors and related

- [CbcTree](#) ()  
*Default Constructor.*
- [CbcTree](#) (const [CbcTree](#) &rhs)  
*Copy constructor.*
- [CbcTree](#) & [operator=](#) (const [CbcTree](#) &rhs)  
*= operator*
- virtual [~CbcTree](#) ()  
*Destructor.*
- virtual [CbcTree](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*)  
*Create C++ lines to get to current state.*

### Heap access and maintenance methods

- void [setComparison](#) ([CbcCompareBase](#) &compare)  
*Set comparison function and resort heap.*
- virtual [CbcNode](#) \* [top](#) () const  
*Return the top node of the heap.*
- virtual void [push](#) ([CbcNode](#) \*x)  
*Add a node to the heap.*
- virtual void [pop](#) ()  
*Remove the top node from the heap.*
- virtual [CbcNode](#) \* [bestNode](#) (double cutoff)  
*Gets best node and takes off heap.*
- virtual void [rebuild](#) ()  
*Rebuild the heap.*

### Direct node access methods

- virtual bool [empty](#) ()  
*Test for an empty tree.*
- virtual int [size](#) () const  
*Return size.*
- [CbcNode](#) \* [operator\[\]](#) (int i) const  
*Return a node pointer.*
- [CbcNode](#) \* [nodePointer](#) (int i) const  
*Return a node pointer.*
- void [realpop](#) ()
- void [fixTop](#) ()

*After changing data in the top node, fix the heap.*

- void `realpush (CbcNode *node)`

### Search tree maintenance

- virtual void `cleanTree (CbcModel *model, double cutoff, double &bestPossibleObjective)`

*Prune the tree using an objective function cutoff.*

- `CbcNode * bestAlternate ()`

*Get best on list using alternate method.*

- virtual void `endSearch ()`

*We may have got an intelligent tree so give it one more chance.*

- virtual double `getBestPossibleObjective ()`

*Get best possible objective function in the tree.*

- void `resetNodeNumbers ()`

*Reset maximum node number.*

- int `maximumNodeNumber () const`

*Get maximum node number.*

- void `setNumberBranching (int value)`

*Set number of branches.*

- int `getNumberBranching () const`

*Get number of branches.*

- void `setMaximumBranching (int value)`

*Set maximum branches.*

- int `getMaximumBranching () const`

*Get maximum branches.*

- unsigned int \* `branched () const`

*Get branched variables.*

- int \* `newBounds () const`

*Get bounds.*

- double `lastObjective () const`

*Last objective in branch-and-cut search tree.*

- int `lastDepth () const`

*Last depth in branch-and-cut search tree.*

- int `lastUnsatisfied () const`

*Last number of objects unsatisfied.*

- void `addBranchingInformation (const CbcModel *model, const CbcNodeInfo *nodeInfo, const double *currentLower, const double *currentUpper)`

*Adds branching information to complete state.*

- void `increaseSpace ()`

*Increase space for data.*

### Protected Attributes

- `std::vector< CbcNode * > nodes_`

*Storage vector for the heap.*

- `CbcCompare comparison_`

*Sort predicate for heap ordering.*

- int `maximumNodeNumber_`

*Maximum "node" number so far to split ties.*

- int `numberBranching_`

*Size of variable list.*

- int `maximumBranching_`

*Maximum size of variable list.*

- double [lastObjective\\_](#)

*Objective of last node pushed on tree.*

- int [lastDepth\\_](#)

*Depth of last node pushed on tree.*

- int [lastUnsatisfied\\_](#)

*Number unsatisfied of last node pushed on tree.*

- unsigned int \* [branched\\_](#)

*Integer variables branched or bounded top bit set if new upper bound next bit set if a branch.*

- int \* [newBound\\_](#)

*New bound.*

#### 7.104.1 Detailed Description

Using MS heap implementation.

It's unclear if this is needed any longer, or even if it should be allowed. Cbc occasionally tries to do things to the tree (typically tweaking the comparison predicate) that can cause a violation of the heap property (parent better than either child). In a debug build, Microsoft's heap implementation does checks that detect this and fail. This symbol switched to an alternate implementation of [CbcTree](#), and there are clearly differences, but no explanation as to why or what for.

As of 100921, the code is cleaned up to make it through 'cbc -unitTest' without triggering 'Invalid heap' in an MSVS debug build. The method `validateHeap()` can be used for debugging if this turns up again.

Controls search tree debugging

In order to have `validateHeap()` available, set `CBC_DEBUG_HEAP` to 1 or higher.

- 1 calls `validateHeap()` after each change to the heap
- 2 will print a line for major operations (clean, set comparison, etc.)
- 3 will print information about each push and pop

```
#define CBC_DEBUG_HEAP 1
```

Implementation of the live set as a heap.

This class is used to hold the set of live nodes in the search tree.

Definition at line 53 of file `CbcTree.hpp`.

#### 7.104.2 Constructor & Destructor Documentation

##### 7.104.2.1 CbcTree::CbcTree ( )

Default Constructor.

##### 7.104.2.2 CbcTree::CbcTree ( const CbcTree & rhs )

Copy constructor.

##### 7.104.2.3 virtual CbcTree::~~CbcTree ( ) [virtual]

Destructor.

### 7.104.3 Member Function Documentation

#### 7.104.3.1 `CbcTree& CbcTree::operator= ( const CbcTree & rhs )`

= operator

#### 7.104.3.2 `virtual CbcTree* CbcTree::clone ( ) const [virtual]`

Clone.

Reimplemented in [CbcTreeVariable](#), and [CbcTreeLocal](#).

#### 7.104.3.3 `virtual void CbcTree::generateCpp ( FILE * ) [inline],[virtual]`

Create C++ lines to get to current state.

Reimplemented in [CbcTreeVariable](#), and [CbcTreeLocal](#).

Definition at line 74 of file CbcTree.hpp.

#### 7.104.3.4 `void CbcTree::setComparison ( CbcCompareBase & compare )`

Set comparison function and resort heap.

#### 7.104.3.5 `virtual CbcNode* CbcTree::top ( ) const [virtual]`

Return the top node of the heap.

Reimplemented in [CbcTreeVariable](#), and [CbcTreeLocal](#).

#### 7.104.3.6 `virtual void CbcTree::push ( CbcNode * x ) [virtual]`

Add a node to the heap.

Reimplemented in [CbcTreeVariable](#), and [CbcTreeLocal](#).

#### 7.104.3.7 `virtual void CbcTree::pop ( ) [virtual]`

Remove the top node from the heap.

Reimplemented in [CbcTreeVariable](#), and [CbcTreeLocal](#).

#### 7.104.3.8 `virtual CbcNode* CbcTree::bestNode ( double cutoff ) [virtual]`

Gets best node and takes off heap.

Before returning the node from the top of the heap, the node is offered an opportunity to reevaluate itself. Callers should be prepared to check that the node returned is suitable for use.

#### 7.104.3.9 `virtual void CbcTree::rebuild ( ) [virtual]`

Rebuild the heap.

#### 7.104.3.10 `virtual bool CbcTree::empty ( ) [virtual]`

Test for an empty tree.

Reimplemented in [CbcTreeVariable](#), and [CbcTreeLocal](#).

7.104.3.11 `virtual int CbcTree::size ( ) const` `[inline],[virtual]`

Return size.

Definition at line 109 of file CbcTree.hpp.

7.104.3.12 `CbcNode* CbcTree::operator[] ( int i ) const` `[inline]`

Return a node pointer.

Definition at line 112 of file CbcTree.hpp.

7.104.3.13 `CbcNode* CbcTree::nodePointer ( int i ) const` `[inline]`

Return a node pointer.

Definition at line 115 of file CbcTree.hpp.

7.104.3.14 `void CbcTree::realpop ( )`

7.104.3.15 `void CbcTree::fixTop ( )`

After changing data in the top node, fix the heap.

7.104.3.16 `void CbcTree::realpush ( CbcNode * node )`

7.104.3.17 `virtual void CbcTree::cleanTree ( CbcModel * model, double cutoff, double & bestPossibleObjective )` `[virtual]`

Prune the tree using an objective function cutoff.

This routine removes all nodes with objective worse than the specified cutoff value. It also sets bestPossibleObjective to the best objective over remaining nodes.

7.104.3.18 `CbcNode* CbcTree::bestAlternate ( )`

Get best on list using alternate method.

7.104.3.19 `virtual void CbcTree::endSearch ( )` `[inline],[virtual]`

We may have got an intelligent tree so give it one more chance.

Reimplemented in [CbcTreeVariable](#), and [CbcTreeLocal](#).

Definition at line 136 of file CbcTree.hpp.

7.104.3.20 `virtual double CbcTree::getBestPossibleObjective ( )` `[virtual]`

Get best possible objective function in the tree.

7.104.3.21 `void CbcTree::resetNodeNumbers ( )` `[inline]`

Reset maximum node number.

Definition at line 142 of file CbcTree.hpp.

7.104.3.22 `int CbcTree::maximumNodeNumber ( ) const` `[inline]`

Get maximum node number.

Definition at line 145 of file CbcTree.hpp.

**7.104.3.23** `void CbcTree::setNumberBranching ( int value ) [inline]`

Set number of branches.

Definition at line 148 of file CbcTree.hpp.

**7.104.3.24** `int CbcTree::getNumberBranching ( ) const [inline]`

Get number of branches.

Definition at line 151 of file CbcTree.hpp.

**7.104.3.25** `void CbcTree::setMaximumBranching ( int value ) [inline]`

Set maximum branches.

Definition at line 154 of file CbcTree.hpp.

**7.104.3.26** `int CbcTree::getMaximumBranching ( ) const [inline]`

Get maximum branches.

Definition at line 157 of file CbcTree.hpp.

**7.104.3.27** `unsigned int* CbcTree::branched ( ) const [inline]`

Get branched variables.

Definition at line 160 of file CbcTree.hpp.

**7.104.3.28** `int* CbcTree::newBounds ( ) const [inline]`

Get bounds.

Definition at line 163 of file CbcTree.hpp.

**7.104.3.29** `double CbcTree::lastObjective ( ) const [inline]`

Last objective in branch-and-cut search tree.

Definition at line 166 of file CbcTree.hpp.

**7.104.3.30** `int CbcTree::lastDepth ( ) const [inline]`

Last depth in branch-and-cut search tree.

Definition at line 170 of file CbcTree.hpp.

**7.104.3.31** `int CbcTree::lastUnsatisfied ( ) const [inline]`

Last number of objects unsatisfied.

Definition at line 174 of file CbcTree.hpp.

**7.104.3.32** `void CbcTree::addBranchingInformation ( const CbcModel * model, const CbcNodeInfo * nodeInfo, const double * currentLower, const double * currentUpper )`

Adds branching information to complete state.

**7.104.3.33** `void CbcTree::increaseSpace ( )`

Increase space for data.



#### 7.104.4 Member Data Documentation

##### 7.104.4.1 `std::vector<CbcNode*> CbcTree::nodes_` [protected]

Storage vector for the heap.

Definition at line 195 of file CbcTree.hpp.

##### 7.104.4.2 `CbcCompare CbcTree::comparison_` [protected]

Sort predicate for heap ordering.

Definition at line 197 of file CbcTree.hpp.

##### 7.104.4.3 `int CbcTree::maximumNodeNumber_` [protected]

Maximum "node" number so far to split ties.

Definition at line 199 of file CbcTree.hpp.

##### 7.104.4.4 `int CbcTree::numberBranching_` [protected]

Size of variable list.

Definition at line 201 of file CbcTree.hpp.

##### 7.104.4.5 `int CbcTree::maximumBranching_` [protected]

Maximum size of variable list.

Definition at line 203 of file CbcTree.hpp.

##### 7.104.4.6 `double CbcTree::lastObjective_` [protected]

Objective of last node pushed on tree.

Definition at line 205 of file CbcTree.hpp.

##### 7.104.4.7 `int CbcTree::lastDepth_` [protected]

Depth of last node pushed on tree.

Definition at line 207 of file CbcTree.hpp.

##### 7.104.4.8 `int CbcTree::lastUnsatisfied_` [protected]

Number unsatisfied of last node pushed on tree.

Definition at line 209 of file CbcTree.hpp.

##### 7.104.4.9 `unsigned int* CbcTree::branched_` [protected]

Integer variables branched or bounded top bit set if new upper bound next bit set if a branch.

Definition at line 214 of file CbcTree.hpp.

##### 7.104.4.10 `int* CbcTree::newBound_` [protected]

New bound.

Definition at line 216 of file CbcTree.hpp.

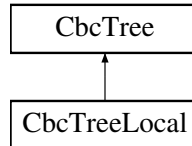
The documentation for this class was generated from the following file:

- </home/ted/COIN/trunk/Cbc/src/CbcTree.hpp>

## 7.105 CbcTreeLocal Class Reference

```
#include <CbcTreeLocal.hpp>
```

Inheritance diagram for CbcTreeLocal:



### Public Member Functions

- [CbcTreeLocal](#) ()
- [CbcTreeLocal](#) ([CbcModel](#) \*model, const double \*solution, int [range](#)=10, int [typeCuts](#)=0, int [maxDiversification](#)=0, int [timeLimit](#)=1000000, int [nodeLimit](#)=1000000, bool [refine](#)=true)
- [CbcTreeLocal](#) (const [CbcTreeLocal](#) &rhs)
- [CbcTreeLocal](#) & [operator=](#) (const [CbcTreeLocal](#) &rhs)
- virtual [~CbcTreeLocal](#) ()
- virtual [CbcTree](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

### Heap access and maintenance methods

- virtual [CbcNode](#) \* [top](#) () const  
*Return the top node of the heap.*
- virtual void [push](#) ([CbcNode](#) \*x)  
*Add a node to the heap.*
- virtual void [pop](#) ()  
*Remove the top node from the heap.*

### Other stuff

- int [createCut](#) (const double \*solution, [OsiRowCut](#) &cut)  
*Create cut - return -1 if bad, 0 if okay and 1 if cut is everything.*
- virtual bool [empty](#) ()  
*Test if empty \*\*\* note may be overridden.*
- virtual void [endSearch](#) ()  
*We may have got an intelligent tree so give it one more chance.*
- void [reverseCut](#) (int state, double bias=0.0)  
*Other side of last cut branch (if bias==rhs\_ will be weakest possible)*
- void [deleteCut](#) ([OsiRowCut](#) &cut)  
*Delete last cut branch.*
- void [passInSolution](#) (const double \*solution, double solutionValue)

*Pass in solution (so can be used after heuristic)*

- int [range](#) () const
- void [setRange](#) (int value)
- int [typeCuts](#) () const
- void [setTypeCuts](#) (int value)
- int [maxDiversification](#) () const
- void [setMaxDiversification](#) (int value)
- int [timeLimit](#) () const
- void [setTimeLimit](#) (int value)
- int [nodeLimit](#) () const
- void [setNodeLimit](#) (int value)
- bool [refine](#) () const
- void [setRefine](#) (bool yesNo)

#### Additional Inherited Members

##### 7.105.1 Detailed Description

Definition at line 40 of file CbcTreeLocal.hpp.

##### 7.105.2 Constructor & Destructor Documentation

###### 7.105.2.1 CbcTreeLocal::CbcTreeLocal ( )

7.105.2.2 **CbcTreeLocal::CbcTreeLocal ( CbcModel \* model, const double \* solution, int range = 10, int typeCuts = 0, int maxDiversification = 0, int timeLimit = 1000000, int nodeLimit = 1000000, bool refine = true )**

###### 7.105.2.3 CbcTreeLocal::CbcTreeLocal ( const CbcTreeLocal & rhs )

###### 7.105.2.4 virtual CbcTreeLocal::~CbcTreeLocal ( ) [virtual]

##### 7.105.3 Member Function Documentation

###### 7.105.3.1 CbcTreeLocal& CbcTreeLocal::operator= ( const CbcTreeLocal & rhs )

###### 7.105.3.2 virtual CbcTree\* CbcTreeLocal::clone ( ) const [virtual]

Clone.

Reimplemented from [CbcTree](#).

###### 7.105.3.3 virtual void CbcTreeLocal::generateCpp ( FILE \* fp ) [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcTree](#).

###### 7.105.3.4 virtual CbcNode\* CbcTreeLocal::top ( ) const [virtual]

Return the top node of the heap.

Reimplemented from [CbcTree](#).

###### 7.105.3.5 virtual void CbcTreeLocal::push ( CbcNode \* x ) [virtual]

Add a node to the heap.

Reimplemented from [CbcTree](#).

7.105.3.6 `virtual void CbcTreeLocal::pop ( ) [virtual]`

Remove the top node from the heap.

Reimplemented from [CbcTree](#).

7.105.3.7 `int CbcTreeLocal::createCut ( const double * solution, OsiRowCut & cut )`

Create cut - return -1 if bad, 0 if okay and 1 if cut is everything.

7.105.3.8 `virtual bool CbcTreeLocal::empty ( ) [virtual]`

Test if empty \*\*\* note may be overridden.

Reimplemented from [CbcTree](#).

7.105.3.9 `virtual void CbcTreeLocal::endSearch ( ) [virtual]`

We may have got an intelligent tree so give it one more chance.

Reimplemented from [CbcTree](#).

7.105.3.10 `void CbcTreeLocal::reverseCut ( int state, double bias = 0.0 )`

Other side of last cut branch (if *bias*==*rhs\_* will be weakest possible)

7.105.3.11 `void CbcTreeLocal::deleteCut ( OsiRowCut & cut )`

Delete last cut branch.

7.105.3.12 `void CbcTreeLocal::passInSolution ( const double * solution, double solutionValue )`

Pass in solution (so can be used after heuristic)

7.105.3.13 `int CbcTreeLocal::range ( ) const [inline]`

Definition at line 107 of file *CbcTreeLocal.hpp*.

7.105.3.14 `void CbcTreeLocal::setRange ( int value ) [inline]`

Definition at line 111 of file *CbcTreeLocal.hpp*.

7.105.3.15 `int CbcTreeLocal::typeCuts ( ) const [inline]`

Definition at line 115 of file *CbcTreeLocal.hpp*.

7.105.3.16 `void CbcTreeLocal::setTypeCuts ( int value ) [inline]`

Definition at line 119 of file *CbcTreeLocal.hpp*.

7.105.3.17 `int CbcTreeLocal::maxDiversification ( ) const [inline]`

Definition at line 123 of file *CbcTreeLocal.hpp*.

7.105.3.18 `void CbcTreeLocal::setMaxDiversification ( int value ) [inline]`

Definition at line 127 of file *CbcTreeLocal.hpp*.

7.105.3.19 `int CbcTreeLocal::timeLimit ( ) const [inline]`

Definition at line 131 of file CbcTreeLocal.hpp.

7.105.3.20 `void CbcTreeLocal::setTimeLimit ( int value ) [inline]`

Definition at line 135 of file CbcTreeLocal.hpp.

7.105.3.21 `int CbcTreeLocal::nodeLimit ( ) const [inline]`

Definition at line 139 of file CbcTreeLocal.hpp.

7.105.3.22 `void CbcTreeLocal::setNodeLimit ( int value ) [inline]`

Definition at line 143 of file CbcTreeLocal.hpp.

7.105.3.23 `bool CbcTreeLocal::refine ( ) const [inline]`

Definition at line 147 of file CbcTreeLocal.hpp.

7.105.3.24 `void CbcTreeLocal::setRefine ( bool yesNo ) [inline]`

Definition at line 151 of file CbcTreeLocal.hpp.

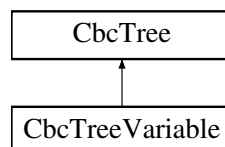
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcTreeLocal.hpp](#)

## 7.106 CbcTreeVariable Class Reference

```
#include <CbcTreeLocal.hpp>
```

Inheritance diagram for CbcTreeVariable:



### Public Member Functions

- [CbcTreeVariable](#) ()
- [CbcTreeVariable](#) ([CbcModel](#) \*model, const double \*solution, int [range](#)=10, int [typeCuts](#)=0, int [max-Diversification](#)=0, int [timeLimit](#)=1000000, int [nodeLimit](#)=1000000, bool [refine](#)=true)
- [CbcTreeVariable](#) (const [CbcTreeVariable](#) &rhs)
- [CbcTreeVariable](#) & [operator=](#) (const [CbcTreeVariable](#) &rhs)
- virtual [~CbcTreeVariable](#) ()
- virtual [CbcTree](#) \* [clone](#) () const  
*Clone.*
- virtual void [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

### Heap access and maintenance methods

- virtual `CbcNode * top () const`  
*Return the top node of the heap.*
- virtual void `push (CbcNode *x)`  
*Add a node to the heap.*
- virtual void `pop ()`  
*Remove the top node from the heap.*

#### Other stuff

- int `createCut (const double *solution, OsiRowCut &cut)`  
*Create cut - return -1 if bad, 0 if okay and 1 if cut is everything.*
- virtual bool `empty ()`  
*Test if empty \*\*\* note may be overridden.*
- virtual void `endSearch ()`  
*We may have got an intelligent tree so give it one more chance.*
- void `reverseCut (int state, double bias=0.0)`  
*Other side of last cut branch (if bias==rhs\_ will be weakest possible)*
- void `deleteCut (OsiRowCut &cut)`  
*Delete last cut branch.*
- void `passInSolution (const double *solution, double solutionValue)`  
*Pass in solution (so can be used after heuristic)*
- int `range () const`
- void `setRange (int value)`
- int `typeCuts () const`
- void `setTypeCuts (int value)`
- int `maxDiversification () const`
- void `setMaxDiversification (int value)`
- int `timeLimit () const`
- void `setTimeLimit (int value)`
- int `nodeLimit () const`
- void `setNodeLimit (int value)`
- bool `refine () const`
- void `setRefine (bool yesNo)`

#### Additional Inherited Members

##### 7.106.1 Detailed Description

Definition at line 206 of file `CbcTreeLocal.hpp`.

##### 7.106.2 Constructor & Destructor Documentation

###### 7.106.2.1 `CbcTreeVariable::CbcTreeVariable ( )`

###### 7.106.2.2 `CbcTreeVariable::CbcTreeVariable ( CbcModel * model, const double * solution, int range = 10, int typeCuts = 0, int maxDiversification = 0, int timeLimit = 10000000, int nodeLimit = 10000000, bool refine = true )`

###### 7.106.2.3 `CbcTreeVariable::CbcTreeVariable ( const CbcTreeVariable & rhs )`

###### 7.106.2.4 `virtual CbcTreeVariable::~~CbcTreeVariable ( ) [virtual]`

##### 7.106.3 Member Function Documentation

###### 7.106.3.1 `CbcTreeVariable& CbcTreeVariable::operator= ( const CbcTreeVariable & rhs )`

7.106.3.2 `virtual CbcTree* CbcTreeVariable::clone ( ) const` [virtual]

Clone.

Reimplemented from [CbcTree](#).

7.106.3.3 `virtual void CbcTreeVariable::generateCpp ( FILE * fp )` [virtual]

Create C++ lines to get to current state.

Reimplemented from [CbcTree](#).

7.106.3.4 `virtual CbcNode* CbcTreeVariable::top ( ) const` [virtual]

Return the top node of the heap.

Reimplemented from [CbcTree](#).

7.106.3.5 `virtual void CbcTreeVariable::push ( CbcNode * x )` [virtual]

Add a node to the heap.

Reimplemented from [CbcTree](#).

7.106.3.6 `virtual void CbcTreeVariable::pop ( )` [virtual]

Remove the top node from the heap.

Reimplemented from [CbcTree](#).

7.106.3.7 `int CbcTreeVariable::createCut ( const double * solution, OsiRowCut & cut )`

Create cut - return -1 if bad, 0 if okay and 1 if cut is everything.

7.106.3.8 `virtual bool CbcTreeVariable::empty ( )` [virtual]

Test if empty \*\*\* note may be overridden.

Reimplemented from [CbcTree](#).

7.106.3.9 `virtual void CbcTreeVariable::endSearch ( )` [virtual]

We may have got an intelligent tree so give it one more chance.

Reimplemented from [CbcTree](#).

7.106.3.10 `void CbcTreeVariable::reverseCut ( int state, double bias = 0.0 )`

Other side of last cut branch (if bias==rhs\_ will be weakest possible)

7.106.3.11 `void CbcTreeVariable::deleteCut ( OsiRowCut & cut )`

Delete last cut branch.

7.106.3.12 `void CbcTreeVariable::passInSolution ( const double * solution, double solutionValue )`

Pass in solution (so can be used after heuristic)

7.106.3.13 `int CbcTreeVariable::range ( ) const` [inline]

Definition at line 273 of file CbcTreeLocal.hpp.

**7.106.3.14** `void CbcTreeVariable::setRange ( int value )` `[inline]`

Definition at line 277 of file CbcTreeLocal.hpp.

**7.106.3.15** `int CbcTreeVariable::typeCuts ( ) const` `[inline]`

Definition at line 281 of file CbcTreeLocal.hpp.

**7.106.3.16** `void CbcTreeVariable::setTypeCuts ( int value )` `[inline]`

Definition at line 285 of file CbcTreeLocal.hpp.

**7.106.3.17** `int CbcTreeVariable::maxDiversification ( ) const` `[inline]`

Definition at line 289 of file CbcTreeLocal.hpp.

**7.106.3.18** `void CbcTreeVariable::setMaxDiversification ( int value )` `[inline]`

Definition at line 293 of file CbcTreeLocal.hpp.

**7.106.3.19** `int CbcTreeVariable::timeLimit ( ) const` `[inline]`

Definition at line 297 of file CbcTreeLocal.hpp.

**7.106.3.20** `void CbcTreeVariable::setTimeLimit ( int value )` `[inline]`

Definition at line 301 of file CbcTreeLocal.hpp.

**7.106.3.21** `int CbcTreeVariable::nodeLimit ( ) const` `[inline]`

Definition at line 305 of file CbcTreeLocal.hpp.

**7.106.3.22** `void CbcTreeVariable::setNodeLimit ( int value )` `[inline]`

Definition at line 309 of file CbcTreeLocal.hpp.

**7.106.3.23** `bool CbcTreeVariable::refine ( ) const` `[inline]`

Definition at line 313 of file CbcTreeLocal.hpp.

**7.106.3.24** `void CbcTreeVariable::setRefine ( bool yesNo )` `[inline]`

Definition at line 317 of file CbcTreeLocal.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcTreeLocal.hpp](#)

## 7.107 CbcUser Class Reference

A class to allow the use of unknown user functionality.

```
#include <CbcSolver.hpp>
```

### Public Member Functions

#### import/export methods



- virtual int `importData` (`CbcSolver *`, int &, char \*\*)
   
    *Import - gets full command arguments.*
- virtual void `exportSolution` (`CbcSolver *`, int, const char \*=NULL)
   
    *Export.*
- virtual void `exportData` (`CbcSolver *`)
   
    *Export Data (i.e. at very end)*
- virtual void `fillInformation` (`CbcSolver *`, `CbcSolverUsefulData` &)
   
    *Get useful stuff.*

#### usage methods

- `CoinModel *` `coinModel` () const
   
    *CoinModel if valid.*
- virtual void \* `stuff` ()
   
    *Other info - needs expanding.*
- std::string `name` () const
   
    *Name.*
- virtual void `solve` (`CbcSolver *`model, const char \*options)=0
   
    *Solve (whatever that means)*
- virtual bool `canDo` (const char \*options)=0
   
    *Returns true if function knows about option.*

#### Constructors and destructors etc

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

#### Protected Attributes

##### Private member data

- `CoinModel *` `coinModel_`
  
    *CoinModel.*
- std::string `userName_`
  
    *Name of user function.*

#### 7.107.1 Detailed Description

A class to allow the use of unknown user functionality.

For example, access to a modelling language (CbcAmpl).

Definition at line 260 of file CbcSolver.hpp.

### 7.107.2 Constructor & Destructor Documentation

#### 7.107.2.1 `CbcUser::CbcUser ( )`

Default Constructor.

#### 7.107.2.2 `CbcUser::CbcUser ( const CbcUser & rhs )`

Copy constructor.

#### 7.107.2.3 `virtual CbcUser::~~CbcUser ( ) [virtual]`

Destructor.

### 7.107.3 Member Function Documentation

#### 7.107.3.1 `virtual int CbcUser::importData ( CbcSolver *, int &, char ** ) [inline],[virtual]`

Import - gets full command arguments.

Returns

- -1 - no action
- 0 - data read in without error
- 1 - errors

Definition at line 272 of file CbcSolver.hpp.

#### 7.107.3.2 `virtual void CbcUser::exportSolution ( CbcSolver *, int, const char * =NULL ) [inline],[virtual]`

Export.

Values for mode:

- 1 OsiClpSolver
- 2 [CbcModel](#)
- add 10 if infeasible from odd situation

Definition at line 283 of file CbcSolver.hpp.

#### 7.107.3.3 `virtual void CbcUser::exportData ( CbcSolver * ) [inline],[virtual]`

Export Data (i.e. at very end)

Definition at line 287 of file CbcSolver.hpp.

#### 7.107.3.4 `virtual void CbcUser::fillInformation ( CbcSolver *, CbcSolverUsefulData & ) [inline],[virtual]`

Get useful stuff.

Definition at line 290 of file CbcSolver.hpp.

#### 7.107.3.5 `CoinModel* CbcUser::coinModel ( ) const [inline]`

CoinModel if valid.

Definition at line 297 of file CbcSolver.hpp.

7.107.3.6 `virtual void* CbcUser::stuff ( ) [inline],[virtual]`

Other info - needs expanding.

Definition at line 301 of file CbcSolver.hpp.

7.107.3.7 `std::string CbcUser::name ( ) const [inline]`

Name.

Definition at line 305 of file CbcSolver.hpp.

7.107.3.8 `virtual void CbcUser::solve ( CbcSolver * model, const char * options ) [pure virtual]`

Solve (whatever that means)

7.107.3.9 `virtual bool CbcUser::canDo ( const char * options ) [pure virtual]`

Returns true if function knows about option.

7.107.3.10 `CbcUser& CbcUser::operator= ( const CbcUser & rhs )`

Assignment operator.

7.107.3.11 `virtual CbcUser* CbcUser::clone ( ) const [pure virtual]`

Clone.

#### 7.107.4 Member Data Documentation

7.107.4.1 `CoinModel* CbcUser::coinModel_ [protected]`

CoinModel.

Definition at line 337 of file CbcSolver.hpp.

7.107.4.2 `std::string CbcUser::userName_ [protected]`

Name of user function.

Definition at line 340 of file CbcSolver.hpp.

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

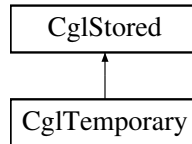
- [/home/ted/COIN/trunk/Cbc/src/CbcSolver.hpp](#)

## 7.108 CglTemporary Class Reference

Stored Temporary Cut Generator Class - destroyed after first use.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for CglTemporary:



## Public Member Functions

### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const CglTreeInfo info=CglTreeInfo())  
*Generate Mixed Integer Stored cuts for the model of the solver interface, si.*

### Constructors and destructors

- [CglTemporary](#) ()  
*Default constructor.*
- [CglTemporary](#) (const [CglTemporary](#) &rhs)  
*Copy constructor.*
- virtual CglCutGenerator \* [clone](#) () const  
*Clone.*
- [CglTemporary](#) & [operator=](#) (const [CglTemporary](#) &rhs)  
*Assignment operator.*
- virtual [~CglTemporary](#) ()  
*Destructor.*

## 7.108.1 Detailed Description

Stored Temporary Cut Generator Class - destroyed after first use.

Definition at line 1266 of file CbcLinked.hpp.

## 7.108.2 Constructor & Destructor Documentation

### 7.108.2.1 [CglTemporary::CglTemporary](#) ( )

Default constructor.

### 7.108.2.2 [CglTemporary::CglTemporary](#) ( const [CglTemporary](#) & rhs )

Copy constructor.

### 7.108.2.3 [virtual CglTemporary::~~CglTemporary](#) ( ) [virtual]

Destructor.

## 7.108.3 Member Function Documentation

### 7.108.3.1 [virtual void CglTemporary::generateCuts](#) ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]

Generate Mixed Integer Stored cuts for the model of the solver interface, si.

Insert the generated cuts into OsiCut, cs.

This generator just looks at previously stored cuts and inserts any that are violated by enough

**7.108.3.2** `virtual CglCutGenerator* CglTemporary::clone ( ) const [virtual]`

Clone.

**7.108.3.3** `CglTemporary& CglTemporary::operator= ( const CglTemporary & rhs )`

Assignment operator.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.109 CbcGenCtlBlk::chooseStrongCtl\_struct Struct Reference

Control variables for a strong branching method.

```
#include <CbcGenCtlBlk.hpp>
```

### Public Attributes

- int [numBeforeTrust\\_](#)
- int [numStrong\\_](#)
- int [shadowPriceMode\\_](#)

### 7.109.1 Detailed Description

Control variables for a strong branching method.

Consult OsiChooseVariable and [CbcModel](#) for details. An artifact of the changeover from CbcObjects to OsiObjects is that the number of uses before pseudo costs are trusted ([numBeforeTrust\\_](#)) and the number of variables evaluated with strong branching ([numStrong\\_](#)) are parameters of [CbcModel](#).

Definition at line 765 of file CbcGenCtlBlk.hpp.

### 7.109.2 Member Data Documentation

**7.109.2.1** `int CbcGenCtlBlk::chooseStrongCtl_struct::numBeforeTrust_`

Definition at line 766 of file CbcGenCtlBlk.hpp.

**7.109.2.2** `int CbcGenCtlBlk::chooseStrongCtl_struct::numStrong_`

Definition at line 767 of file CbcGenCtlBlk.hpp.

**7.109.2.3** `int CbcGenCtlBlk::chooseStrongCtl_struct::shadowPriceMode_`

Definition at line 768 of file CbcGenCtlBlk.hpp.

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

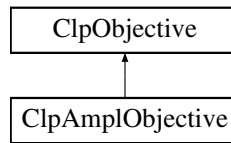
- [/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp](#)

## 7.110 ClpAmplObjective Class Reference

Ampl Objective Class.

```
#include <ClpAmplObjective.hpp>
```

Inheritance diagram for ClpAmplObjective:



### Public Member Functions

#### Stuff

- virtual double \* [gradient](#) (const ClpSimplex \*model, const double \*solution, double &offset, bool refresh, int includeLinear=2)  
*Returns gradient.*
- virtual double [reducedGradient](#) (ClpSimplex \*model, double \*region, bool useFeasibleCosts)  
*Resize objective.*
- virtual double [stepLength](#) (ClpSimplex \*model, const double \*solution, const double \*change, double maximumTheta, double &currentObj, double &predictedObj, double &thetaObj)  
*Returns step length which gives minimum of objective for solution + theta \* change vector up to maximum theta.*
- virtual double [objectiveValue](#) (const ClpSimplex \*model, const double \*solution) const  
*Return objective value (without any ClpModel offset) (model may be NULL)*
- virtual void [resize](#) (int newNumberColumns)
- virtual void [deleteSome](#) (int numberToDelete, const int \*which)  
*Delete columns in objective.*
- virtual void [reallyScale](#) (const double \*columnScale)  
*Scale objective.*
- virtual int [markNonlinear](#) (char \*which)  
*Given a zeroed array sets nonlinear columns to 1.*
- virtual void [newXValues](#) ()  
*Say we have new primal solution - so may need to recompute.*

#### Constructors and destructors

- [ClpAmplObjective](#) ()  
*Default Constructor.*
- [ClpAmplObjective](#) (void \*amplInfo)  
*Constructor from ampl info.*
- [ClpAmplObjective](#) (const [ClpAmplObjective](#) &rhs)  
*Copy constructor.*
- [ClpAmplObjective](#) & [operator=](#) (const [ClpAmplObjective](#) &rhs)  
*Assignment operator.*
- virtual [~ClpAmplObjective](#) ()  
*Destructor.*
- virtual ClpObjective \* [clone](#) () const  
*Clone.*

#### Gets and sets

- double \* [linearObjective](#) () const  
*Linear objective.*

## 7.110.1 Detailed Description

Ampl Objective Class.

Definition at line 18 of file ClpAmplObjective.hpp.

## 7.110.2 Constructor &amp; Destructor Documentation

## 7.110.2.1 ClpAmplObjective::ClpAmplObjective ( )

Default Constructor.

7.110.2.2 ClpAmplObjective::ClpAmplObjective ( void \* *amplInfo* )

Constructor from ampl info.

7.110.2.3 ClpAmplObjective::ClpAmplObjective ( const ClpAmplObjective & *rhs* )

Copy constructor .

## 7.110.2.4 virtual ClpAmplObjective::~~ClpAmplObjective ( ) [virtual]

Destructor.

## 7.110.3 Member Function Documentation

7.110.3.1 virtual double\* ClpAmplObjective::gradient ( const ClpSimplex \* *model*, const double \* *solution*, double & *offset*, bool *refresh*, int *includeLinear* = 2 ) [virtual]

Returns gradient.

If Ampl then solution may be NULL, also returns an offset (to be added to current one) If refresh is false then uses last solution Uses model for scaling includeLinear 0 - no, 1 as is, 2 as feasible

7.110.3.2 virtual double ClpAmplObjective::reducedGradient ( ClpSimplex \* *model*, double \* *region*, bool *useFeasibleCosts* ) [virtual]

Resize objective.

Returns reduced gradient.Returns an offset (to be added to current one).

7.110.3.3 virtual double ClpAmplObjective::stepLength ( ClpSimplex \* *model*, const double \* *solution*, const double \* *change*, double *maximumTheta*, double & *currentObj*, double & *predictedObj*, double & *thetaObj* ) [virtual]

Returns step length which gives minimum of objective for solution + theta \* change vector up to maximum theta.

arrays are numberColumns+numberRows Also sets current objective, predicted and at maximumTheta

7.110.3.4 virtual double ClpAmplObjective::objectiveValue ( const ClpSimplex \* *model*, const double \* *solution* ) const [virtual]

Return objective value (without any ClpModel offset) (model may be NULL)

7.110.3.5 virtual void ClpAmplObjective::resize ( int *newNumberColumns* ) [virtual]

7.110.3.6 `virtual void ClpAmplObjective::deleteSome ( int numberToDelete, const int * which ) [virtual]`

Delete columns in objective.

7.110.3.7 `virtual void ClpAmplObjective::reallyScale ( const double * columnScale ) [virtual]`

Scale objective.

7.110.3.8 `virtual int ClpAmplObjective::markNonlinear ( char * which ) [virtual]`

Given a zeroed array sets nonlinear columns to 1.

Returns number of nonlinear columns

7.110.3.9 `virtual void ClpAmplObjective::newXValues ( ) [virtual]`

Say we have new primal solution - so may need to recompute.

7.110.3.10 `ClpAmplObjective& ClpAmplObjective::operator= ( const ClpAmplObjective & rhs )`

Assignment operator.

7.110.3.11 `virtual ClpObjective* ClpAmplObjective::clone ( ) const [virtual]`

Clone.

7.110.3.12 `double* ClpAmplObjective::linearObjective ( ) const`

Linear objective.

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

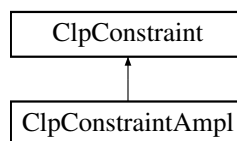
- </home/ted/COIN/trunk/Cbc/src/ClpAmplObjective.hpp>

## 7.111 ClpConstraintAmpl Class Reference

Ampl Constraint Class.

```
#include <ClpConstraintAmpl.hpp>
```

Inheritance diagram for ClpConstraintAmpl:



### Public Member Functions

#### Stuff

- virtual int [gradient](#) (const ClpSimplex \*model, const double \*solution, double \*gradient, double &functionValue, double &offset, bool useScaling=false, bool refresh=true) const  
*Fills gradient.*
- virtual void [resize](#) (int newNumberColumns)



- Resize constraint.*
- virtual void [deleteSome](#) (int numberToDelete, const int \*which)
- Delete columns in constraint.*
- virtual void [reallyScale](#) (const double \*columnScale)
- Scale constraint.*
- virtual int [markNonlinear](#) (char \*which) const
- Given a zeroed array sets nonampl columns to 1.*
- virtual int [markNonzero](#) (char \*which) const
- Given a zeroed array sets possible nonzero coefficients to 1.*
- virtual void [newXValues](#) ()
- Say we have new primal solution - so may need to recompute.*

### Constructors and destructors

- [ClpConstraintAmpl](#) ()
- Default Constructor.*
- [ClpConstraintAmpl](#) (int row, void \*amplInfo)
- Constructor from ampl.*
- [ClpConstraintAmpl](#) (const [ClpConstraintAmpl](#) &rhs)
- Copy constructor .*
- [ClpConstraintAmpl](#) & [operator=](#) (const [ClpConstraintAmpl](#) &rhs)
- Assignment operator.*
- virtual [~ClpConstraintAmpl](#) ()
- Destructor.*
- virtual [ClpConstraint](#) \* [clone](#) () const
- Clone.*

### Gets and sets

- virtual int [numberCoefficients](#) () const
- Number of coefficients.*
- const int \* [column](#) () const
- Columns.*
- const double \* [coefficient](#) () const
- Coefficients.*

#### 7.111.1 Detailed Description

Ampl Constraint Class.

Definition at line 17 of file [ClpConstraintAmpl.hpp](#).

#### 7.111.2 Constructor & Destructor Documentation

##### 7.111.2.1 [ClpConstraintAmpl::ClpConstraintAmpl](#) ( )

Default Constructor.

##### 7.111.2.2 [ClpConstraintAmpl::ClpConstraintAmpl](#) ( int row, void \* amplInfo )

Constructor from ampl.

##### 7.111.2.3 [ClpConstraintAmpl::ClpConstraintAmpl](#) ( const [ClpConstraintAmpl](#) & rhs )

Copy constructor .

7.111.2.4 `virtual ClpConstraintAmpl::~~ClpConstraintAmpl ( ) [virtual]`

Destructor.

### 7.111.3 Member Function Documentation

7.111.3.1 `virtual int ClpConstraintAmpl::gradient ( const ClpSimplex * model, const double * solution, double * gradient, double & functionValue, double & offset, bool useScaling = false, bool refresh = true ) const [virtual]`

Fills gradient.

If Ampl then solution may be NULL, also returns true value of function and offset so we can use x not deltaX in constraint  
If refresh is false then uses last solution Uses model for scaling Returns non-zero if gradient undefined at current solution

7.111.3.2 `virtual void ClpConstraintAmpl::resize ( int newNumberColumns ) [virtual]`

Resize constraint.

7.111.3.3 `virtual void ClpConstraintAmpl::deleteSome ( int numberToDelete, const int * which ) [virtual]`

Delete columns in constraint.

7.111.3.4 `virtual void ClpConstraintAmpl::reallyScale ( const double * columnScale ) [virtual]`

Scale constraint.

7.111.3.5 `virtual int ClpConstraintAmpl::markNonlinear ( char * which ) const [virtual]`

Given a zeroed array sets nonampl columns to 1.

Returns number of nonampl columns

7.111.3.6 `virtual int ClpConstraintAmpl::markNonzero ( char * which ) const [virtual]`

Given a zeroed array sets possible nonzero coefficients to 1.

Returns number of nonzeros

7.111.3.7 `virtual void ClpConstraintAmpl::newXValues ( ) [virtual]`

Say we have new primal solution - so may need to recompute.

7.111.3.8 `ClpConstraintAmpl& ClpConstraintAmpl::operator= ( const ClpConstraintAmpl & rhs )`

Assignment operator.

7.111.3.9 `virtual ClpConstraint* ClpConstraintAmpl::clone ( ) const [virtual]`

Clone.

7.111.3.10 `virtual int ClpConstraintAmpl::numberCoefficients ( ) const [virtual]`

Number of coefficients.

7.111.3.11 `const int* ClpConstraintAmpl::column ( ) const [inline]`

Columns.

Definition at line 83 of file ClpConstraintAmpl.hpp.

**7.111.3.12** `const double* ClpConstraintAmpl::coefficient ( ) const [inline]`

Coefficients.

Definition at line 87 of file ClpConstraintAmpl.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/ClpConstraintAmpl.hpp](#)

## 7.112 CoinHashLink Struct Reference

Really for Conflict cuts to - a) stop duplicates b) allow half baked cuts The whichRow\_ field in OsiRowCut2 is used for a type 0 - normal 1 - processed cut 2 - unprocessed cut i.e.

```
#include <CbcCountRowCut.hpp>
```

### Public Attributes

- int [index](#)
- int [next](#)

### 7.112.1 Detailed Description

Really for Conflict cuts to - a) stop duplicates b) allow half baked cuts The whichRow\_ field in OsiRowCut2 is used for a type 0 - normal 1 - processed cut 2 - unprocessed cut i.e.

dual ray computation

Definition at line 131 of file CbcCountRowCut.hpp.

### 7.112.2 Member Data Documentation

#### 7.112.2.1 int CoinHashLink::index

Definition at line 132 of file CbcCountRowCut.hpp.

#### 7.112.2.2 int CoinHashLink::next

Definition at line 132 of file CbcCountRowCut.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcCountRowCut.hpp](#)

## 7.113 CbcGenCtlBlk::debugSolInfo\_struct Struct Reference

Array of primal variable values for debugging.

```
#include <CbcGenCtlBlk.hpp>
```

## Public Attributes

- int [numCols\\_](#)
- double \* [values\\_](#)

### 7.113.1 Detailed Description

Array of primal variable values for debugging.

Used to provide a known optimal solution to `activateRowCutDebugger()`.

Definition at line 669 of file `CbcGenCtlBlk.hpp`.

### 7.113.2 Member Data Documentation

#### 7.113.2.1 int CbcGenCtlBlk::debugSolInfo\_struct::numCols\_

Definition at line 670 of file `CbcGenCtlBlk.hpp`.

#### 7.113.2.2 double\* CbcGenCtlBlk::debugSolInfo\_struct::values\_

Definition at line 671 of file `CbcGenCtlBlk.hpp`.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp](#)

## 7.114 CbcGenCtlBlk::djFixCtl\_struct Struct Reference

Control use of reduced cost fixing prior to B&C.

```
#include <CbcGenCtlBlk.hpp>
```

## Public Attributes

- bool [action\\_](#)
- double [threshold\\_](#)

### 7.114.1 Detailed Description

Control use of reduced cost fixing prior to B&C.

This heuristic fixes variables whose reduced cost for the root relaxation exceeds the specified threshold. This is purely a heuristic, performed before there's any incumbent solution. It may well fix variables at the wrong bound!

Definition at line 739 of file `CbcGenCtlBlk.hpp`.

### 7.114.2 Member Data Documentation

#### 7.114.2.1 bool CbcGenCtlBlk::djFixCtl\_struct::action\_

Definition at line 740 of file `CbcGenCtlBlk.hpp`.

#### 7.114.2.2 double CbcGenCtlBlk::djFixCtl\_struct::threshold\_

Definition at line 741 of file CbcGenCtlBlk.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp](#)

### 7.115 CbcGenCtlBlk::genParamsInfo\_struct Struct Reference

Start and end of cbc-generic parameters in parameter vector.

```
#include <CbcGenCtlBlk.hpp>
```

#### Public Attributes

- int [first\\_](#)
- int [last\\_](#)

#### 7.115.1 Detailed Description

Start and end of cbc-generic parameters in parameter vector.

Definition at line 598 of file CbcGenCtlBlk.hpp.

#### 7.115.2 Member Data Documentation

##### 7.115.2.1 int CbcGenCtlBlk::genParamsInfo\_struct::first\_

Definition at line 599 of file CbcGenCtlBlk.hpp.

##### 7.115.2.2 int CbcGenCtlBlk::genParamsInfo\_struct::last\_

Definition at line 600 of file CbcGenCtlBlk.hpp.

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

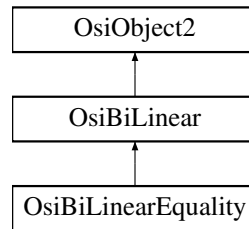
- [/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp](#)

### 7.116 OsiBiLinear Class Reference

Define BiLinear objects.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for OsiBiLinear:



### Public Member Functions

- [OsiBiLinear](#) ()
- [OsiBiLinear](#) (OsiSolverInterface \*solver, int [xColumn](#), int [yColumn](#), int [xyRow](#), double [coefficient](#), double xMesh, double yMesh, int numberExistingObjects=0, const OsiObject \*\*objects=NULL)
 

*Useful constructor - This Adds in rows and variables to construct valid Linked Ordered Set Adds extra constraints to match other x/y So note not const solver.*
- [OsiBiLinear](#) (CoinModel \*coinModel, int [xColumn](#), int [yColumn](#), int [xyRow](#), double [coefficient](#), double xMesh, double yMesh, int numberExistingObjects=0, const OsiObject \*\*objects=NULL)
 

*Useful constructor - This Adds in rows and variables to construct valid Linked Ordered Set Adds extra constraints to match other x/y So note not const model.*
- [OsiBiLinear](#) (const [OsiBiLinear](#) &)
- virtual OsiObject \* [clone](#) () const
 

*Clone.*
- [OsiBiLinear](#) & [operator=](#) (const [OsiBiLinear](#) &rhs)
- virtual [~OsiBiLinear](#) ()
- virtual double [infeasibility](#) (const OsiBranchingInformation \*info, int &whichWay) const
 

*Infeasibility - large is 0.5.*
- virtual double [feasibleRegion](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info) const
 

*Set bounds to fix the variable at the current (integer) value.*
- virtual OsiBranchingObject \* [createBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way) const
 

*Creates a branching object.*
- virtual void [resetSequenceEtc](#) (int numberColumns, const int \*originalColumns)
 

*Redoes data when sequence numbers change.*
- virtual double [checkInfeasibility](#) (const OsiBranchingInformation \*info) const
- virtual bool [canDoHeuristics](#) () const
 

*Return true if object can take part in normal heuristics.*
- virtual bool [boundBranch](#) () const
 

*Return true if branch should only bound variables.*
- int [xColumn](#) () const
 

*X column.*
- int [yColumn](#) () const
 

*Y column.*
- int [xRow](#) () const
 

*X row.*
- int [yRow](#) () const
 

*Y row.*
- int [xyRow](#) () const
 

*XY row.*

- double [coefficient](#) () const  
*Coefficient.*
- void [setCoefficient](#) (double value)  
*Set coefficient.*
- int [firstLambda](#) () const  
*First lambda (of 4)*
- double [xSatisfied](#) () const  
*X satisfied if less than this away from mesh.*
- void [setXSatisfied](#) (double value)
- double [ySatisfied](#) () const  
*Y satisfied if less than this away from mesh.*
- void [setYSatisfied](#) (double value)
- double [xOtherSatisfied](#) () const  
*X other satisfied if less than this away from mesh.*
- void [setXOtherSatisfied](#) (double value)
- double [yOtherSatisfied](#) () const  
*Y other satisfied if less than this away from mesh.*
- void [setYOtherSatisfied](#) (double value)
- double [xMeshSize](#) () const  
*X meshSize.*
- void [setXMeshSize](#) (double value)
- double [yMeshSize](#) () const  
*Y meshSize.*
- void [setYMeshSize](#) (double value)
- double [xySatisfied](#) () const  
*XY satisfied if two version differ by less than this.*
- void [setXYSatisfied](#) (double value)
- void [setMeshSizes](#) (const OsiSolverInterface \*solver, double x, double y)  
*Set sizes and other stuff.*
- int [branchingStrategy](#) () const  
*branching strategy etc bottom 2 bits 0 branch on either, 1 branch on x, 2 branch on y next bit 4 set to say don't update coefficients next bit 8 set to say don't use in feasible region next bit 16 set to say - Always satisfied !!*
- void [setBranchingStrategy](#) (int value)
- int [boundType](#) () const  
*Simple quadratic bound marker.*
- void [setBoundType](#) (int value)
- void [newBounds](#) (OsiSolverInterface \*solver, int way, short xOrY, double separator) const  
*Does work of branching.*
- int [updateCoefficients](#) (const double \*lower, const double \*upper, double \*objective, CoinPackedMatrix \*matrix, CoinWarmStartBasis \*basis) const  
*Updates coefficients - returns number updated.*
- double [xyCoefficient](#) (const double \*solution) const  
*Returns true value of single xyRow coefficient.*
- void [getCoefficients](#) (const OsiSolverInterface \*solver, double xB[2], double yB[2], double xybar[4]) const  
*Get LU coefficients from matrix.*
- double [computeLambdas](#) (const double xB[3], const double yB[3], const double xybar[4], double lambda[4]) const  
*Compute lambdas (third entry in each .B is current value) (nonzero if bad)*
- void [addExtraRow](#) (int row, double multiplier)

*Adds in data for extra row with variable coefficients.*

- void [getPseudoShadow](#) (const OsiBranchingInformation \*info)

*Sets infeasibility and other when pseudo shadow prices.*

- double [getMovement](#) (const OsiBranchingInformation \*info)

*Gets sum of movements to correct value.*

### Protected Member Functions

- void [computeLambdas](#) (const OsiSolverInterface \*solver, double lambda[4]) const

*Compute lambdas if coefficients not changing.*

### Protected Attributes

- double [coefficient\\_](#)  
*data*
- double [xMeshSize\\_](#)  
*x mesh*
- double [yMeshSize\\_](#)  
*y mesh*
- double [xSatisfied\\_](#)  
*x satisfied if less than this away from mesh*
- double [ySatisfied\\_](#)  
*y satisfied if less than this away from mesh*
- double [xOtherSatisfied\\_](#)  
*X other satisfied if less than this away from mesh.*
- double [yOtherSatisfied\\_](#)  
*Y other satisfied if less than this away from mesh.*
- double [xySatisfied\\_](#)  
*xy satisfied if less than this away from true*
- double [xyBranchValue\\_](#)  
*value of x or y to branch about*
- int [xColumn\\_](#)  
*x column*
- int [yColumn\\_](#)  
*y column*
- int [firstLambda\\_](#)  
*First lambda (of 4)*
- int [branchingStrategy\\_](#)  
*branching strategy etc bottom 2 bits 0 branch on either, 1 branch on x, 2 branch on y next bit 4 set to say don't update coefficients next bit 8 set to say don't use in feasible region next bit 16 set to say - Always satisfied !!*
- int [boundType\\_](#)  
*Simple quadratic bound marker.*
- int [xRow\\_](#)  
*x row*
- int [yRow\\_](#)  
*y row (-1 if x\*x)*
- int [xyRow\\_](#)



- Output row.*
- int [convexity\\_](#)  
*Convexity row.*
- int [numberExtraRows\\_](#)  
*Number of extra rows (coefficients to be modified)*
- double \* [multiplier\\_](#)  
*Multiplier for coefficient on row.*
- int \* [extraRow\\_](#)  
*Row number.*
- short [chosen\\_](#)  
*Which chosen -1 none, 0 x, 1 y.*

### 7.116.1 Detailed Description

Define BiLinear objects.

This models  $x*y$  where one or both are integer

Definition at line 720 of file CbcLinked.hpp.

### 7.116.2 Constructor & Destructor Documentation

#### 7.116.2.1 OsiBiLinear::OsiBiLinear ( )

**7.116.2.2** `OsiBiLinear::OsiBiLinear ( OsiSolverInterface * solver, int xColumn, int yColumn, int xyRow, double coefficient, double xMesh, double yMesh, int numberExistingObjects = 0, const OsiObject ** objects = NULL )`

Useful constructor - This Adds in rows and variables to construct valid Linked Ordered Set Adds extra constraints to match other x/y So note not const solver.

**7.116.2.3** `OsiBiLinear::OsiBiLinear ( CoinModel * coinModel, int xColumn, int yColumn, int xyRow, double coefficient, double xMesh, double yMesh, int numberExistingObjects = 0, const OsiObject ** objects = NULL )`

Useful constructor - This Adds in rows and variables to construct valid Linked Ordered Set Adds extra constraints to match other x/y So note not const model.

#### 7.116.2.4 OsiBiLinear::OsiBiLinear ( const OsiBiLinear & )

#### 7.116.2.5 virtual OsiBiLinear::~~OsiBiLinear ( ) [virtual]

### 7.116.3 Member Function Documentation

#### 7.116.3.1 virtual OsiObject\* OsiBiLinear::clone ( ) const [virtual]

Clone.

Reimplemented in [OsiBiLinearEquality](#).

#### 7.116.3.2 OsiBiLinear& OsiBiLinear::operator= ( const OsiBiLinear & rhs )

#### 7.116.3.3 virtual double OsiBiLinear::infeasibility ( const OsiBranchingInformation \* info, int & whichWay ) const [virtual]

Infeasibility - large is 0.5.

7.116.3.4 `virtual double OsiBiLinear::feasibleRegion ( OsiSolverInterface * solver, const OsiBranchingInformation * info ) const` `[virtual]`

Set bounds to fix the variable at the current (integer) value.

Given an integer value, set the lower and upper bounds to fix the variable. Returns amount it had to move variable.

7.116.3.5 `virtual OsiBranchingObject* OsiBiLinear::createBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way ) const` `[virtual]`

Creates a branching object.

The preferred direction is set by *way*, 0 for down, 1 for up.

7.116.3.6 `virtual void OsiBiLinear::resetSequenceEtc ( int numberColumns, const int * originalColumns )` `[virtual]`

Redoes data when sequence numbers change.

7.116.3.7 `virtual double OsiBiLinear::checkInfeasibility ( const OsiBranchingInformation * info ) const` `[virtual]`

7.116.3.8 `virtual bool OsiBiLinear::canDoHeuristics ( ) const` `[inline],[virtual]`

Return true if object can take part in normal heuristics.

Definition at line 785 of file CbcLinked.hpp.

7.116.3.9 `virtual bool OsiBiLinear::boundBranch ( ) const` `[inline],[virtual]`

Return true if branch should only bound variables.

Definition at line 790 of file CbcLinked.hpp.

7.116.3.10 `int OsiBiLinear::xColumn ( ) const` `[inline]`

X column.

Definition at line 794 of file CbcLinked.hpp.

7.116.3.11 `int OsiBiLinear::yColumn ( ) const` `[inline]`

Y column.

Definition at line 798 of file CbcLinked.hpp.

7.116.3.12 `int OsiBiLinear::xRow ( ) const` `[inline]`

X row.

Definition at line 802 of file CbcLinked.hpp.

7.116.3.13 `int OsiBiLinear::yRow ( ) const` `[inline]`

Y row.

Definition at line 806 of file CbcLinked.hpp.

7.116.3.14 `int OsiBiLinear::xyRow ( ) const` `[inline]`

XY row.

Definition at line 810 of file CbcLinked.hpp.

**7.116.3.15** `double OsiBiLinear::coefficient ( ) const [inline]`

Coefficient.

Definition at line 814 of file CbcLinked.hpp.

**7.116.3.16** `void OsiBiLinear::setCoefficient ( double value ) [inline]`

Set coefficient.

Definition at line 818 of file CbcLinked.hpp.

**7.116.3.17** `int OsiBiLinear::firstLambda ( ) const [inline]`

First lambda (of 4)

Definition at line 822 of file CbcLinked.hpp.

**7.116.3.18** `double OsiBiLinear::xSatisfied ( ) const [inline]`

X satisfied if less than this away from mesh.

Definition at line 826 of file CbcLinked.hpp.

**7.116.3.19** `void OsiBiLinear::setXSatisfied ( double value ) [inline]`

Definition at line 829 of file CbcLinked.hpp.

**7.116.3.20** `double OsiBiLinear::ySatisfied ( ) const [inline]`

Y satisfied if less than this away from mesh.

Definition at line 833 of file CbcLinked.hpp.

**7.116.3.21** `void OsiBiLinear::setYSatisfied ( double value ) [inline]`

Definition at line 836 of file CbcLinked.hpp.

**7.116.3.22** `double OsiBiLinear::xOtherSatisfied ( ) const [inline]`

X other satisfied if less than this away from mesh.

Definition at line 840 of file CbcLinked.hpp.

**7.116.3.23** `void OsiBiLinear::setXOtherSatisfied ( double value ) [inline]`

Definition at line 843 of file CbcLinked.hpp.

**7.116.3.24** `double OsiBiLinear::yOtherSatisfied ( ) const [inline]`

Y other satisfied if less than this away from mesh.

Definition at line 847 of file CbcLinked.hpp.

**7.116.3.25** `void OsiBiLinear::setYOtherSatisfied ( double value ) [inline]`

Definition at line 850 of file CbcLinked.hpp.

**7.116.3.26** `double OsiBiLinear::xMeshSize ( ) const [inline]`

X meshSize.

Definition at line 854 of file CbcLinked.hpp.

**7.116.3.27** `void OsiBiLinear::setXMeshSize ( double value ) [inline]`

Definition at line 857 of file CbcLinked.hpp.

**7.116.3.28** `double OsiBiLinear::yMeshSize ( ) const [inline]`

Y meshSize.

Definition at line 861 of file CbcLinked.hpp.

**7.116.3.29** `void OsiBiLinear::setYMeshSize ( double value ) [inline]`

Definition at line 864 of file CbcLinked.hpp.

**7.116.3.30** `double OsiBiLinear::xySatisfied ( ) const [inline]`

XY satisfied if two version differ by less than this.

Definition at line 868 of file CbcLinked.hpp.

**7.116.3.31** `void OsiBiLinear::setXYSatisfied ( double value ) [inline]`

Definition at line 871 of file CbcLinked.hpp.

**7.116.3.32** `void OsiBiLinear::setMeshSizes ( const OsiSolverInterface * solver, double x, double y )`

Set sizes and other stuff.

**7.116.3.33** `int OsiBiLinear::branchingStrategy ( ) const [inline]`

branching strategy etc bottom 2 bits 0 branch on either, 1 branch on x, 2 branch on y next bit 4 set to say don't update coefficients next bit 8 set to say don't use in feasible region next bit 16 set to say - Always satisfied !!

Definition at line 886 of file CbcLinked.hpp.

**7.116.3.34** `void OsiBiLinear::setBranchingStrategy ( int value ) [inline]`

Definition at line 889 of file CbcLinked.hpp.

**7.116.3.35** `int OsiBiLinear::boundType ( ) const [inline]`

Simple quadratic bound marker.

0 no 1 L if coefficient pos, G if negative i.e. value is ub on xy 2 G if coefficient pos, L if negative i.e. value is lb on xy 3 E  
If bound then real coefficient is 1.0 and coefficient\_ is bound

Definition at line 899 of file CbcLinked.hpp.

**7.116.3.36** `void OsiBiLinear::setBoundType ( int value ) [inline]`

Definition at line 902 of file CbcLinked.hpp.

**7.116.3.37** `void OsiBiLinear::newBounds ( OsiSolverInterface * solver, int way, short xOrY, double separator ) const`

Does work of branching.

7.116.3.38 `int OsiBiLinear::updateCoefficients ( const double * lower, const double * upper, double * objective, CoinPackedMatrix * matrix, CoinWarmStartBasis * basis ) const`

Updates coefficients - returns number updated.

7.116.3.39 `double OsiBiLinear::xyCoefficient ( const double * solution ) const`

Returns true value of single xyRow coefficient.

7.116.3.40 `void OsiBiLinear::getCoefficients ( const OsiSolverInterface * solver, double xB[2], double yB[2], double xybar[4] ) const`

Get LU coefficients from matrix.

7.116.3.41 `double OsiBiLinear::computeLambdas ( const double xB[3], const double yB[3], const double xybar[4], double lambda[4] ) const`

Compute lambdas (third entry in each .B is current value) (nonzero if bad)

7.116.3.42 `void OsiBiLinear::addExtraRow ( int row, double multiplier )`

Adds in data for extra row with variable coefficients.

7.116.3.43 `void OsiBiLinear::getPseudoShadow ( const OsiBranchingInformation * info )`

Sets infeasibility and other when pseudo shadow prices.

7.116.3.44 `double OsiBiLinear::getMovement ( const OsiBranchingInformation * info )`

Gets sum of movements to correct value.

7.116.3.45 `void OsiBiLinear::computeLambdas ( const OsiSolverInterface * solver, double lambda[4] ) const` [protected]

Compute lambdas if coefficients not changing.

#### 7.116.4 Member Data Documentation

7.116.4.1 `double OsiBiLinear::coefficient_` [protected]

data

Coefficient

Definition at line 929 of file CbcLinked.hpp.

7.116.4.2 `double OsiBiLinear::xMeshSize_` [protected]

x mesh

Definition at line 931 of file CbcLinked.hpp.

7.116.4.3 `double OsiBiLinear::yMeshSize_` [protected]

y mesh

Definition at line 933 of file CbcLinked.hpp.

**7.116.4.4 double OsiBiLinear::xSatisfied\_ [protected]**

x satisfied if less than this away from mesh

Definition at line 935 of file CbcLinked.hpp.

**7.116.4.5 double OsiBiLinear::ySatisfied\_ [protected]**

y satisfied if less than this away from mesh

Definition at line 937 of file CbcLinked.hpp.

**7.116.4.6 double OsiBiLinear::xOtherSatisfied\_ [protected]**

X other satisfied if less than this away from mesh.

Definition at line 939 of file CbcLinked.hpp.

**7.116.4.7 double OsiBiLinear::yOtherSatisfied\_ [protected]**

Y other satisfied if less than this away from mesh.

Definition at line 941 of file CbcLinked.hpp.

**7.116.4.8 double OsiBiLinear::xySatisfied\_ [protected]**

xy satisfied if less than this away from true

Definition at line 943 of file CbcLinked.hpp.

**7.116.4.9 double OsiBiLinear::xyBranchValue\_ [mutable],[protected]**

value of x or y to branch about

Definition at line 945 of file CbcLinked.hpp.

**7.116.4.10 int OsiBiLinear::xColumn\_ [protected]**

x column

Definition at line 947 of file CbcLinked.hpp.

**7.116.4.11 int OsiBiLinear::yColumn\_ [protected]**

y column

Definition at line 949 of file CbcLinked.hpp.

**7.116.4.12 int OsiBiLinear::firstLambda\_ [protected]**

First lambda (of 4)

Definition at line 951 of file CbcLinked.hpp.

**7.116.4.13 int OsiBiLinear::branchingStrategy\_ [protected]**

branching strategy etc bottom 2 bits 0 branch on either, 1 branch on x, 2 branch on y next bit 4 set to say don't update coefficients next bit 8 set to say don't use in feasible region next bit 16 set to say - Always satisfied !!

Definition at line 962 of file CbcLinked.hpp.

**7.116.4.14** `int OsiBiLinear::boundType_` `[protected]`

Simple quadratic bound marker.

0 no 1 L if coefficient pos, G if negative i.e. value is ub on xy 2 G if coefficient pos, L if negative i.e. value is lb on xy 3 E  
If bound then real coefficient is 1.0 and coefficient\_ is bound

Definition at line 970 of file CbcLinked.hpp.

**7.116.4.15** `int OsiBiLinear::xRow_` `[protected]`

x row

Definition at line 972 of file CbcLinked.hpp.

**7.116.4.16** `int OsiBiLinear::yRow_` `[protected]`

y row (-1 if x\*x)

Definition at line 974 of file CbcLinked.hpp.

**7.116.4.17** `int OsiBiLinear::xyRow_` `[protected]`

Output row.

Definition at line 976 of file CbcLinked.hpp.

**7.116.4.18** `int OsiBiLinear::convexity_` `[protected]`

Convexity row.

Definition at line 978 of file CbcLinked.hpp.

**7.116.4.19** `int OsiBiLinear::numberExtraRows_` `[protected]`

Number of extra rows (coefficients to be modified)

Definition at line 980 of file CbcLinked.hpp.

**7.116.4.20** `double* OsiBiLinear::multiplier_` `[protected]`

Multiplier for coefficient on row.

Definition at line 982 of file CbcLinked.hpp.

**7.116.4.21** `int* OsiBiLinear::extraRow_` `[protected]`

Row number.

Definition at line 984 of file CbcLinked.hpp.

**7.116.4.22** `short OsiBiLinear::chosen_` `[mutable], [protected]`

Which chosen -1 none, 0 x, 1 y.

Definition at line 986 of file CbcLinked.hpp.

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

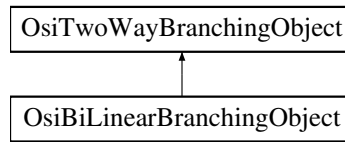
- </home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp>

### 7.117 OsiBiLinearBranchingObject Class Reference

Branching object for BiLinear objects.

```
#include <CbCLinked.hpp>
```

Inheritance diagram for OsiBiLinearBranchingObject:



#### Public Member Functions

- [OsiBiLinearBranchingObject](#) ()
- [OsiBiLinearBranchingObject](#) (OsiSolverInterface \*solver, const [OsiBiLinear](#) \*originalObject, int way, double separator, int chosen)
- [OsiBiLinearBranchingObject](#) (const [OsiBiLinearBranchingObject](#) &)
- [OsiBiLinearBranchingObject](#) & operator= (const [OsiBiLinearBranchingObject](#) &rhs)
- virtual OsiBranchingObject \* [clone](#) () const  
*Clone.*
- virtual [~OsiBiLinearBranchingObject](#) ()
- virtual double [branch](#) (OsiSolverInterface \*solver)  
*Does next branch and updates state.*
- virtual void [print](#) (const OsiSolverInterface \*solver=NULL)  
*Print something about branch - only if log level high.*
- virtual bool [boundBranch](#) () const  
*Return true if branch should only bound variables.*

#### 7.117.1 Detailed Description

Branching object for BiLinear objects.

Definition at line 991 of file CbcLinked.hpp.

#### 7.117.2 Constructor & Destructor Documentation

7.117.2.1 [OsiBiLinearBranchingObject::OsiBiLinearBranchingObject](#) ( )

7.117.2.2 [OsiBiLinearBranchingObject::OsiBiLinearBranchingObject](#) ( OsiSolverInterface \* solver, const [OsiBiLinear](#) \* originalObject, int way, double separator, int chosen )

7.117.2.3 [OsiBiLinearBranchingObject::OsiBiLinearBranchingObject](#) ( const [OsiBiLinearBranchingObject](#) & )

7.117.2.4 [virtual OsiBiLinearBranchingObject::~~OsiBiLinearBranchingObject](#) ( ) [virtual]

#### 7.117.3 Member Function Documentation

7.117.3.1 [OsiBiLinearBranchingObject& OsiBiLinearBranchingObject::operator=](#) ( const [OsiBiLinearBranchingObject](#) & rhs )



7.117.3.2 `virtual OsiBranchingObject* OsiBiLinearBranchingObject::clone ( ) const` [virtual]

Clone.

7.117.3.3 `virtual double OsiBiLinearBranchingObject::branch ( OsiSolverInterface * solver )` [virtual]

Does next branch and updates state.

7.117.3.4 `virtual void OsiBiLinearBranchingObject::print ( const OsiSolverInterface * solver = NULL )` [virtual]

Print something about branch - only if log level high.

7.117.3.5 `virtual bool OsiBiLinearBranchingObject::boundBranch ( ) const` [virtual]

Return true if branch should only bound variables.

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

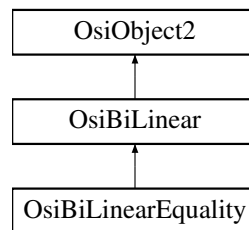
- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.118 OsiBiLinearEquality Class Reference

Define Continuous BiLinear objects for an == bound.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for OsiBiLinearEquality:



### Public Member Functions

- [OsiBiLinearEquality \( \)](#)
- [OsiBiLinearEquality \(OsiSolverInterface \\*solver, int xColumn, int yColumn, int xyRow, double rhs, double xMesh\)](#)  
*Useful constructor - This Adds in rows and variables to construct Ordered Set for  $x*y = b$  So note not const solver.*
- [OsiBiLinearEquality \(const OsiBiLinearEquality &\)](#)
- `virtual OsiObject * clone ( ) const`  
*Clone.*
- [OsiBiLinearEquality & operator= \(const OsiBiLinearEquality &rhs\)](#)
- `virtual ~OsiBiLinearEquality ( )`
- `virtual double improvement (const OsiSolverInterface *solver) const`  
*Possible improvement.*
- `double newGrid (OsiSolverInterface *solver, int type) const`  
*change grid if type 0 then use solution and make finer if 1 then back to original returns mesh size*
- `int numberPoints ( ) const`  
*Number of points.*
- `void setNumberPoints (int value)`

## Additional Inherited Members

### 7.118.1 Detailed Description

Define Continuous BiLinear objects for an == bound.

This models  $x*y = b$  where both are continuous

Definition at line 1038 of file CbcLinked.hpp.

### 7.118.2 Constructor & Destructor Documentation

#### 7.118.2.1 OsiBiLinearEquality::OsiBiLinearEquality ( )

#### 7.118.2.2 OsiBiLinearEquality::OsiBiLinearEquality ( OsiSolverInterface \* *solver*, int *xColumn*, int *yColumn*, int *xyRow*, double *rhs*, double *xMesh* )

Useful constructor - This Adds in rows and variables to construct Ordered Set for  $x*y = b$  So note not const solver.

#### 7.118.2.3 OsiBiLinearEquality::OsiBiLinearEquality ( const OsiBiLinearEquality & )

#### 7.118.2.4 virtual OsiBiLinearEquality::~OsiBiLinearEquality ( ) [virtual]

### 7.118.3 Member Function Documentation

#### 7.118.3.1 virtual OsiObject\* OsiBiLinearEquality::clone ( ) const [virtual]

Clone.

Reimplemented from [OsiBiLinear](#).

#### 7.118.3.2 OsiBiLinearEquality& OsiBiLinearEquality::operator= ( const OsiBiLinearEquality & *rhs* )

#### 7.118.3.3 virtual double OsiBiLinearEquality::improvement ( const OsiSolverInterface \* *solver* ) const [virtual]

Possible improvement.

#### 7.118.3.4 double OsiBiLinearEquality::newGrid ( OsiSolverInterface \* *solver*, int *type* ) const

change grid if type 0 then use solution and make finer if 1 then back to original returns mesh size

#### 7.118.3.5 int OsiBiLinearEquality::numberPoints ( ) const [inline]

Number of points.

Definition at line 1075 of file CbcLinked.hpp.

#### 7.118.3.6 void OsiBiLinearEquality::setNumberPoints ( int *value* ) [inline]

Definition at line 1078 of file CbcLinked.hpp.

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

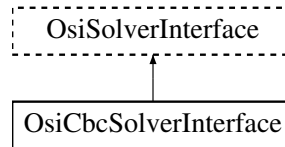
- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.119 OsiCbcSolverInterface Class Reference

Cbc Solver Interface.

```
#include <OsiCbcSolverInterface.hpp>
```

Inheritance diagram for OsiCbcSolverInterface:



### Public Member Functions

- virtual void [setObjSense](#) (double s)  
*Set objective function sense (1 for min (default), -1 for max.)*
- virtual void [setColSolution](#) (const double \*colsol)  
*Set the primal solution column values.*
- virtual void [setRowPrice](#) (const double \*rowprice)  
*Set dual solution vector.*

### Solve methods

- virtual void [initialSolve](#) ()  
*Solve initial LP relaxation.*
- virtual void [resolve](#) ()  
*Resolve an LP relaxation after problem modification.*
- virtual void [branchAndBound](#) ()  
*Invoke solver's built-in enumeration algorithm.*

### Parameter set/get methods

*The set methods return true if the parameter was set to the given value, false otherwise.*

*There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the cbc algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.*

*The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.*

- bool [setIntParam](#) (OsiIntParam key, int value)
- bool [setDbfParam](#) (OsiDbfParam key, double value)
- bool [setStrParam](#) (OsiStrParam key, const std::string &value)
- bool [getIntParam](#) (OsiIntParam key, int &value) const
- bool [getDbfParam](#) (OsiDbfParam key, double &value) const
- bool [getStrParam](#) (OsiStrParam key, std::string &value) const
- virtual bool [setHintParam](#) (OsiHintParam key, bool yesNo=true, OsiHintStrength strength=OsiHintTry, void \*otherInformation=NULL)
- virtual bool [getHintParam](#) (OsiHintParam key, bool &yesNo, OsiHintStrength &strength, void \*&otherInformation) const  
*Get a hint parameter.*
- virtual bool [getHintParam](#) (OsiHintParam key, bool &yesNo, OsiHintStrength &strength) const

*Get a hint parameter.*

### Methods returning info on how the solution process terminated

- virtual bool [isAbandoned](#) () const  
*Are there a numerical difficulties?*
- virtual bool [isProvenOptimal](#) () const  
*Is optimality proven?*
- virtual bool [isProvenPrimalInfeasible](#) () const  
*Is primal infeasibility proven?*
- virtual bool [isProvenDualInfeasible](#) () const  
*Is dual infeasibility proven?*
- virtual bool [isPrimalObjectiveLimitReached](#) () const  
*Is the given primal objective limit reached?*
- virtual bool [isDualObjectiveLimitReached](#) () const  
*Is the given dual objective limit reached?*
- virtual bool [isIterationLimitReached](#) () const  
*Iteration limit reached?*

### WarmStart related methods

- virtual CoinWarmStart \* [getEmptyWarmStart](#) () const  
*Get an empty warm start object.*
- virtual CoinWarmStart \* [getWarmStart](#) () const  
*Get warmstarting information.*
- virtual bool [setWarmStart](#) (const CoinWarmStart \*warmstart)  
*Set warmstarting information.*

### Hotstart related methods (primarily used in strong branching). <br>

*The user can create a hotstart (a snapshot) of the optimization process then reoptimize over and over again always starting from there.*

**NOTE:** *between hotstarted optimizations only bound changes are allowed.*

- virtual void [markHotStart](#) ()  
*Create a hotstart point of the optimization process.*
- virtual void [solveFromHotStart](#) ()  
*Optimize starting from the hotstart.*
- virtual void [unmarkHotStart](#) ()  
*Delete the snapshot.*

### Methods related to querying the input data

- virtual int [getNumCols](#) () const  
*Get number of columns.*
- virtual int [getNumRows](#) () const  
*Get number of rows.*
- virtual int [getNumElements](#) () const  
*Get number of nonzero elements.*
- virtual const double \* [getColLower](#) () const  
*Get pointer to array[getNumCols()] of column lower bounds.*
- virtual const double \* [getColUpper](#) () const  
*Get pointer to array[getNumCols()] of column upper bounds.*
- virtual const char \* [getRowSense](#) () const  
*Get pointer to array[getNumRows()] of row constraint senses.*

- virtual const double \* [getRightHandSide](#) () const  
*Get pointer to array[getNumRows()] of rows right-hand sides.*
- virtual const double \* [getRowRange](#) () const  
*Get pointer to array[getNumRows()] of row ranges.*
- virtual const double \* [getRowLower](#) () const  
*Get pointer to array[getNumRows()] of row lower bounds.*
- virtual const double \* [getRowUpper](#) () const  
*Get pointer to array[getNumRows()] of row upper bounds.*
- virtual const double \* [getObjCoefficients](#) () const  
*Get pointer to array[getNumCols()] of objective function coefficients.*
- virtual double [getObjSense](#) () const  
*Get objective function sense (1 for min (default), -1 for max)*
- virtual bool [isContinuous](#) (int colNumber) const  
*Return true if column is continuous.*
- virtual const CoinPackedMatrix \* [getMatrixByRow](#) () const  
*Get pointer to row-wise copy of matrix.*
- virtual const CoinPackedMatrix \* [getMatrixByCol](#) () const  
*Get pointer to column-wise copy of matrix.*
- virtual double [getInfinity](#) () const  
*Get solver's value for infinity.*

#### Methods related to querying the solution

- virtual const double \* [getColSolution](#) () const  
*Get pointer to array[getNumCols()] of primal solution vector.*
- virtual const double \* [getRowPrice](#) () const  
*Get pointer to array[getNumRows()] of dual prices.*
- virtual const double \* [getReducedCost](#) () const  
*Get a pointer to array[getNumCols()] of reduced costs.*
- virtual const double \* [getRowActivity](#) () const  
*Get pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).*
- virtual double [getObjValue](#) () const  
*Get objective function value.*
- virtual int [getIterationCount](#) () const  
*Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver).*
- virtual std::vector< double \* > [getDualRays](#) (int maxNumRays, bool fullRay=false) const  
*Get as many dual rays as the solver can provide.*
- virtual std::vector< double \* > [getPrimalRays](#) (int maxNumRays) const  
*Get as many primal rays as the solver can provide.*

#### Methods for row and column names.

Because OsiCbc is a pass-through class, it's necessary to override any virtual method in order to be sure we catch an override by the underlying solver.

See the OsiSolverInterface class documentation for detailed descriptions.

- virtual std::string [dfmtRowColName](#) (char rc, int ndx, unsigned digits=7) const  
*Generate a standard name of the form Rnnnnnnn or Cnnnnnnn.*
- virtual std::string [getObjName](#) (unsigned maxLen=std::string::npos) const  
*Return the name of the objective function.*
- virtual void [setObjName](#) (std::string name)  
*Set the name of the objective function.*
- virtual std::string [getRowName](#) (int rowIndex, unsigned maxLen=std::string::npos) const  
*Return the name of the row.*
- virtual const OsiNameVec & [getRowNames](#) ()

- Return a pointer to a vector of row names.*
- virtual void [setRowName](#) (int ndx, std::string name)  
*Set a row name.*
- virtual void [setRowNames](#) (OsiNameVec &srcNames, int srcStart, int len, int tgtStart)  
*Set multiple row names.*
- virtual void [deleteRowNames](#) (int tgtStart, int len)  
*Delete len row names starting at index tgtStart.*
- virtual std::string [getColName](#) (int colIndex, unsigned maxlen=std::string::npos) const  
*Return the name of the column.*
- virtual const OsiNameVec & [getColNames](#) ()  
*Return a pointer to a vector of column names.*
- virtual void [setColName](#) (int ndx, std::string name)  
*Set a column name.*
- virtual void [setColNames](#) (OsiNameVec &srcNames, int srcStart, int len, int tgtStart)  
*Set multiple column names.*
- virtual void [deleteColNames](#) (int tgtStart, int len)  
*Delete len column names starting at index tgtStart.*

### Changing bounds on variables and constraints

- virtual void [setObjCoeff](#) (int elementIndex, double elementValue)  
*Set an objective function coefficient.*
- virtual void [setColLower](#) (int elementIndex, double elementValue)  
*Set a single column lower bound*  
*Use -DBL\_MAX for -infinity.*
- virtual void [setColUpper](#) (int elementIndex, double elementValue)  
*Set a single column upper bound*  
*Use DBL\_MAX for infinity.*
- virtual void [setColBounds](#) (int elementIndex, double lower, double upper)  
*Set a single column lower and upper bound.*
- virtual void [setColSetBounds](#) (const int \*indexFirst, const int \*indexLast, const double \*boundList)  
*Set the bounds on a number of columns simultaneously*  
*The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#) over and over again.*
- virtual void [setRowLower](#) (int elementIndex, double elementValue)  
*Set a single row lower bound*  
*Use -DBL\_MAX for -infinity.*
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)  
*Set a single row upper bound*  
*Use DBL\_MAX for infinity.*
- virtual void [setRowBounds](#) (int elementIndex, double lower, double upper)  
*Set a single row lower and upper bound.*
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)  
*Set the type of a single row*
- virtual void [setRowSetBounds](#) (const int \*indexFirst, const int \*indexLast, const double \*boundList)  
*Set the bounds on a number of rows simultaneously*  
*The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#) over and over again.*
- virtual void [setRowSetTypes](#) (const int \*indexFirst, const int \*indexLast, const char \*senseList, const double \*rhsList, const double \*rangeList)  
*Set the type of a number of rows simultaneously*  
*The default implementation just invokes [setRowType\(\)](#) over and over again.*

### Integrality related changing methods

- virtual void [setContinuous](#) (int index)  
*Set the index-th variable to be a continuous variable.*

- virtual void [setInteger](#) (int index)  
*Set the index-th variable to be an integer variable.*
- virtual void [setContinuous](#) (const int \*indices, int len)  
*Set the variables listed in indices (which is of length len) to be continuous variables.*
- virtual void [setInteger](#) (const int \*indices, int len)  
*Set the variables listed in indices (which is of length len) to be integer variables.*

#### Methods to expand a problem.<br>

*Note that if a column is added then by default it will correspond to a continuous variable.*

- virtual void [addCol](#) (const CoinPackedVectorBase &vec, const double collb, const double colub, const double obj)
- virtual void [addCol](#) (int numberElements, const int \*rows, const double \*elements, const double collb, const double colub, const double obj)  
*Add a column (primal variable) to the problem.*
- virtual void [addCols](#) (const int numcols, const CoinPackedVectorBase \*const \*cols, const double \*collb, const double \*colub, const double \*obj)
- virtual void [deleteCols](#) (const int num, const int \*colIndices)
- virtual void [addRow](#) (const CoinPackedVectorBase &vec, const double rowlb, const double rowub)
- virtual void [addRow](#) (const CoinPackedVectorBase &vec, const char rowsen, const double rowrhs, const double rowrng)
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase \*const \*rows, const double \*rowlb, const double \*rowub)
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase \*const \*rows, const char \*rowsen, const double \*rowrhs, const double \*rowrng)
- virtual void [deleteRows](#) (const int num, const int \*rowIndices)
- virtual void [applyRowCuts](#) (int numberCuts, const OsiRowCut \*cuts)  
*Apply a collection of row cuts which are all effective.*
- virtual void [applyRowCuts](#) (int numberCuts, const OsiRowCut \*\*cuts)  
*Apply a collection of row cuts which are all effective.*

#### Methods to input a problem

- virtual void [loadProblem](#) (const CoinPackedMatrix &matrix, const double \*collb, const double \*colub, const double \*obj, const double \*rowlb, const double \*rowub)  
*Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).*
- virtual void [assignProblem](#) (CoinPackedMatrix \*&matrix, double \*&collb, double \*&colub, double \*&obj, double \*&rowlb, double \*&rowub)  
*Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).*
- virtual void [loadProblem](#) (const CoinPackedMatrix &matrix, const double \*collb, const double \*colub, const double \*obj, const char \*rowsen, const double \*rowrhs, const double \*rowrng)  
*Load in an problem by copying the arguments (the constraints on the rows are given by sense/rhs/range triplets).*
- virtual void [assignProblem](#) (CoinPackedMatrix \*&matrix, double \*&collb, double \*&colub, double \*&obj, char \*&rowsen, double \*&rowrhs, double \*&rowrng)  
*Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by sense/rhs/range triplets).*
- virtual void [loadProblem](#) (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)  
*Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).*
- virtual void [loadProblem](#) (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)

Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).

- virtual int [readMps](#) (const char \*filename, const char \*extension="mps")  
Read an mps file from the given filename (defaults to Osi reader) - returns number of errors (see [OsiMpsReader](#) class)
- virtual void [writeMps](#) (const char \*filename, const char \*extension="mps", double objSense=0.0) const  
Write the problem into an mps file of the given filename.
- virtual int [writeMpsNative](#) (const char \*filename, const char \*\*rowNames, const char \*\*columnNames, int formatType=0, int numberAcross=2, double objSense=0.0) const  
Write the problem into an mps file of the given filename, names may be null.

### Message handling (extra for Cbc messages).

Normally I presume you would want the same language.

If not then you could use underlying model pointer

- void [newLanguage](#) (CoinMessages::Language language)  
Set language.
- void [setLanguage](#) (CoinMessages::Language language)

### Cbc specific public interfaces

- [CbcModel](#) \* [getModelPtr](#) () const  
Get pointer to Cbc model.
- [OsiSolverInterface](#) \* [getRealSolverPtr](#) () const  
Get pointer to underlying solver.
- void [setCutoff](#) (double value)  
Set cutoff bound on the objective function.
- double [getCutoff](#) () const  
Get the cutoff bound on the objective function - always as minimize.
- void [setMaximumNodes](#) (int value)  
Set the [CbcModel::CbcMaxNumNode](#) maximum node limit.
- int [getMaximumNodes](#) () const  
Get the [CbcModel::CbcMaxNumNode](#) maximum node limit.
- void [setMaximumSolutions](#) (int value)  
Set the [CbcModel::CbcMaxNumSol](#) maximum number of solutions.
- int [getMaximumSolutions](#) () const  
Get the [CbcModel::CbcMaxNumSol](#) maximum number of solutions.
- void [setMaximumSeconds](#) (double value)  
Set the [CbcModel::CbcMaximumSeconds](#) maximum number of seconds.
- double [getMaximumSeconds](#) () const  
Get the [CbcModel::CbcMaximumSeconds](#) maximum number of seconds.
- bool [isNodeLimitReached](#) () const  
Node limit reached?
- bool [isSolutionLimitReached](#) () const  
Solution limit reached?
- int [getNodeCount](#) () const  
Get how many Nodes it took to solve the problem.
- int [status](#) () const  
Final status of problem - 0 finished, 1 stopped, 2 difficulties.
- virtual void [passInMessageHandler](#) (CoinMessageHandler \*handler)  
Pass in a message handler.

### Constructors and destructors

- [OsiCbcSolverInterface](#) ([OsiSolverInterface](#) \*solver=NULL, [CbcStrategy](#) \*strategy=NULL)



- Default Constructor.*
- virtual OsiSolverInterface \* [clone](#) (bool copyData=true) const  
*Clone.*
- [OsiCbcSolverInterface](#) (const [OsiCbcSolverInterface](#) &)  
*Copy constructor.*
- [OsiCbcSolverInterface](#) & [operator=](#) (const [OsiCbcSolverInterface](#) &rhs)  
*Assignment operator.*
- virtual [~OsiCbcSolverInterface](#) ()  
*Destructor.*

## Protected Member Functions

### Protected methods

- virtual void [applyRowCut](#) (const OsiRowCut &rc)  
*Apply a row cut (append to constraint matrix).*
- virtual void [applyColCut](#) (const OsiColCut &cc)  
*Apply a column cut (adjust one or more bounds).*

## Protected Attributes

### Protected member data

- [CbcModel](#) \* [modelPtr\\_](#)  
*Cbc model represented by this class instance.*

## Friends

- void [OsiCbcSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)  
*A function that tests the methods in the [OsiCbcSolverInterface](#) class.*

## 7.119.1 Detailed Description

Cbc Solver Interface.

Instantiation of [OsiCbcSolverInterface](#) for the Model Algorithm.

Definition at line 30 of file OsiCbcSolverInterface.hpp.

## 7.119.2 Constructor & Destructor Documentation

7.119.2.1 [OsiCbcSolverInterface::OsiCbcSolverInterface](#) ( [OsiSolverInterface](#) \* *solver* = NULL, [CbcStrategy](#) \* *strategy* = NULL )

Default Constructor.

7.119.2.2 [OsiCbcSolverInterface::OsiCbcSolverInterface](#) ( const [OsiCbcSolverInterface](#) & )

Copy constructor.

7.119.2.3 [virtual OsiCbcSolverInterface::~~OsiCbcSolverInterface](#) ( ) [virtual]

Destructor.

### 7.119.3 Member Function Documentation

7.119.3.1 `virtual void OsiCbcSolverInterface::initialSolve ( ) [virtual]`

Solve initial LP relaxation.

7.119.3.2 `virtual void OsiCbcSolverInterface::resolve ( ) [virtual]`

Resolve an LP relaxation after problem modification.

7.119.3.3 `virtual void OsiCbcSolverInterface::branchAndBound ( ) [virtual]`

Invoke solver's built-in enumeration algorithm.

7.119.3.4 `bool OsiCbcSolverInterface::setIntParam ( OsiIntParam key, int value )`

7.119.3.5 `bool OsiCbcSolverInterface::setDbiParam ( OsiDbiParam key, double value )`

7.119.3.6 `bool OsiCbcSolverInterface::setStrParam ( OsiStrParam key, const std::string & value )`

7.119.3.7 `bool OsiCbcSolverInterface::getIntParam ( OsiIntParam key, int & value ) const`

7.119.3.8 `bool OsiCbcSolverInterface::getDbiParam ( OsiDbiParam key, double & value ) const`

7.119.3.9 `bool OsiCbcSolverInterface::getStrParam ( OsiStrParam key, std::string & value ) const`

7.119.3.10 `virtual bool OsiCbcSolverInterface::setHintParam ( OsiHintParam key, bool yesNo = true, OsiHintStrength strength = OsiHintTry, void * otherInformation = NULL ) [virtual]`

7.119.3.11 `virtual bool OsiCbcSolverInterface::getHintParam ( OsiHintParam key, bool & yesNo, OsiHintStrength & strength, void *& otherInformation ) const [virtual]`

Get a hint parameter.

7.119.3.12 `virtual bool OsiCbcSolverInterface::getHintParam ( OsiHintParam key, bool & yesNo, OsiHintStrength & strength ) const [virtual]`

Get a hint parameter.

7.119.3.13 `virtual bool OsiCbcSolverInterface::isAbandoned ( ) const [virtual]`

Are there a numerical difficulties?

7.119.3.14 `virtual bool OsiCbcSolverInterface::isProvenOptimal ( ) const [virtual]`

Is optimality proven?

7.119.3.15 `virtual bool OsiCbcSolverInterface::isProvenPrimalInfeasible ( ) const [virtual]`

Is primal infeasibility proven?

7.119.3.16 `virtual bool OsiCbcSolverInterface::isProvenDualInfeasible ( ) const [virtual]`

Is dual infeasibility proven?

7.119.3.17 `virtual bool OsiCbcSolverInterface::isPrimalObjectiveLimitReached ( ) const [virtual]`

Is the given primal objective limit reached?

7.119.3.18 `virtual bool OsiCbcSolverInterface::isDualObjectiveLimitReached ( ) const [virtual]`

Is the given dual objective limit reached?

7.119.3.19 `virtual bool OsiCbcSolverInterface::isIterationLimitReached ( ) const [virtual]`

Iteration limit reached?

7.119.3.20 `virtual CoinWarmStart* OsiCbcSolverInterface::getEmptyWarmStart ( ) const [virtual]`

Get an empty warm start object.

This routine returns an empty CoinWarmStartBasis object. Its purpose is to provide a way to give a client a warm start basis object of the appropriate type, which can be resized and modified as desired.

7.119.3.21 `virtual CoinWarmStart* OsiCbcSolverInterface::getWarmStart ( ) const [virtual]`

Get warmstarting information.

7.119.3.22 `virtual bool OsiCbcSolverInterface::setWarmStart ( const CoinWarmStart * warmstart ) [virtual]`

Set warmstarting information.

Return true/false depending on whether the warmstart information was accepted or not.

7.119.3.23 `virtual void OsiCbcSolverInterface::markHotStart ( ) [virtual]`

Create a hotstart point of the optimization process.

7.119.3.24 `virtual void OsiCbcSolverInterface::solveFromHotStart ( ) [virtual]`

Optimize starting from the hotstart.

7.119.3.25 `virtual void OsiCbcSolverInterface::unmarkHotStart ( ) [virtual]`

Delete the snapshot.

7.119.3.26 `virtual int OsiCbcSolverInterface::getNumCols ( ) const [virtual]`

Get number of columns.

7.119.3.27 `virtual int OsiCbcSolverInterface::getNumRows ( ) const [virtual]`

Get number of rows.

7.119.3.28 `virtual int OsiCbcSolverInterface::getNumElements ( ) const [virtual]`

Get number of nonzero elements.

7.119.3.29 `virtual const double* OsiCbcSolverInterface::getColLower ( ) const [virtual]`

Get pointer to array[getNumCols()] of column lower bounds.

**7.119.3.30** `virtual const double* OsiCbcSolverInterface::getColUpper ( ) const` [virtual]

Get pointer to array[[getNumCols\(\)](#)] of column upper bounds.

**7.119.3.31** `virtual const char* OsiCbcSolverInterface::getRowSense ( ) const` [virtual]

Get pointer to array[[getNumRows\(\)](#)] of row constraint senses.

- 'L' <= constraint
- 'E' = constraint
- 'G' >= constraint
- 'R' ranged constraint
- 'N' free constraint

**7.119.3.32** `virtual const double* OsiCbcSolverInterface::getRightHandSide ( ) const` [virtual]

Get pointer to array[[getNumRows\(\)](#)] of rows right-hand sides.

- 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`

**7.119.3.33** `virtual const double* OsiCbcSolverInterface::getRowRange ( ) const` [virtual]

Get pointer to array[[getNumRows\(\)](#)] of row ranges.

- if `rowsense()[i] == 'R'` then `rowrange()[i] == rowupper()[i] - rowlower()[i]`
- if `rowsense()[i] != 'R'` then `rowrange()[i]` is undefined

**7.119.3.34** `virtual const double* OsiCbcSolverInterface::getRowLower ( ) const` [virtual]

Get pointer to array[[getNumRows\(\)](#)] of row lower bounds.

**7.119.3.35** `virtual const double* OsiCbcSolverInterface::getRowUpper ( ) const` [virtual]

Get pointer to array[[getNumRows\(\)](#)] of row upper bounds.

**7.119.3.36** `virtual const double* OsiCbcSolverInterface::getObjCoefficients ( ) const` [virtual]

Get pointer to array[[getNumCols\(\)](#)] of objective function coefficients.

**7.119.3.37** `virtual double OsiCbcSolverInterface::getObjSense ( ) const` [virtual]

Get objective function sense (1 for min (default), -1 for max)

**7.119.3.38** `virtual bool OsiCbcSolverInterface::isContinuous ( int colNumber ) const` [virtual]

Return true if column is continuous.

7.119.3.39 `virtual const CoinPackedMatrix* OsiCbcSolverInterface::getMatrixByRow ( ) const [virtual]`

Get pointer to row-wise copy of matrix.

7.119.3.40 `virtual const CoinPackedMatrix* OsiCbcSolverInterface::getMatrixByCol ( ) const [virtual]`

Get pointer to column-wise copy of matrix.

7.119.3.41 `virtual double OsiCbcSolverInterface::getInfinity ( ) const [virtual]`

Get solver's value for infinity.

7.119.3.42 `virtual const double* OsiCbcSolverInterface::getColSolution ( ) const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of primal solution vector.

7.119.3.43 `virtual const double* OsiCbcSolverInterface::getRowPrice ( ) const [virtual]`

Get pointer to array[[getNumRows\(\)](#)] of dual prices.

7.119.3.44 `virtual const double* OsiCbcSolverInterface::getReducedCost ( ) const [virtual]`

Get a pointer to array[[getNumCols\(\)](#)] of reduced costs.

7.119.3.45 `virtual const double* OsiCbcSolverInterface::getRowActivity ( ) const [virtual]`

Get pointer to array[[getNumRows\(\)](#)] of row activity levels (constraint matrix times the solution vector).

7.119.3.46 `virtual double OsiCbcSolverInterface::getObjValue ( ) const [virtual]`

Get objective function value.

7.119.3.47 `virtual int OsiCbcSolverInterface::getIterationCount ( ) const [virtual]`

Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver).

7.119.3.48 `virtual std::vector<double*> OsiCbcSolverInterface::getDualRays ( int maxNumRays, bool fullRay = false ) const [virtual]`

Get as many dual rays as the solver can provide.

(In case of proven primal infeasibility there should be at least one.)

The first [getNumRows\(\)](#) ray components will always be associated with the row duals (as returned by [getRowPrice\(\)](#)). If `fullRay` is true, the final [getNumCols\(\)](#) entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

**NOTE for implementers of solver interfaces:**

The double pointers in the vector should point to arrays of length [getNumRows\(\)](#) and they should be allocated via `new[]`.

**NOTE for users of solver interfaces:**

It is the user's responsibility to free the double pointers in the vector using `delete[]`.

7.119.3.49 `virtual std::vector<double*> OsiCbcSolverInterface::getPrimalRays ( int maxNumRays ) const [virtual]`

Get as many primal rays as the solver can provide.

(In case of proven dual infeasibility there should be at least one.)

**NOTE for implementers of solver interfaces:**

The double pointers in the vector should point to arrays of length `getNumCols()` and they should be allocated via `new[]`.

**NOTE for users of solver interfaces:**

It is the user's responsibility to free the double pointers in the vector using `delete[]`.

```
7.119.3.50 virtual std::string OsiCbcSolverInterface::dfltRowColName ( char rc, int ndx, unsigned digits = 7 ) const
           [virtual]
```

Generate a standard name of the form Rnnnnnnn or Cnnnnnnn.

```
7.119.3.51 virtual std::string OsiCbcSolverInterface::getObjName ( unsigned maxLen = std::string::npos ) const
           [virtual]
```

Return the name of the objective function.

```
7.119.3.52 virtual void OsiCbcSolverInterface::setObjName ( std::string name ) [virtual]
```

Set the name of the objective function.

```
7.119.3.53 virtual std::string OsiCbcSolverInterface::getRowName ( int rowIndex, unsigned maxLen = std::string::npos
           ) const [virtual]
```

Return the name of the row.

```
7.119.3.54 virtual const OsiNameVec& OsiCbcSolverInterface::getRowNames ( ) [virtual]
```

Return a pointer to a vector of row names.

```
7.119.3.55 virtual void OsiCbcSolverInterface::setRowName ( int ndx, std::string name ) [virtual]
```

Set a row name.

```
7.119.3.56 virtual void OsiCbcSolverInterface::setRowNames ( OsiNameVec & srcNames, int srcStart, int len, int tgtStart )
           [virtual]
```

Set multiple row names.

```
7.119.3.57 virtual void OsiCbcSolverInterface::deleteRowNames ( int tgtStart, int len ) [virtual]
```

Delete len row names starting at index tgtStart.

```
7.119.3.58 virtual std::string OsiCbcSolverInterface::getColName ( int colIndex, unsigned maxLen = std::string::npos )
           const [virtual]
```

Return the name of the column.

```
7.119.3.59 virtual const OsiNameVec& OsiCbcSolverInterface::getColNames ( ) [virtual]
```

Return a pointer to a vector of column names.

```
7.119.3.60 virtual void OsiCbcSolverInterface::setColName ( int ndx, std::string name ) [virtual]
```

Set a column name.

7.119.3.61 `virtual void OsiCbcSolverInterface::setColNames ( OsiNameVec & srcNames, int srcStart, int len, int tgtStart )` [virtual]

Set multiple column names.

7.119.3.62 `virtual void OsiCbcSolverInterface::deleteColNames ( int tgtStart, int len )` [virtual]

Delete *len* column names starting at index *tgtStart*.

7.119.3.63 `virtual void OsiCbcSolverInterface::setObjCoeff ( int elementIndex, double elementValue )` [virtual]

Set an objective function coefficient.

7.119.3.64 `virtual void OsiCbcSolverInterface::setColLower ( int elementIndex, double elementValue )` [virtual]

Set a single column lower bound

Use -DBL\_MAX for -infinity.

7.119.3.65 `virtual void OsiCbcSolverInterface::setColUpper ( int elementIndex, double elementValue )` [virtual]

Set a single column upper bound

Use DBL\_MAX for infinity.

7.119.3.66 `virtual void OsiCbcSolverInterface::setColBounds ( int elementIndex, double lower, double upper )` [virtual]

Set a single column lower and upper bound.

7.119.3.67 `virtual void OsiCbcSolverInterface::setColSetBounds ( const int * indexFirst, const int * indexLast, const double * boundList )` [virtual]

Set the bounds on a number of columns simultaneously

The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#) over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the variables whose <i>either</i> bound changes
<i>boundList</i>	the new lower/upper bound pairs for the variables

7.119.3.68 `virtual void OsiCbcSolverInterface::setRowLower ( int elementIndex, double elementValue )` [virtual]

Set a single row lower bound

Use -DBL\_MAX for -infinity.

7.119.3.69 `virtual void OsiCbcSolverInterface::setRowUpper ( int elementIndex, double elementValue )` [virtual]

Set a single row upper bound

Use DBL\_MAX for infinity.

7.119.3.70 `virtual void OsiCbcSolverInterface::setRowBounds ( int elementIndex, double lower, double upper )` [virtual]

Set a single row lower and upper bound.

**7.119.3.71** `virtual void OsiCbcSolverInterface::setRowType ( int index, char sense, double rightHandSide, double range )`  
`[virtual]`

Set the type of a single row

**7.119.3.72** `virtual void OsiCbcSolverInterface::setRowSetBounds ( const int * indexFirst, const int * indexLast, const double * boundList )` `[virtual]`

Set the bounds on a number of rows simultaneously

The default implementation just invokes `setRowLower()` and `setRowUpper()` over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the constraints whose <i>either</i> bound changes
<i>boundList</i>	the new lower/upper bound pairs for the constraints

**7.119.3.73** `virtual void OsiCbcSolverInterface::setRowSetTypes ( const int * indexFirst, const int * indexLast, const char * senseList, const double * rhsList, const double * rangeList )` `[virtual]`

Set the type of a number of rows simultaneously

The default implementation just invokes `setRowType()` over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the constraints whose <i>any</i> characteristics changes
<i>senseList</i>	the new senses
<i>rhsList</i>	the new right hand sides
<i>rangeList</i>	the new ranges

**7.119.3.74** `virtual void OsiCbcSolverInterface::setContinuous ( int index )` `[virtual]`

Set the index-th variable to be a continuous variable.

**7.119.3.75** `virtual void OsiCbcSolverInterface::setInteger ( int index )` `[virtual]`

Set the index-th variable to be an integer variable.

**7.119.3.76** `virtual void OsiCbcSolverInterface::setContinuous ( const int * indices, int len )` `[virtual]`

Set the variables listed in indices (which is of length len) to be continuous variables.

**7.119.3.77** `virtual void OsiCbcSolverInterface::setInteger ( const int * indices, int len )` `[virtual]`

Set the variables listed in indices (which is of length len) to be integer variables.

**7.119.3.78** `virtual void OsiCbcSolverInterface::setObjSense ( double s )` `[virtual]`

Set objective function sense (1 for min (default), -1 for max,)

**7.119.3.79** `virtual void OsiCbcSolverInterface::setColSolution ( const double * colsol )` `[virtual]`

Set the primal solution column values.

`colsol[numcols()]` is an array of values of the problem column variables. These values are copied to memory owned by



the solver object or the solver. They will be returned as the result of `colsol()` until changed by another call to `setColsol()` or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

**7.119.3.80** `virtual void OsiCbcSolverInterface::setRowPrice ( const double * rowprice ) [virtual]`

Set dual solution vector.

`rowprice[numrows()]` is an array of values of the problem row dual variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of `rowprice()` until changed by another call to `setRowprice()` or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

**7.119.3.81** `virtual void OsiCbcSolverInterface::addCol ( const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj ) [virtual]`

**7.119.3.82** `virtual void OsiCbcSolverInterface::addCol ( int numberElements, const int * rows, const double * elements, const double collb, const double colub, const double obj ) [virtual]`

Add a column (primal variable) to the problem.

**7.119.3.83** `virtual void OsiCbcSolverInterface::addCols ( const int numcols, const CoinPackedVectorBase *const * cols, const double * collb, const double * colub, const double * obj ) [virtual]`

**7.119.3.84** `virtual void OsiCbcSolverInterface::deleteCols ( const int num, const int * colIndices ) [virtual]`

**7.119.3.85** `virtual void OsiCbcSolverInterface::addRow ( const CoinPackedVectorBase & vec, const double rowlb, const double rowub ) [virtual]`

**7.119.3.86** `virtual void OsiCbcSolverInterface::addRow ( const CoinPackedVectorBase & vec, const char rowSEN, const double rowrhs, const double rowrng ) [virtual]`

**7.119.3.87** `virtual void OsiCbcSolverInterface::addRows ( const int numrows, const CoinPackedVectorBase *const * rows, const double * rowlb, const double * rowub ) [virtual]`

**7.119.3.88** `virtual void OsiCbcSolverInterface::addRows ( const int numrows, const CoinPackedVectorBase *const * rows, const char * rowSEN, const double * rowrhs, const double * rowrng ) [virtual]`

**7.119.3.89** `virtual void OsiCbcSolverInterface::deleteRows ( const int num, const int * rowIndices ) [virtual]`

**7.119.3.90** `virtual void OsiCbcSolverInterface::applyRowCuts ( int numberCuts, const OsiRowCut * cuts ) [virtual]`

Apply a collection of row cuts which are all effective.

`applyCuts` seems to do one at a time which seems inefficient.

**7.119.3.91** `virtual void OsiCbcSolverInterface::applyRowCuts ( int numberCuts, const OsiRowCut ** cuts ) [virtual]`

Apply a collection of row cuts which are all effective.

`applyCuts` seems to do one at a time which seems inefficient. This uses array of pointers

**7.119.3.92** `virtual void OsiCbcSolverInterface::loadProblem ( const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub ) [virtual]`

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

**7.119.3.93** `virtual void OsiCbcSolverInterface::assignProblem ( CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, double *& rowlb, double *& rowub ) [virtual]`

Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).

For default values see the previous method.

**WARNING:** The arguments passed to this method will be freed using the C++ `delete` and `delete[]` functions.

**7.119.3.94** `virtual void OsiCbcSolverInterface::loadProblem ( const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const char * rowsen, const double * rowrhs, const double * rowrng ) [virtual]`

Load in an 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

**7.119.3.95** `virtual void OsiCbcSolverInterface::assignProblem ( CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, char *& rowsen, double *& rowrhs, double *& rowrng ) [virtual]`

Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by sense/rhs/range triplets).

For default values see the previous method.

**WARNING:** The arguments passed to this method will be freed using the C++ `delete` and `delete[]` functions.

**7.119.3.96** `virtual void OsiCbcSolverInterface::loadProblem ( 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 ) [virtual]`

Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).

**7.119.3.97** `virtual void OsiCbcSolverInterface::loadProblem ( 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 ) [virtual]`

Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).

7.119.3.98 `virtual int OsiCbcSolverInterface::readMps ( const char * filename, const char * extension = "mps" ) [virtual]`

Read an mps file from the given filename (defaults to Osi reader) - returns number of errors (see OsiMpsReader class)

7.119.3.99 `virtual void OsiCbcSolverInterface::writeMps ( const char * filename, const char * extension = "mps", double objSense = 0.0 ) const [virtual]`

Write the problem into an mps file of the given filename.

If objSense is non zero then -1.0 forces the code to write a maximization objective and +1.0 to write a minimization one.

If 0.0 then solver can do what it wants

7.119.3.100 `virtual int OsiCbcSolverInterface::writeMpsNative ( const char * filename, const char ** rowNames, const char ** columnNames, int formatType = 0, int numberAcross = 2, double objSense = 0.0 ) const [virtual]`

Write the problem into an mps file of the given filename, names may be null.

formatType is 0 - normal 1 - extra accuracy 2 - IEEE hex (later)

Returns non-zero on I/O error

7.119.3.101 `void OsiCbcSolverInterface::newLanguage ( CoinMessages::Language language )`

Set language.

7.119.3.102 `void OsiCbcSolverInterface::setLanguage ( CoinMessages::Language language ) [inline]`

Definition at line 655 of file OsiCbcSolverInterface.hpp.

7.119.3.103 `CbcModel* OsiCbcSolverInterface::getModelPtr ( ) const [inline]`

Get pointer to Cbc model.

Definition at line 663 of file OsiCbcSolverInterface.hpp.

7.119.3.104 `OsiSolverInterface* OsiCbcSolverInterface::getRealSolverPtr ( ) const [inline]`

Get pointer to underlying solver.

Definition at line 666 of file OsiCbcSolverInterface.hpp.

7.119.3.105 `void OsiCbcSolverInterface::setCutoff ( double value ) [inline]`

Set cutoff bound on the objective function.

Definition at line 669 of file OsiCbcSolverInterface.hpp.

7.119.3.106 `double OsiCbcSolverInterface::getCutoff ( ) const [inline]`

Get the cutoff bound on the objective function - always as minimize.

Definition at line 672 of file OsiCbcSolverInterface.hpp.

7.119.3.107 `void OsiCbcSolverInterface::setMaximumNodes ( int value ) [inline]`

Set the [CbcModel::CbcMaxNumNode](#) maximum node limit.

Definition at line 675 of file OsiCbcSolverInterface.hpp.

7.119.3.108 `int OsiCbcSolverInterface::getMaximumNodes ( ) const [inline]`

Get the [CbcModel::CbcMaxNumNode](#) maximum node limit.

Definition at line 678 of file `OsiCbcSolverInterface.hpp`.

7.119.3.109 `void OsiCbcSolverInterface::setMaximumSolutions ( int value ) [inline]`

Set the [CbcModel::CbcMaxNumSol](#) maximum number of solutions.

Definition at line 681 of file `OsiCbcSolverInterface.hpp`.

7.119.3.110 `int OsiCbcSolverInterface::getMaximumSolutions ( ) const [inline]`

Get the [CbcModel::CbcMaxNumSol](#) maximum number of solutions.

Definition at line 684 of file `OsiCbcSolverInterface.hpp`.

7.119.3.111 `void OsiCbcSolverInterface::setMaximumSeconds ( double value ) [inline]`

Set the [CbcModel::CbcMaximumSeconds](#) maximum number of seconds.

Definition at line 687 of file `OsiCbcSolverInterface.hpp`.

7.119.3.112 `double OsiCbcSolverInterface::getMaximumSeconds ( ) const [inline]`

Get the [CbcModel::CbcMaximumSeconds](#) maximum number of seconds.

Definition at line 690 of file `OsiCbcSolverInterface.hpp`.

7.119.3.113 `bool OsiCbcSolverInterface::isNodeLimitReached ( ) const [inline]`

Node limit reached?

Definition at line 693 of file `OsiCbcSolverInterface.hpp`.

7.119.3.114 `bool OsiCbcSolverInterface::isSolutionLimitReached ( ) const [inline]`

Solution limit reached?

Definition at line 696 of file `OsiCbcSolverInterface.hpp`.

7.119.3.115 `int OsiCbcSolverInterface::getNodeCount ( ) const [inline]`

Get how many Nodes it took to solve the problem.

Definition at line 699 of file `OsiCbcSolverInterface.hpp`.

7.119.3.116 `int OsiCbcSolverInterface::status ( ) const [inline]`

Final status of problem - 0 finished, 1 stopped, 2 difficulties.

Definition at line 702 of file `OsiCbcSolverInterface.hpp`.

7.119.3.117 `virtual void OsiCbcSolverInterface::passInMessageHandler ( CoinMessageHandler * handler ) [virtual]`

Pass in a message handler.

It is the client's responsibility to destroy a message handler installed by this routine; it will not be destroyed when the solver interface is destroyed.

7.119.3.118 `virtual OsiSolverInterface* OsiCbcSolverInterface::clone ( bool copyData = true ) const` [virtual]

Clone.

7.119.3.119 `OsiCbcSolverInterface& OsiCbcSolverInterface::operator= ( const OsiCbcSolverInterface & rhs )`

Assignment operator.

7.119.3.120 `virtual void OsiCbcSolverInterface::applyRowCut ( const OsiRowCut & rc )` [protected],[virtual]

Apply a row cut (append to constraint matrix).

7.119.3.121 `virtual void OsiCbcSolverInterface::applyColCut ( const OsiColCut & cc )` [protected],[virtual]

Apply a column cut (adjust one or more bounds).

#### 7.119.4 Friends And Related Function Documentation

7.119.4.1 `void OsiCbcSolverInterfaceUnitTest ( const std::string & mpsDir, const std::string & netlibDir )` [friend]

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

#### 7.119.5 Member Data Documentation

7.119.5.1 `CbcModel* OsiCbcSolverInterface::modelPtr_` [mutable],[protected]

Cbc model represented by this class instance.

Definition at line 754 of file `OsiCbcSolverInterface.hpp`.

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

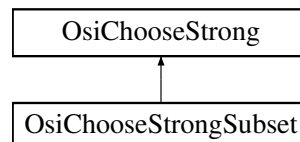
- `/home/ted/COIN/trunk/Cbc/src/OsiCbc/OsiCbcSolverInterface.hpp`

## 7.120 OsiChooseStrongSubset Class Reference

This class chooses a variable to branch on.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for OsiChooseStrongSubset:



#### Public Member Functions

- [OsiChooseStrongSubset](#) ()  
*Default Constructor.*
- [OsiChooseStrongSubset](#) (const OsiSolverInterface \*solver)

- Constructor from solver (so we can set up arrays etc)
- `OsiChooseStrongSubset` (const `OsiChooseStrongSubset` &)
- Copy constructor.
- `OsiChooseStrongSubset` & `operator=` (const `OsiChooseStrongSubset` &rhs)
- Assignment operator.
- virtual `OsiChooseVariable` \* `clone` () const
- Clone.
- virtual `~OsiChooseStrongSubset` ()
- Destructor.
- virtual int `setupList` (OsiBranchingInformation \*info, bool initialize)
- Sets up strong list and clears all if initialize is true.
- virtual int `chooseVariable` (OsiSolverInterface \*solver, OsiBranchingInformation \*info, bool fixVariables)
- Choose a variable Returns - -1 Node is infeasible 0 Normal termination - we have a candidate 1 All looks satisfied - no candidate 2 We can change the bound on a variable - but we also have a strong branching candidate 3 We can change the bound on a variable - but we have a non-strong branching candidate 4 We can change the bound on a variable - no other candidates We can pick up branch from `bestObjectIndex()` and `bestWhichWay()` We can pick up a forced branch (can change bound) from `firstForcedObjectIndex()` and `firstForcedWhichWay()` If we have a solution then we can pick up from `goodObjectiveValue()` and `goodSolution()` If fixVariables is true then 2,3,4 are all really same as problem changed.
- int `numberObjectsToUse` () const
- Number of objects to use.
- void `setNumberObjectsToUse` (int value)
- Set number of objects to use.

#### Protected Attributes

- int `numberObjectsToUse_`
- Number of objects to be used (and set in solver)

#### 7.120.1 Detailed Description

This class chooses a variable to branch on.

This is just as `OsiChooseStrong` but it fakes it so only first so many are looked at in this phase

Definition at line 1203 of file `CbcLinked.hpp`.

#### 7.120.2 Constructor & Destructor Documentation

##### 7.120.2.1 `OsiChooseStrongSubset::OsiChooseStrongSubset ( )`

Default Constructor.

##### 7.120.2.2 `OsiChooseStrongSubset::OsiChooseStrongSubset ( const OsiSolverInterface * solver )`

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

##### 7.120.2.3 `OsiChooseStrongSubset::OsiChooseStrongSubset ( const OsiChooseStrongSubset & )`

Copy constructor.

7.120.2.4 `virtual OsiChooseStrongSubset::~~OsiChooseStrongSubset ( ) [virtual]`

Destructor.

### 7.120.3 Member Function Documentation

7.120.3.1 `OsiChooseStrongSubset& OsiChooseStrongSubset::operator= ( const OsiChooseStrongSubset & rhs )`

Assignment operator.

7.120.3.2 `virtual OsiChooseVariable* OsiChooseStrongSubset::clone ( ) const [virtual]`

Clone.

7.120.3.3 `virtual int OsiChooseStrongSubset::setupList ( OsiBranchingInformation * info, bool initialize ) [virtual]`

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

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

7.120.3.4 `virtual int OsiChooseStrongSubset::chooseVariable ( OsiSolverInterface * solver, OsiBranchingInformation * info, bool fixVariables ) [virtual]`

Choose a variable Returns - -1 Node is infeasible 0 Normal termination - we have a candidate 1 All looks satisfied - no candidate 2 We can change the bound on a variable - but we also have a strong branching candidate 3 We can change the bound on a variable - but we have a non-strong branching candidate 4 We can change the bound on a variable - no other candidates We can pick up branch from bestObjectIndex() and bestWhichWay() We can pick up a forced branch (can change bound) from firstForcedObjectIndex() and firstForcedWhichWay() If we have a solution then we can pick up from goodObjectiveValue() and goodSolution() If fixVariables is true then 2,3,4 are all really same as problem changed.

7.120.3.5 `int OsiChooseStrongSubset::numberOfObjectsToUse ( ) const [inline]`

Number of objects to use.

Definition at line 1246 of file CbcLinked.hpp.

7.120.3.6 `void OsiChooseStrongSubset::setNumberOfObjectsToUse ( int value ) [inline]`

Set number of objects to use.

Definition at line 1250 of file CbcLinked.hpp.

### 7.120.4 Member Data Documentation

7.120.4.1 `int OsiChooseStrongSubset::numberOfObjectsToUse_ [protected]`

Number of objects to be used (and set in solver)

Definition at line 1257 of file CbcLinked.hpp.

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

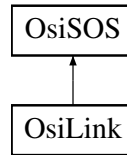
- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.121 OsiLink Class Reference

Define Special Linked Ordered Sets.

```
#include <CbCLinked.hpp>
```

Inheritance diagram for OsiLink:



### Public Member Functions

- [OsiLink](#) ()
- [OsiLink](#) (const OsiSolverInterface \*solver, int yRow, int yColumn, double meshSize)  
*Useful constructor -.*
- [OsiLink](#) (const [OsiLink](#) &)
- virtual OsiObject \* [clone](#) () const  
*Clone.*
- [OsiLink](#) & [operator=](#) (const [OsiLink](#) &rhs)
- virtual ~[OsiLink](#) ()
- virtual double [infeasibility](#) (const OsiBranchingInformation \*info, int &whichWay) const  
*Infeasibility - large is 0.5.*
- virtual double [feasibleRegion](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info) const  
*Set bounds to fix the variable at the current (integer) value.*
- virtual OsiBranchingObject \* [createBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way) const  
*Creates a branching object.*
- virtual void [resetSequenceEtc](#) (int numberColumns, const int \*originalColumns)  
*Redoes data when sequence numbers change.*
- int [numberLinks](#) () const  
*Number of links for each member.*
- virtual bool [canDoHeuristics](#) () const  
*Return true if object can take part in normal heuristics.*
- virtual bool [boundBranch](#) () const  
*Return true if branch should only bound variables.*

### 7.121.1 Detailed Description

Define Special Linked Ordered Sets.

New style

members and weights may be stored in SOS object

This is for y and  $x*f(y)$  and  $z*g(y)$  etc

Definition at line 600 of file CbcLinked.hpp.



## 7.121.2 Constructor &amp; Destructor Documentation

7.121.2.1 OsiLink::OsiLink ( )

7.121.2.2 OsiLink::OsiLink ( const OsiSolverInterface \* *solver*, int *yRow*, int *yColumn*, double *meshSize* )

Useful constructor -.

7.121.2.3 OsiLink::OsiLink ( const OsiLink &amp; )

7.121.2.4 virtual OsiLink::~~OsiLink ( ) [virtual]

## 7.121.3 Member Function Documentation

7.121.3.1 virtual OsiObject\* OsiLink::clone ( ) const [virtual]

Clone.

7.121.3.2 OsiLink& OsiLink::operator= ( const OsiLink & *rhs* )7.121.3.3 virtual double OsiLink::infeasibility ( const OsiBranchingInformation \* *info*, int & *whichWay* ) const [virtual]

Infeasibility - large is 0.5.

7.121.3.4 virtual double OsiLink::feasibleRegion ( OsiSolverInterface \* *solver*, const OsiBranchingInformation \* *info* ) const [virtual]

Set bounds to fix the variable at the current (integer) value.

Given an integer value, set the lower and upper bounds to fix the variable. Returns amount it had to move variable.

7.121.3.5 virtual OsiBranchingObject\* OsiLink::createBranch ( OsiSolverInterface \* *solver*, const OsiBranchingInformation \* *info*, int *way* ) const [virtual]

Creates a branching object.

The preferred direction is set by *way*, 0 for down, 1 for up.7.121.3.6 virtual void OsiLink::resetSequenceEtc ( int *numberColumns*, const int \* *originalColumns* ) [virtual]

Redoes data when sequence numbers change.

7.121.3.7 int OsiLink::numberLinks ( ) const [inline]

Number of links for each member.

Definition at line 647 of file CbcLinked.hpp.

7.121.3.8 virtual bool OsiLink::canDoHeuristics ( ) const [inline],[virtual]

Return true if object can take part in normal heuristics.

Definition at line 653 of file CbcLinked.hpp.

7.121.3.9 virtual bool OsiLink::boundBranch ( ) const [inline],[virtual]

Return true if branch should only bound variables.

Definition at line 658 of file CbcLinked.hpp.

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

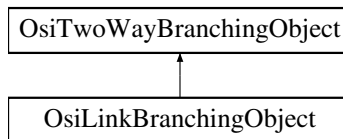
- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.122 OsiLinkBranchingObject Class Reference

Branching object for Linked ordered sets.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for OsiLinkBranchingObject:



### Public Member Functions

- [OsiLinkBranchingObject](#) ()
- [OsiLinkBranchingObject](#) (OsiSolverInterface \*solver, const [OsiLink](#) \*originalObject, int way, double separator)
- [OsiLinkBranchingObject](#) (const [OsiLinkBranchingObject](#) &)
- [OsiLinkBranchingObject](#) & operator= (const [OsiLinkBranchingObject](#) &rhs)
- virtual OsiBranchingObject \* [clone](#) () const  
*Clone.*
- virtual [~OsiLinkBranchingObject](#) ()
- virtual double [branch](#) (OsiSolverInterface \*solver)  
*Does next branch and updates state.*
- virtual void [print](#) (const OsiSolverInterface \*solver=NULL)  
*Print something about branch - only if log level high.*

### 7.122.1 Detailed Description

Branching object for Linked ordered sets.

Definition at line 678 of file CbcLinked.hpp.

### 7.122.2 Constructor & Destructor Documentation

7.122.2.1 [OsiLinkBranchingObject::OsiLinkBranchingObject](#) ( )

7.122.2.2 [OsiLinkBranchingObject::OsiLinkBranchingObject](#) ( OsiSolverInterface \* solver, const [OsiLink](#) \* originalObject, int way, double separator )

7.122.2.3 [OsiLinkBranchingObject::OsiLinkBranchingObject](#) ( const [OsiLinkBranchingObject](#) & )

7.122.2.4 [virtual OsiLinkBranchingObject::~~OsiLinkBranchingObject](#) ( ) [virtual]

### 7.122.3 Member Function Documentation

7.122.3.1 `OsiLinkBranchingObject& OsiLinkBranchingObject::operator= ( const OsiLinkBranchingObject & rhs )`

7.122.3.2 `virtual OsiBranchingObject* OsiLinkBranchingObject::clone ( ) const [virtual]`

Clone.

7.122.3.3 `virtual double OsiLinkBranchingObject::branch ( OsiSolverInterface * solver ) [virtual]`

Does next branch and updates state.

7.122.3.4 `virtual void OsiLinkBranchingObject::print ( const OsiSolverInterface * solver=NULL ) [virtual]`

Print something about branch - only if log level high.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.123 OsiLinkedBound Class Reference

List of bounds which depend on other bounds.

```
#include <CbcLinked.hpp>
```

### Public Member Functions

#### Action methods

- void [updateBounds](#) (ClpSimplex \*solver)  
*Update other bounds.*

#### Constructors and destructors

- [OsiLinkedBound](#) ()  
*Default Constructor.*
- [OsiLinkedBound](#) (OsiSolverInterface \*model, int [variable](#), int numberAffected, const int \*positionL, const int \*positionU, const double \*multiplier)  
*Useful Constructor.*
- [OsiLinkedBound](#) (const [OsiLinkedBound](#) &)  
*Copy constructor.*
- [OsiLinkedBound](#) & [operator=](#) (const [OsiLinkedBound](#) &rhs)  
*Assignment operator.*
- [~OsiLinkedBound](#) ()  
*Destructor.*

#### Sets and Gets

- int [variable](#) () const  
*Get variable.*
- void [addBoundModifier](#) (bool upperBoundAffected, bool useUpperBound, int whichVariable, double multiplier=1.0)  
*Add a bound modifier.*

### 7.123.1 Detailed Description

List of bounds which depend on other bounds.

Definition at line 300 of file CbcLinked.hpp.

### 7.123.2 Constructor & Destructor Documentation

#### 7.123.2.1 OsiLinkedBound::OsiLinkedBound ( )

Default Constructor.

#### 7.123.2.2 OsiLinkedBound::OsiLinkedBound ( OsiSolverInterface \* *model*, int *variable*, int *numberAffected*, const int \* *positionL*, const int \* *positionU*, const double \* *multiplier* )

Useful Constructor.

#### 7.123.2.3 OsiLinkedBound::OsiLinkedBound ( const OsiLinkedBound & )

Copy constructor.

#### 7.123.2.4 OsiLinkedBound::~~OsiLinkedBound ( )

Destructor.

### 7.123.3 Member Function Documentation

#### 7.123.3.1 void OsiLinkedBound::updateBounds ( ClpSimplex \* *solver* )

Update other bounds.

#### 7.123.3.2 OsiLinkedBound& OsiLinkedBound::operator= ( const OsiLinkedBound & *rhs* )

Assignment operator.

#### 7.123.3.3 int OsiLinkedBound::variable ( ) const [inline]

Get variable.

Definition at line 334 of file CbcLinked.hpp.

#### 7.123.3.4 void OsiLinkedBound::addBoundModifier ( bool *upperBoundAffected*, bool *useUpperBound*, int *whichVariable*, double *multiplier* = 1.0 )

Add a bound modifier.

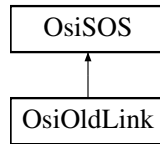
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.124 OsiOldLink Class Reference

```
#include <CbcLinked.hpp>
```

Inheritance diagram for OsiOldLink:



## Public Member Functions

- [OsiOldLink](#) ()
- [OsiOldLink](#) (const OsiSolverInterface \*solver, int numberMembers, int [numberLinks](#), int first, const double \*weights, int setNumber)
 

*Useful constructor - A valid solution is if all variables are zero apart from  $k \cdot \text{numberLink}$  to  $(k+1) \cdot \text{numberLink} - 1$  where  $k$  is 0 through  $\text{numberInSet} - 1$ .*
- [OsiOldLink](#) (const OsiSolverInterface \*solver, int numberMembers, int [numberLinks](#), int typeSOS, const int \*which, const double \*weights, int setNumber)
 

*Useful constructor - A valid solution is if all variables are zero apart from  $k \cdot \text{numberLink}$  to  $(k+1) \cdot \text{numberLink} - 1$  where  $k$  is 0 through  $\text{numberInSet} - 1$ .*
- [OsiOldLink](#) (const [OsiOldLink](#) &)
- virtual OsiObject \* [clone](#) () const
 

*Clone.*
- [OsiOldLink](#) & [operator=](#) (const [OsiOldLink](#) &rhs)
- virtual ~[OsiOldLink](#) ()
- virtual double [infeasibility](#) (const OsiBranchingInformation \*info, int &whichWay) const
 

*Infeasibility - large is 0.5.*
- virtual double [feasibleRegion](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info) const
 

*Set bounds to fix the variable at the current (integer) value.*
- virtual OsiBranchingObject \* [createBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way) const
 

*Creates a branching object.*
- virtual void [resetSequenceEtc](#) (int numberColumns, const int \*originalColumns)
 

*Redoes data when sequence numbers change.*
- int [numberLinks](#) () const
 

*Number of links for each member.*
- virtual bool [canDoHeuristics](#) () const
 

*Return true if object can take part in normal heuristics.*
- virtual bool [boundBranch](#) () const
 

*Return true if branch should only bound variables.*

### 7.124.1 Detailed Description

Definition at line 434 of file CbcLinked.hpp.

### 7.124.2 Constructor & Destructor Documentation

#### 7.124.2.1 OsiOldLink::OsiOldLink ( )

**7.124.2.2** `OsiOldLink::OsiOldLink ( const OsiSolverInterface * solver, int numberMembers, int numberLinks, int first, const double * weights, int setNumber )`

Useful constructor - A valid solution is if all variables are zero apart from  $k \cdot \text{numberLink}$  to  $(k+1) \cdot \text{numberLink} - 1$  where  $k$  is 0 through  $\text{numberInSet} - 1$ .

The length of weights array is  $\text{numberInSet}$ . For this constructor the variables in matrix are the  $\text{numberInSet} \cdot \text{numberLink}$  starting at first. If weights null then 0,1,2..

**7.124.2.3** `OsiOldLink::OsiOldLink ( const OsiSolverInterface * solver, int numberMembers, int numberLinks, int typeSOS, const int * which, const double * weights, int setNumber )`

Useful constructor - A valid solution is if all variables are zero apart from  $k \cdot \text{numberLink}$  to  $(k+1) \cdot \text{numberLink} - 1$  where  $k$  is 0 through  $\text{numberInSet} - 1$ .

The length of weights array is  $\text{numberInSet}$ . For this constructor the variables are given by list - grouped. If weights null then 0,1,2..

**7.124.2.4** `OsiOldLink::OsiOldLink ( const OsiOldLink & )`

**7.124.2.5** `virtual OsiOldLink::~~OsiOldLink ( ) [virtual]`

#### 7.124.3 Member Function Documentation

**7.124.3.1** `virtual OsiObject* OsiOldLink::clone ( ) const [virtual]`

Clone.

**7.124.3.2** `OsiOldLink& OsiOldLink::operator= ( const OsiOldLink & rhs )`

**7.124.3.3** `virtual double OsiOldLink::infeasibility ( const OsiBranchingInformation * info, int & whichWay ) const [virtual]`

Infeasibility - large is 0.5.

**7.124.3.4** `virtual double OsiOldLink::feasibleRegion ( OsiSolverInterface * solver, const OsiBranchingInformation * info ) const [virtual]`

Set bounds to fix the variable at the current (integer) value.

Given an integer value, set the lower and upper bounds to fix the variable. Returns amount it had to move variable.

**7.124.3.5** `virtual OsiBranchingObject* OsiOldLink::createBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way ) const [virtual]`

Creates a branching object.

The preferred direction is set by `way`, 0 for down, 1 for up.

**7.124.3.6** `virtual void OsiOldLink::resetSequenceEtc ( int numberColumns, const int * originalColumns ) [virtual]`

Redoes data when sequence numbers change.

**7.124.3.7** `int OsiOldLink::numberLinks ( ) const [inline]`

Number of links for each member.

Definition at line 494 of file `CbcLinked.hpp`.

7.124.3.8 `virtual bool OsiOldLink::canDoHeuristics ( ) const [inline],[virtual]`

Return true if object can take part in normal heuristics.

Definition at line 500 of file CbcLinked.hpp.

7.124.3.9 `virtual bool OsiOldLink::boundBranch ( ) const [inline],[virtual]`

Return true if branch should only bound variables.

Definition at line 505 of file CbcLinked.hpp.

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

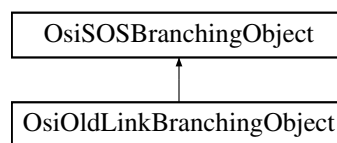
- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.125 OsiOldLinkBranchingObject Class Reference

Branching object for Linked ordered sets.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for OsiOldLinkBranchingObject:



### Public Member Functions

- [OsiOldLinkBranchingObject](#) ( )
- [OsiOldLinkBranchingObject](#) (OsiSolverInterface \*solver, const [OsiOldLink](#) \*originalObject, int way, double separator)
- [OsiOldLinkBranchingObject](#) (const [OsiOldLinkBranchingObject](#) &)
- [OsiOldLinkBranchingObject](#) & operator= (const [OsiOldLinkBranchingObject](#) &rhs)
- virtual [OsiBranchingObject](#) \* [clone](#) ( ) const  
*Clone.*
- virtual [~OsiOldLinkBranchingObject](#) ( )
- virtual double [branch](#) (OsiSolverInterface \*solver)  
*Does next branch and updates state.*
- virtual void [print](#) (const OsiSolverInterface \*solver=NULL)  
*Print something about branch - only if log level high.*

### 7.125.1 Detailed Description

Branching object for Linked ordered sets.

Definition at line 518 of file CbcLinked.hpp.

### 7.125.2 Constructor & Destructor Documentation

7.125.2.1 `OsiOldLinkBranchingObject::OsiOldLinkBranchingObject ( )`

7.125.2.2 `OsiOldLinkBranchingObject::OsiOldLinkBranchingObject ( OsiSolverInterface * solver, const OsiOldLink * originalObject, int way, double separator )`

7.125.2.3 `OsiOldLinkBranchingObject::OsiOldLinkBranchingObject ( const OsiOldLinkBranchingObject & )`

7.125.2.4 `virtual OsiOldLinkBranchingObject::~~OsiOldLinkBranchingObject ( ) [virtual]`

### 7.125.3 Member Function Documentation

7.125.3.1 `OsiOldLinkBranchingObject& OsiOldLinkBranchingObject::operator= ( const OsiOldLinkBranchingObject & rhs )`

7.125.3.2 `virtual OsiBranchingObject* OsiOldLinkBranchingObject::clone ( ) const [virtual]`

Clone.

7.125.3.3 `virtual double OsiOldLinkBranchingObject::branch ( OsiSolverInterface * solver ) [virtual]`

Does next branch and updates state.

7.125.3.4 `virtual void OsiOldLinkBranchingObject::print ( const OsiSolverInterface * solver = NULL ) [virtual]`

Print something about branch - only if log level high.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.126 OsiOneLink Class Reference

Define data for one link.

```
#include <CbcLinked.hpp>
```

### Public Member Functions

- [OsiOneLink](#) ( )
- [OsiOneLink](#) (const OsiSolverInterface \**solver*, int *xRow*, int *xColumn*, int [xyRow](#), const char \**functionString*)  
*Useful constructor -.*
- [OsiOneLink](#) (const [OsiOneLink](#) &)
- [OsiOneLink](#) & `operator=` (const [OsiOneLink](#) &*rhs*)
- `virtual ~OsiOneLink` ( )

### Public Attributes

- int [xRow\\_](#)  
*data*
- int [xColumn\\_](#)  
*Column which defines x.*



- int `xyRow`  
*Output row.*
- std::string `function_`  
*Function.*

### 7.126.1 Detailed Description

Define data for one link.

Definition at line 558 of file CbcLinked.hpp.

### 7.126.2 Constructor & Destructor Documentation

#### 7.126.2.1 OsiOneLink::OsiOneLink ( )

#### 7.126.2.2 OsiOneLink::OsiOneLink ( const OsiSolverInterface \* *solver*, int *xRow*, int *xColumn*, int *xyRow*, const char \* *functionString* )

Useful constructor -.

#### 7.126.2.3 OsiOneLink::OsiOneLink ( const OsiOneLink & )

#### 7.126.2.4 virtual OsiOneLink::~~OsiOneLink ( ) [virtual]

### 7.126.3 Member Function Documentation

#### 7.126.3.1 OsiOneLink& OsiOneLink::operator= ( const OsiOneLink & *rhs* )

### 7.126.4 Member Data Documentation

#### 7.126.4.1 int OsiOneLink::xRow\_

data

Row which defines x (if -1 then no x)

Definition at line 583 of file CbcLinked.hpp.

#### 7.126.4.2 int OsiOneLink::xColumn\_

Column which defines x.

Definition at line 585 of file CbcLinked.hpp.

#### 7.126.4.3 int OsiOneLink::xyRow

Output row.

Definition at line 587 of file CbcLinked.hpp.

#### 7.126.4.4 std::string OsiOneLink::function\_

Function.

Definition at line 589 of file CbcLinked.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

### 7.127 CbcGenCtlBlk::osiParamsInfo\_struct Struct Reference

Start and end of OsiSolverInterface parameters in parameter vector.

```
#include <CbcGenCtlBlk.hpp>
```

#### Public Attributes

- int [first\\_](#)
- int [last\\_](#)

#### 7.127.1 Detailed Description

Start and end of OsiSolverInterface parameters in parameter vector.

Definition at line 614 of file CbcGenCtlBlk.hpp.

#### 7.127.2 Member Data Documentation

##### 7.127.2.1 int CbcGenCtlBlk::osiParamsInfo\_struct::first\_

Definition at line 615 of file CbcGenCtlBlk.hpp.

##### 7.127.2.2 int CbcGenCtlBlk::osiParamsInfo\_struct::last\_

Definition at line 616 of file CbcGenCtlBlk.hpp.

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

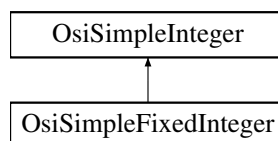
- [/home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp](#)

### 7.128 OsiSimpleFixedInteger Class Reference

Define a single integer class - but one where you keep branching until fixed even if satisfied.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for OsiSimpleFixedInteger:



#### Public Member Functions

- [OsiSimpleFixedInteger \(\)](#)  
*Default Constructor.*

- [OsiSimpleFixedInteger](#) (const OsiSolverInterface \*solver, int iColumn)  
*Useful constructor - passed solver index.*
- [OsiSimpleFixedInteger](#) (int iColumn, double lower, double upper)  
*Useful constructor - passed solver index and original bounds.*
- [OsiSimpleFixedInteger](#) (const OsiSimpleInteger &)  
*Useful constructor - passed simple integer.*
- [OsiSimpleFixedInteger](#) (const [OsiSimpleFixedInteger](#) &)  
*Copy constructor.*
- virtual OsiObject \* [clone](#) () const  
*Clone.*
- [OsiSimpleFixedInteger](#) & [operator=](#) (const [OsiSimpleFixedInteger](#) &rhs)  
*Assignment operator.*
- virtual [~OsiSimpleFixedInteger](#) ()  
*Destructor.*
- virtual double [infeasibility](#) (const OsiBranchingInformation \*info, int &whichWay) const  
*Infeasibility - large is 0.5.*
- virtual OsiBranchingObject \* [createBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way) const  
*Creates a branching object.*

### 7.128.1 Detailed Description

Define a single integer class - but one where you keep branching until fixed even if satisfied.

Definition at line 1089 of file CbcLinked.hpp.

### 7.128.2 Constructor & Destructor Documentation

#### 7.128.2.1 OsiSimpleFixedInteger::OsiSimpleFixedInteger ( )

Default Constructor.

#### 7.128.2.2 OsiSimpleFixedInteger::OsiSimpleFixedInteger ( const OsiSolverInterface \* solver, int iColumn )

Useful constructor - passed solver index.

#### 7.128.2.3 OsiSimpleFixedInteger::OsiSimpleFixedInteger ( int iColumn, double lower, double upper )

Useful constructor - passed solver index and original bounds.

#### 7.128.2.4 OsiSimpleFixedInteger::OsiSimpleFixedInteger ( const OsiSimpleInteger & )

Useful constructor - passed simple integer.

#### 7.128.2.5 OsiSimpleFixedInteger::OsiSimpleFixedInteger ( const OsiSimpleFixedInteger & )

Copy constructor.

#### 7.128.2.6 virtual OsiSimpleFixedInteger::~~OsiSimpleFixedInteger ( ) [virtual]

Destructor.

### 7.128.3 Member Function Documentation

7.128.3.1 `virtual OsiObject* OsiSimpleFixedInteger::clone ( ) const` `[virtual]`

Clone.

7.128.3.2 `OsiSimpleFixedInteger& OsiSimpleFixedInteger::operator= ( const OsiSimpleFixedInteger & rhs )`

Assignment operator.

7.128.3.3 `virtual double OsiSimpleFixedInteger::infeasibility ( const OsiBranchingInformation * info, int & whichWay ) const`  
`[virtual]`

Infeasibility - large is 0.5.

7.128.3.4 `virtual OsiBranchingObject* OsiSimpleFixedInteger::createBranch ( OsiSolverInterface * solver, const OsiBranchingInformation * info, int way ) const` `[virtual]`

Creates a branching object.

The preferred direction is set by `way`, 0 for down, 1 for up.

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

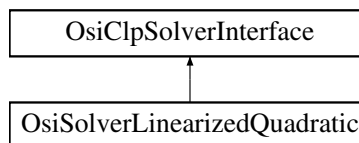
- </home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp>

## 7.129 OsiSolverLinearizedQuadratic Class Reference

This is to allow the user to replace `initialSolve` and `resolve`.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for `OsiSolverLinearizedQuadratic`:



### Public Member Functions

#### Solve methods

- `virtual void initialSolve ( )`  
*Solve initial LP relaxation.*

#### Constructors and destructors

- `OsiSolverLinearizedQuadratic ( )`  
*Default Constructor.*
- `OsiSolverLinearizedQuadratic (ClpSimplex *quadraticModel)`  
*Useful constructor (solution should be good)*
- `virtual OsiSolverInterface * clone (bool copyData=true) const`  
*Clone.*

- [OsiSolverLinearizedQuadratic](#) (const [OsiSolverLinearizedQuadratic](#) &)  
*Copy constructor.*
- [OsiSolverLinearizedQuadratic](#) & [operator=](#) (const [OsiSolverLinearizedQuadratic](#) &rhs)  
*Assignment operator.*
- virtual [~OsiSolverLinearizedQuadratic](#) ()  
*Destructor.*

### Sets and Gets

- double [bestObjectiveValue](#) () const  
*Objective value of best solution found internally.*
- const double \* [bestSolution](#) () const  
*Best solution found internally.*
- void [setSpecialOptions3](#) (int value)  
*Set special options.*
- int [specialOptions3](#) () const  
*Get special options.*
- ClpSimplex \* [quadraticModel](#) () const  
*Copy of quadratic model if one.*

### Protected Attributes

#### Private member data

- double [bestObjectiveValue\\_](#)  
*Objective value of best solution found internally.*
- ClpSimplex \* [quadraticModel\\_](#)  
*Copy of quadratic model if one.*
- double \* [bestSolution\\_](#)  
*Best solution found internally.*
- int [specialOptions3\\_](#)  
*0 bit (1) - don't do mini B&B 1 bit (2) - quadratic only in objective*

#### 7.129.1 Detailed Description

This is to allow the user to replace initialSolve and resolve.

Definition at line 1318 of file CbcLinked.hpp.

#### 7.129.2 Constructor & Destructor Documentation

##### 7.129.2.1 OsiSolverLinearizedQuadratic::OsiSolverLinearizedQuadratic ( )

Default Constructor.

##### 7.129.2.2 OsiSolverLinearizedQuadratic::OsiSolverLinearizedQuadratic ( ClpSimplex \* *quadraticModel* )

Useful constructor (solution should be good)

##### 7.129.2.3 OsiSolverLinearizedQuadratic::OsiSolverLinearizedQuadratic ( const OsiSolverLinearizedQuadratic & )

Copy constructor.

7.129.2.4 `virtual OsiSolverLinearizedQuadratic::~~OsiSolverLinearizedQuadratic ( ) [virtual]`

Destructor.

### 7.129.3 Member Function Documentation

7.129.3.1 `virtual void OsiSolverLinearizedQuadratic::initialSolve ( ) [virtual]`

Solve initial LP relaxation.

7.129.3.2 `virtual OsiSolverInterface* OsiSolverLinearizedQuadratic::clone ( bool copyData = true ) const [virtual]`

Clone.

7.129.3.3 `OsiSolverLinearizedQuadratic& OsiSolverLinearizedQuadratic::operator= ( const OsiSolverLinearizedQuadratic & rhs )`

Assignment operator.

7.129.3.4 `double OsiSolverLinearizedQuadratic::bestObjectiveValue ( ) const [inline]`

Objective value of best solution found internally.

Definition at line 1353 of file CbcLinked.hpp.

7.129.3.5 `const double* OsiSolverLinearizedQuadratic::bestSolution ( ) const [inline]`

Best solution found internally.

Definition at line 1357 of file CbcLinked.hpp.

7.129.3.6 `void OsiSolverLinearizedQuadratic::setSpecialOptions3 ( int value ) [inline]`

Set special options.

Definition at line 1361 of file CbcLinked.hpp.

7.129.3.7 `int OsiSolverLinearizedQuadratic::specialOptions3 ( ) const [inline]`

Get special options.

Definition at line 1365 of file CbcLinked.hpp.

7.129.3.8 `ClpSimplex* OsiSolverLinearizedQuadratic::quadraticModel ( ) const [inline]`

Copy of quadratic model if one.

Definition at line 1369 of file CbcLinked.hpp.

### 7.129.4 Member Data Documentation

7.129.4.1 `double OsiSolverLinearizedQuadratic::bestObjectiveValue_ [protected]`

Objective value of best solution found internally.

Definition at line 1385 of file CbcLinked.hpp.

#### 7.129.4.2 ClpSimplex\* OsiSolverLinearizedQuadratic::quadraticModel\_ [protected]

Copy of quadratic model if one.

Definition at line 1387 of file CbcLinked.hpp.

#### 7.129.4.3 double\* OsiSolverLinearizedQuadratic::bestSolution\_ [protected]

Best solution found internally.

Definition at line 1389 of file CbcLinked.hpp.

#### 7.129.4.4 int OsiSolverLinearizedQuadratic::specialOptions3\_ [protected]

0 bit (1) - don't do mini B&B 1 bit (2) - quadratic only in objective

Definition at line 1394 of file CbcLinked.hpp.

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

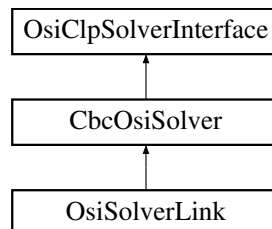
- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.130 OsiSolverLink Class Reference

This is to allow the user to replace initialSolve and resolve This version changes coefficients.

```
#include <CbcLinked.hpp>
```

Inheritance diagram for OsiSolverLink:



### Public Member Functions

#### Solve methods

- virtual void [initialSolve](#) ()  
*Solve initial LP relaxation.*
- virtual void [resolve](#) ()  
*Resolve an LP relaxation after problem modification.*
- virtual int [fathom](#) (bool allFixed)  
*Problem specific Returns -1 if node fathomed and no solution 0 if did nothing 1 if node fathomed and solution allFixed is true if all LinkedBound variables are fixed.*
- double \* [nonlinearSLP](#) (int numberPasses, double deltaTolerance)  
*Solves nonlinear problem from CoinModel using SLP - may be used as crash for other algorithms when number of iterations small.*
- double [linearizedBAB](#) (CglStored \*cut)  
*Solve linearized quadratic objective branch and bound.*
- double \* [heuristicSolution](#) (int numberPasses, double deltaTolerance, int mode)

*Solves nonlinear problem from CoinModel using SLP - and then tries to get heuristic solution Returns solution array mode - 0 just get continuous 1 round and try normal bab 2 use defaultBound\_ to bound integer variables near current solution.*

- int **doAOCuts** (CglTemporary \*cutGen, const double \*solution, const double \*solution2)  
*Do OA cuts.*

### Constructors and destructors

- **OsiSolverLink** ()  
*Default Constructor.*
- **OsiSolverLink** (CoinModel &modelObject)  
*This creates from a coinModel object.*
- void **load** (CoinModel &modelObject, bool tightenBounds=false, int logLevel=1)
- virtual OsiSolverInterface \* **clone** (bool copyData=true) const  
*Clone.*
- **OsiSolverLink** (const **OsiSolverLink** &)  
*Copy constructor.*
- **OsiSolverLink** & **operator=** (const **OsiSolverLink** &rhs)  
*Assignment operator.*
- virtual ~**OsiSolverLink** ()  
*Destructor.*

### Sets and Gets

- void **addBoundModifier** (bool upperBoundAffected, bool useUpperBound, int whichVariable, int whichVariable-Affected, double multiplier=1.0)  
*Add a bound modifier.*
- int **updateCoefficients** (ClpSimplex \*solver, CoinPackedMatrix \*matrix)  
*Update coefficients - returns number updated if in updating mode.*
- void **analyzeObjects** ()  
*Analyze constraints to see which are convex (quadratic)*
- void **addTighterConstraints** ()  
*Add reformulated bilinear constraints.*
- double **bestObjectiveValue** () const  
*Objective value of best solution found internally.*
- void **setBestObjectiveValue** (double value)  
*Set objective value of best solution found internally.*
- const double \* **bestSolution** () const  
*Best solution found internally.*
- void **setBestSolution** (const double \*solution, int numberColumns)  
*Set best solution found internally.*
- void **setSpecialOptions2** (int value)  
*Set special options.*
- void **sayConvex** (bool convex)  
*Say convex (should work it out) - if convex false then strictly concave.*
- int **specialOptions2** () const  
*Get special options.*
- CoinPackedMatrix \* **cleanMatrix** () const  
*Clean copy of matrix So we can add rows.*
- CoinPackedMatrix \* **originalRowCopy** () const  
*Row copy of matrix Just genuine columns and rows Linear part.*
- ClpSimplex \* **quadraticModel** () const  
*Copy of quadratic model if one.*
- CoinPackedMatrix \* **quadraticRow** (int rowNumber, double \*linear) const  
*Gets correct form for a quadratic row - user to delete.*



- double [defaultMeshSize](#) () const  
*Default meshSize.*
- void [setDefaultMeshSize](#) (double value)
- double [defaultBound](#) () const  
*Default maximumbound.*
- void [setDefaultBound](#) (double value)
- void [setIntegerPriority](#) (int value)  
*Set integer priority.*
- int [integerPriority](#) () const  
*Get integer priority.*
- int [objectiveVariable](#) () const  
*Objective transfer variable if one.*
- void [setBiLinearPriority](#) (int value)  
*Set biLinear priority.*
- int [biLinearPriority](#) () const  
*Get biLinear priority.*
- const CoinModel \* [coinModel](#) () const  
*Return CoinModel.*
- void [setBiLinearPriorities](#) (int value, double meshSize=1.0)  
*Set all biLinear priorities on x-x variables.*
- void [setBranchingStrategyOnVariables](#) (int strategyValue, int priorityValue=-1, int mode=7)  
*Set options and priority on all or some biLinear variables 1 - on I-I 2 - on I-x 4 - on x-x or combinations.*
- void [setMeshSizes](#) (double value)  
*Set all mesh sizes on x-x variables.*
- void [setFixedPriority](#) (int priorityValue)  
*Two tier integer problem where when set of variables with priority less than this are fixed the problem becomes an easier integer problem.*

## Protected Member Functions

### functions

- void [gutsOfDestructor](#) (bool justNullify=false)  
*Do real work of initialize.*
- void [gutsOfCopy](#) (const [OsiSolverLink](#) &rhs)  
*Do real work of copy.*

## Protected Attributes

### Private member data

- CoinPackedMatrix \* [matrix\\_](#)  
*Clean copy of matrix Marked coefficients will be multiplied by L or U.*
- CoinPackedMatrix \* [originalRowCopy\\_](#)  
*Row copy of matrix Just genuine columns and rows.*
- ClpSimplex \* [quadraticModel\\_](#)  
*Copy of quadratic model if one.*
- int [numberNonLinearRows\\_](#)  
*Number of rows with nonLinearities.*
- int \* [startNonLinear\\_](#)  
*Starts of lists.*
- int \* [rowNonLinear\\_](#)  
*Row number for a list.*
- int \* [convex\\_](#)

- Indicator whether is convex, concave or neither -1 concave, 0 neither, +1 convex.
- int \* [whichNonLinear\\_](#)  
Indices in a list/row.
- CoinModel [coinModel\\_](#)  
Model in CoinModel format.
- int [numberVariables\\_](#)  
Number of variables in tightening phase.
- [OsiLinkedBound](#) \* [info\\_](#)  
Information.
- int [specialOptions2\\_](#)  
0 bit (1) - call fathom (may do mini B&B) 1 bit (2) - quadratic only in objective (add OA cuts) 2 bit (4) - convex 3 bit (8) - try adding OA cuts 4 bit (16) - add linearized constraints
- int [objectiveRow\\_](#)  
Objective transfer row if one.
- int [objectiveVariable\\_](#)  
Objective transfer variable if one.
- double [bestObjectiveValue\\_](#)  
Objective value of best solution found internally.
- double [defaultMeshSize\\_](#)  
Default mesh.
- double [defaultBound\\_](#)  
Default maximum bound.
- double \* [bestSolution\\_](#)  
Best solution found internally.
- int [integerPriority\\_](#)  
Priority for integers.
- int [biLinearPriority\\_](#)  
Priority for bilinear.
- int [numberFix\\_](#)  
Number of variables which when fixed help.
- int \* [fixVariables\\_](#)  
list of fixed variables

### 7.130.1 Detailed Description

This is to allow the user to replace initialSolve and resolve This version changes coefficients.

Definition at line 29 of file CbcLinked.hpp.

### 7.130.2 Constructor & Destructor Documentation

#### 7.130.2.1 OsiSolverLink::OsiSolverLink ( )

Default Constructor.

#### 7.130.2.2 OsiSolverLink::OsiSolverLink ( CoinModel & modelObject )

This creates from a coinModel object.

if errors.then number of sets is -1

This creates linked ordered sets information. It assumes -

for product terms syntax is yy\*f(zz) also just f(zz) is allowed and even a constant

modelObject not const as may be changed as part of process.

**7.130.2.3 OsiSolverLink::OsiSolverLink ( const OsiSolverLink & )**

Copy constructor.

**7.130.2.4 virtual OsiSolverLink::~~OsiSolverLink ( ) [virtual]**

Destructor.

**7.130.3 Member Function Documentation****7.130.3.1 virtual void OsiSolverLink::initialSolve ( ) [virtual]**

Solve initial LP relaxation.

**7.130.3.2 virtual void OsiSolverLink::resolve ( ) [virtual]**

Resolve an LP relaxation after problem modification.

**7.130.3.3 virtual int OsiSolverLink::fathom ( bool *allFixed* ) [virtual]**

Problem specific Returns -1 if node fathomed and no solution 0 if did nothing 1 if node fathomed and solution *allFixed* is true if all *LinkedBound* variables are fixed.

**7.130.3.4 double\* OsiSolverLink::nonlinearSLP ( int *numberPasses*, double *deltaTolerance* )**

Solves nonlinear problem from *CoinModel* using SLP - may be used as crash for other algorithms when number of iterations small.

Also exits if all problematical variables are changing less than *deltaTolerance* Returns solution array

**7.130.3.5 double OsiSolverLink::linearizedBAB ( CglStored \* *cut* )**

Solve linearized quadratic objective branch and bound.

Return cutoff and OA cut

**7.130.3.6 double\* OsiSolverLink::heuristicSolution ( int *numberPasses*, double *deltaTolerance*, int *mode* )**

Solves nonlinear problem from *CoinModel* using SLP - and then tries to get heuristic solution Returns solution array  
mode - 0 just get continuous 1 round and try normal bab 2 use *defaultBound\_* to bound integer variables near current solution.

**7.130.3.7 int OsiSolverLink::doAOCuts ( CglTemporary \* *cutGen*, const double \* *solution*, const double \* *solution2* )**

Do OA cuts.

**7.130.3.8 void OsiSolverLink::load ( CoinModel & *modelObject*, bool *tightenBounds* = false, int *logLevel* = 1 )****7.130.3.9 virtual OsiSolverInterface\* OsiSolverLink::clone ( bool *copyData* = true ) const [virtual]**

Clone.

Reimplemented from [CbcOsiSolver](#).

**7.130.3.10 OsiSolverLink& OsiSolverLink::operator= ( const OsiSolverLink & *rhs* )**

Assignment operator.

7.130.3.11 `void OsiSolverLink::addBoundModifier ( bool upperBoundAffected, bool useUpperBound, int whichVariable, int whichVariableAffected, double multiplier = 1.0 )`

Add a bound modifier.

7.130.3.12 `int OsiSolverLink::updateCoefficients ( ClpSimplex * solver, CoinPackedMatrix * matrix )`

Update coefficients - returns number updated if in updating mode.

7.130.3.13 `void OsiSolverLink::analyzeObjects ( )`

Analyze constraints to see which are convex (quadratic)

7.130.3.14 `void OsiSolverLink::addTighterConstraints ( )`

Add reformulated bilinear constraints.

7.130.3.15 `double OsiSolverLink::bestObjectiveValue ( ) const [inline]`

Objective value of best solution found internally.

Definition at line 122 of file CbcLinked.hpp.

7.130.3.16 `void OsiSolverLink::setBestObjectiveValue ( double value ) [inline]`

Set objective value of best solution found internally.

Definition at line 126 of file CbcLinked.hpp.

7.130.3.17 `const double* OsiSolverLink::bestSolution ( ) const [inline]`

Best solution found internally.

Definition at line 130 of file CbcLinked.hpp.

7.130.3.18 `void OsiSolverLink::setBestSolution ( const double * solution, int numberColumns )`

Set best solution found internally.

7.130.3.19 `void OsiSolverLink::setSpecialOptions2 ( int value ) [inline]`

Set special options.

Definition at line 136 of file CbcLinked.hpp.

7.130.3.20 `void OsiSolverLink::sayConvex ( bool convex )`

Say convex (should work it out) - if convex false then strictly concave.

7.130.3.21 `int OsiSolverLink::specialOptions2 ( ) const [inline]`

Get special options.

Definition at line 142 of file CbcLinked.hpp.

7.130.3.22 `CoinPackedMatrix* OsiSolverLink::cleanMatrix ( ) const [inline]`

Clean copy of matrix So we can add rows.

Definition at line 148 of file CbcLinked.hpp.

**7.130.3.23** `CoinPackedMatrix* OsiSolverLink::originalRowCopy ( ) const` `[inline]`

Row copy of matrix Just genuine columns and rows Linear part.

Definition at line 155 of file CbcLinked.hpp.

**7.130.3.24** `ClpSimplex* OsiSolverLink::quadraticModel ( ) const` `[inline]`

Copy of quadratic model if one.

Definition at line 159 of file CbcLinked.hpp.

**7.130.3.25** `CoinPackedMatrix* OsiSolverLink::quadraticRow ( int rowNumber, double * linear ) const`

Gets correct form for a quadratic row - user to delete.

**7.130.3.26** `double OsiSolverLink::defaultMeshSize ( ) const` `[inline]`

Default meshSize.

Definition at line 165 of file CbcLinked.hpp.

**7.130.3.27** `void OsiSolverLink::setDefaultMeshSize ( double value )` `[inline]`

Definition at line 168 of file CbcLinked.hpp.

**7.130.3.28** `double OsiSolverLink::defaultBound ( ) const` `[inline]`

Default maximumbound.

Definition at line 172 of file CbcLinked.hpp.

**7.130.3.29** `void OsiSolverLink::setDefaultBound ( double value )` `[inline]`

Definition at line 175 of file CbcLinked.hpp.

**7.130.3.30** `void OsiSolverLink::setIntegerPriority ( int value )` `[inline]`

Set integer priority.

Definition at line 179 of file CbcLinked.hpp.

**7.130.3.31** `int OsiSolverLink::integerPriority ( ) const` `[inline]`

Get integer priority.

Definition at line 183 of file CbcLinked.hpp.

**7.130.3.32** `int OsiSolverLink::objectiveVariable ( ) const` `[inline]`

Objective transfer variable if one.

Definition at line 187 of file CbcLinked.hpp.

**7.130.3.33** `void OsiSolverLink::setBiLinearPriority ( int value )` `[inline]`

Set biLinear priority.

Definition at line 191 of file CbcLinked.hpp.

**7.130.3.34** `int OsiSolverLink::biLinearPriority ( ) const [inline]`

Get biLinear priority.

Definition at line 195 of file CbcLinked.hpp.

**7.130.3.35** `const CoinModel* OsiSolverLink::coinModel ( ) const [inline]`

Return CoinModel.

Definition at line 199 of file CbcLinked.hpp.

**7.130.3.36** `void OsiSolverLink::setBiLinearPriorities ( int value, double meshSize = 1.0 )`

Set all biLinear priorities on x-x variables.

**7.130.3.37** `void OsiSolverLink::setBranchingStrategyOnVariables ( int strategyValue, int priorityValue = -1, int mode = 7 )`

Set options and priority on all or some biLinear variables 1 - on I-I 2 - on I-x 4 - on x-x or combinations.

-1 means leave (for priority value and strategy value)

**7.130.3.38** `void OsiSolverLink::setMeshSizes ( double value )`

Set all mesh sizes on x-x variables.

**7.130.3.39** `void OsiSolverLink::setFixedPriority ( int priorityValue )`

Two tier integer problem where when set of variables with priority less than this are fixed the problem becomes an easier integer problem.

**7.130.3.40** `void OsiSolverLink::gutsOfDestructor ( bool justNullify = false ) [protected]`

Do real work of initialize.

Do real work of delete

**7.130.3.41** `void OsiSolverLink::gutsOfCopy ( const OsiSolverLink & rhs ) [protected]`

Do real work of copy.

#### 7.130.4 Member Data Documentation

**7.130.4.1** `CoinPackedMatrix* OsiSolverLink::matrix_ [protected]`

Clean copy of matrix Marked coefficients will be multiplied by L or U.

Definition at line 241 of file CbcLinked.hpp.

**7.130.4.2** `CoinPackedMatrix* OsiSolverLink::originalRowCopy_ [protected]`

Row copy of matrix Just genuine columns and rows.

Definition at line 245 of file CbcLinked.hpp.

**7.130.4.3** `ClpSimplex* OsiSolverLink::quadraticModel_ [protected]`

Copy of quadratic model if one.

Definition at line 247 of file CbcLinked.hpp.

**7.130.4.4** `int OsiSolverLink::numberNonLinearRows_` [protected]

Number of rows with nonLinearities.

Definition at line 249 of file CbcLinked.hpp.

**7.130.4.5** `int* OsiSolverLink::startNonLinear_` [protected]

Starts of lists.

Definition at line 251 of file CbcLinked.hpp.

**7.130.4.6** `int* OsiSolverLink::rowNonLinear_` [protected]

Row number for a list.

Definition at line 253 of file CbcLinked.hpp.

**7.130.4.7** `int* OsiSolverLink::convex_` [protected]

Indicator whether is convex, concave or neither -1 concave, 0 neither, +1 convex.

Definition at line 257 of file CbcLinked.hpp.

**7.130.4.8** `int* OsiSolverLink::whichNonLinear_` [protected]

Indices in a list/row.

Definition at line 259 of file CbcLinked.hpp.

**7.130.4.9** `CoinModel OsiSolverLink::coinModel_` [protected]

Model in CoinModel format.

Definition at line 261 of file CbcLinked.hpp.

**7.130.4.10** `int OsiSolverLink::numberVariables_` [protected]

Number of variables in tightening phase.

Definition at line 263 of file CbcLinked.hpp.

**7.130.4.11** `OsiLinkedBound* OsiSolverLink::info_` [protected]

Information.

Definition at line 265 of file CbcLinked.hpp.

**7.130.4.12** `int OsiSolverLink::specialOptions2_` [protected]

0 bit (1) - call fathom (may do mini B&B) 1 bit (2) - quadratic only in objective (add OA cuts) 2 bit (4) - convex 3 bit (8) - try adding OA cuts 4 bit (16) - add linearized constraints

Definition at line 273 of file CbcLinked.hpp.

**7.130.4.13** `int OsiSolverLink::objectiveRow_` [protected]

Objective transfer row if one.

Definition at line 275 of file CbcLinked.hpp.

**7.130.4.14** `int OsiSolverLink::objectiveVariable_` `[protected]`

Objective transfer variable if one.

Definition at line 277 of file CbcLinked.hpp.

**7.130.4.15** `double OsiSolverLink::bestObjectiveValue_` `[protected]`

Objective value of best solution found internally.

Definition at line 279 of file CbcLinked.hpp.

**7.130.4.16** `double OsiSolverLink::defaultMeshSize_` `[protected]`

Default mesh.

Definition at line 281 of file CbcLinked.hpp.

**7.130.4.17** `double OsiSolverLink::defaultBound_` `[protected]`

Default maximum bound.

Definition at line 283 of file CbcLinked.hpp.

**7.130.4.18** `double* OsiSolverLink::bestSolution_` `[protected]`

Best solution found internally.

Definition at line 285 of file CbcLinked.hpp.

**7.130.4.19** `int OsiSolverLink::integerPriority_` `[protected]`

Priority for integers.

Definition at line 287 of file CbcLinked.hpp.

**7.130.4.20** `int OsiSolverLink::biLinearPriority_` `[protected]`

Priority for bilinear.

Definition at line 289 of file CbcLinked.hpp.

**7.130.4.21** `int OsiSolverLink::numberFix_` `[protected]`

Number of variables which when fixed help.

Definition at line 291 of file CbcLinked.hpp.

**7.130.4.22** `int* OsiSolverLink::fixVariables_` `[protected]`

list of fixed variables

Definition at line 293 of file CbcLinked.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

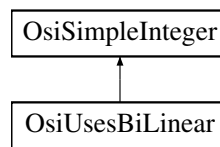
**7.131 OsiUsesBiLinear Class Reference**

Define a single variable class which is involved with [OsiBiLinear](#) objects.



```
#include <CbcbLinked.hpp>
```

Inheritance diagram for OsiUsesBiLinear:



### Public Member Functions

- [OsiUsesBiLinear](#) ()  
*Default Constructor.*
- [OsiUsesBiLinear](#) (const OsiSolverInterface \*solver, int iColumn, int type)  
*Useful constructor - passed solver index.*
- [OsiUsesBiLinear](#) (int iColumn, double lower, double upper, int type)  
*Useful constructor - passed solver index and original bounds.*
- [OsiUsesBiLinear](#) (const OsiSimpleInteger &rhs, int type)  
*Useful constructor - passed simple integer.*
- [OsiUsesBiLinear](#) (const [OsiUsesBiLinear](#) &rhs)  
*Copy constructor.*
- virtual OsiObject \* [clone](#) () const  
*Clone.*
- [OsiUsesBiLinear](#) & [operator=](#) (const [OsiUsesBiLinear](#) &rhs)  
*Assignment operator.*
- virtual [~OsiUsesBiLinear](#) ()  
*Destructor.*
- virtual double [infeasibility](#) (const OsiBranchingInformation \*info, int &whichWay) const  
*Infeasibility - large is 0.5.*
- virtual OsiBranchingObject \* [createBranch](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info, int way) const  
*Creates a branching object.*
- virtual double [feasibleRegion](#) (OsiSolverInterface \*solver, const OsiBranchingInformation \*info) const  
*Set bounds to fix the variable at the current value.*
- void [addBiLinearObjects](#) (OsiSolverLink \*solver)  
*Add all bi-linear objects.*

### Protected Attributes

- int [numberBiLinear\\_](#)  
*data Number of bilinear objects (maybe could be more general)*
- int [type\\_](#)  
*Type of variable - 0 continuous, 1 integer.*
- OsiObject \*\* [objects\\_](#)  
*Objects.*

### 7.131.1 Detailed Description

Define a single variable class which is involved with [OsiBiLinear](#) objects.

This is used so can make better decision on where to branch as it can look at all objects.

This version sees if it can re-use code from [OsiSimpleInteger](#) even if not an integer variable. If not then need to duplicate code.

Definition at line 1139 of file [CbcLinked.hpp](#).

### 7.131.2 Constructor & Destructor Documentation

#### 7.131.2.1 [OsiUsesBiLinear::OsiUsesBiLinear](#) ( )

Default Constructor.

#### 7.131.2.2 [OsiUsesBiLinear::OsiUsesBiLinear](#) ( const [OsiSolverInterface](#) \* *solver*, int *iColumn*, int *type* )

Useful constructor - passed solver index.

#### 7.131.2.3 [OsiUsesBiLinear::OsiUsesBiLinear](#) ( int *iColumn*, double *lower*, double *upper*, int *type* )

Useful constructor - passed solver index and original bounds.

#### 7.131.2.4 [OsiUsesBiLinear::OsiUsesBiLinear](#) ( const [OsiSimpleInteger](#) & *rhs*, int *type* )

Useful constructor - passed simple integer.

#### 7.131.2.5 [OsiUsesBiLinear::OsiUsesBiLinear](#) ( const [OsiUsesBiLinear](#) & *rhs* )

Copy constructor.

#### 7.131.2.6 [virtual OsiUsesBiLinear::~OsiUsesBiLinear](#) ( ) [\[virtual\]](#)

Destructor.

### 7.131.3 Member Function Documentation

#### 7.131.3.1 [virtual OsiObject\\* OsiUsesBiLinear::clone](#) ( ) const [\[virtual\]](#)

Clone.

#### 7.131.3.2 [OsiUsesBiLinear& OsiUsesBiLinear::operator=](#) ( const [OsiUsesBiLinear](#) & *rhs* )

Assignment operator.

#### 7.131.3.3 [virtual double OsiUsesBiLinear::infeasibility](#) ( const [OsiBranchingInformation](#) \* *info*, int & *whichWay* ) const [\[virtual\]](#)

Infeasibility - large is 0.5.

#### 7.131.3.4 [virtual OsiBranchingObject\\* OsiUsesBiLinear::createBranch](#) ( [OsiSolverInterface](#) \* *solver*, const [OsiBranchingInformation](#) \* *info*, int *way* ) const [\[virtual\]](#)

Creates a branching object.

The preferred direction is set by `way`, 0 for down, 1 for up.

**7.131.3.5** `virtual double OsiUsesBiLinear::feasibleRegion ( OsiSolverInterface * solver, const OsiBranchingInformation * info )`  
`const [virtual]`

Set bounds to fix the variable at the current value.

Given an current value, set the lower and upper bounds to fix the variable. Returns amount it had to move variable.

**7.131.3.6** `void OsiUsesBiLinear::addBiLinearObjects ( OsiSolverLink * solver )`

Add all bi-linear objects.

#### 7.131.4 Member Data Documentation

**7.131.4.1** `int OsiUsesBiLinear::numberBiLinear_ [protected]`

data Number of bilinear objects (maybe could be more general)

Definition at line 1190 of file `CbcLinked.hpp`.

**7.131.4.2** `int OsiUsesBiLinear::type_ [protected]`

Type of variable - 0 continuous, 1 integer.

Definition at line 1192 of file `CbcLinked.hpp`.

**7.131.4.3** `OsiObject** OsiUsesBiLinear::objects_ [protected]`

Objects.

Definition at line 1194 of file `CbcLinked.hpp`.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp](#)

## 7.132 PseudoReducedCost Struct Reference

```
#include <CbcHeuristicDive.hpp>
```

#### Public Attributes

- `int` [var](#)
- `double` [pseudoRedCost](#)

#### 7.132.1 Detailed Description

Definition at line 12 of file `CbcHeuristicDive.hpp`.

#### 7.132.2 Member Data Documentation

**7.132.2.1** `int PseudoReducedCost::var`

Definition at line 13 of file `CbcHeuristicDive.hpp`.

### 7.132.2.2 double PseudoReducedCost::pseudoRedCost

Definition at line 14 of file CbcHeuristicDive.hpp.

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

- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicDive.hpp](#)

## 8 File Documentation

### 8.1 /home/ted/COIN/trunk/Cbc/src/Cbc\_ampl.h File Reference

#### Classes

- struct [ampl\\_info](#)

#### Functions

- int [readAmpl](#) ([ampl\\_info](#) \*info, int argc, char \*\*argv, void \*\*coinModel)
- void [freeArrays1](#) ([ampl\\_info](#) \*info)
- void [freeArrays2](#) ([ampl\\_info](#) \*info)
- void [freeArgs](#) ([ampl\\_info](#) \*info)
- void [writeAmpl](#) ([ampl\\_info](#) \*info)
- int [ampl\\_obj\\_prec](#) ()

#### 8.1.1 Function Documentation

8.1.1.1 int [readAmpl](#) ( [ampl\\_info](#) \* *info*, int *argc*, char \*\* *argv*, void \*\* *coinModel* )

8.1.1.2 void [freeArrays1](#) ( [ampl\\_info](#) \* *info* )

8.1.1.3 void [freeArrays2](#) ( [ampl\\_info](#) \* *info* )

8.1.1.4 void [freeArgs](#) ( [ampl\\_info](#) \* *info* )

8.1.1.5 void [writeAmpl](#) ( [ampl\\_info](#) \* *info* )

8.1.1.6 int [ampl\\_obj\\_prec](#) ( )

### 8.2 /home/ted/COIN/trunk/Cbc/src/Cbc\_C\_Interface.h File Reference

```
#include "Coin_C_defines.h"
```

#### Functions

##### Load model - loads some stuff and initializes others

- COINLIBAPI void [COINLINKAGE Cbc\\_loadProblem](#) (Cbc\_Model \*model, 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)

- COINLIBAPI void [COINLINKAGE Cbc\\_addRows](#) (Cbc\_Model \*model, const int number, const double \*rowLower, const double \*rowUpper, const int \*rowStarts, const int \*columns, const double \*elements)  
*Add rows.*
- COINLIBAPI void [COINLINKAGE Cbc\\_addColumns](#) (Cbc\_Model \*model, int number, const double \*columnLower, const double \*columnUpper, const double \*objective, const int \*columnStarts, const int \*rows, const double \*elements)  
*Add columns.*
- COINLIBAPI void [COINLINKAGE Cbc\\_copyNames](#) (Cbc\_Model \*model, const char \*const \*rowNamesIn, const char \*const \*columnNamesIn)  
*Copies in names.*

### Message handling. Call backs are handled by ONE function

- COINLIBAPI void [COINLINKAGE Cbc\\_registerCallBack](#) (Cbc\_Model \*model, cbc\_callback userCallBack)  
*Pass in Callback function.*

### gets and sets - some synonyms

- COINLIBAPI void [COINLINKAGE Cbc\\_addSOS\\_Dense](#) (Cbc\_Model \*model, int numObjects, const int \*len, const int \*const \*which, const double \*weights, const int type)  
*Add SOS constraints to the model using dense matrix.*
- COINLIBAPI void [COINLINKAGE Cbc\\_addSOS\\_Sparse](#) (Cbc\_Model \*model, const int \*rowStarts, const int \*rowIndices, const double \*weights, const int type)  
*Add SOS constraints to the model using row-order matrix.*

### Constructors and destructor

This is a first "C" interface to Cbc.

It is mostly similar to the "C" interface to Clp and was contributed by Bob Enriken. These do not have an exact analogue in C++. The user does not need to know structure of Cbc\_Model.

For all functions outside this group there is an exact C++ analogue created by taking the first parameter out, removing the Cbc\_ from name and applying the method to an object of type ClpSimplex.

- COINLIBAPI Cbc\_Model \* [COINLINKAGE](#)  
*Default Cbc\_Model constructor.*
- COINLIBAPI double [COINLINKAGE Cbc\\_getVersion](#) ()  
*Version.*

### 8.2.1 Function Documentation

#### 8.2.1.1 COINLIBAPI double COINLINKAGE Cbc\_getVersion ( )

Version.

#### 8.2.1.2 COINLIBAPI void COINLINKAGE Cbc\_loadProblem ( Cbc\_Model \* model, 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 )

#### 8.2.1.3 COINLIBAPI void COINLINKAGE Cbc\_addRows ( Cbc\_Model \* model, const int number, const double \* rowLower, const double \* rowUpper, const int \* rowStarts, const int \* columns, const double \* elements )

Add rows.

**8.2.1.4 COINLIBAPI void COINLINKAGE Cbc\_addColumns ( Cbc\_Model \* *model*, int *number*, const double \* *columnLower*, const double \* *columnUpper*, const double \* *objective*, const int \* *columnStarts*, const int \* *rows*, const double \* *elements* )**

Add columns.

**8.2.1.5 COINLIBAPI void COINLINKAGE Cbc\_copyNames ( Cbc\_Model \* *model*, const char \*const \* *rowNamesIn*, const char \*const \* *columnNamesIn* )**

Copies in names.

**8.2.1.6 COINLIBAPI void COINLINKAGE Cbc\_registerCallBack ( Cbc\_Model \* *model*, cbc\_callback *userCallBack* )**

Pass in Callback function.

Message numbers up to 1000000 are Clp, Coin ones have 1000000 added

**8.2.1.7 COINLIBAPI void COINLINKAGE Cbc\_addSOS\_Dense ( Cbc\_Model \* *model*, int *numObjects*, const int \* *len*, const int \*const \* *which*, const double \* *weights*, const int *type* )**

Add SOS constraints to the model using dense matrix.

**8.2.1.8 COINLIBAPI void COINLINKAGE Cbc\_addSOS\_Sparse ( Cbc\_Model \* *model*, const int \* *rowStarts*, const int \* *rowIndices*, const double \* *weights*, const int *type* )**

Add SOS constraints to the model using row-order matrix.

## 8.2.2 Variable Documentation

### 8.2.2.1 COINLIBAPI int COINLINKAGE

Default Cbc\_Model constructor.

Primal algorithm - see ClpSimplexPrimal.hpp for method.

Dual algorithm - see ClpSimplexDual.hpp for method.

Primal initial solve.

Dual initial solve.

Print the solution.

Delete all object information.

Set this the variable to be continuous.

Return a copy of this model.

Number of nodes explored in B&B tree.

Return CPU time.

Determine whether the variable at location *i* is integer restricted.

Print the model.

Iteration limit reached?

Is the given dual objective limit reached?

Is the given primal objective limit reached?

Is dual infeasibility proven?

Is primal infeasibility proven?

Is optimality proven?

Are there a numerical difficulties?

Just check solution (for external use) - sets sum of infeasibilities etc.

Restore model from file, returns 0 if success, deletes current model.

Save model to file, returns 0 if success.

Number of primal infeasibilities.

Sum of primal infeasibilities.

Number of dual infeasibilities.

Sum of dual infeasibilities.

Set algorithm.

Current (or last) algorithm.

Perturbation: 50 - switch on perturbation 100 - auto perturb if takes too long (1.0e-6 largest nonzero) 101 - we are perturbed 102 - don't try perturbing again default is 100 others are for playing.

Infeasibility cost.

Dual bound.

If problem is dual feasible.

If problem is primal feasible.

Crash - at present just aimed at dual, returns -2 if dual preferred and crash basis created -1 if dual preferred and all slack basis preferred 0 if basis going in was not all slack 1 if primal preferred and all slack basis preferred 2 if primal preferred and crash basis created.

Gets scalingFlag.

Sets or unsets scaling, 0 -off, 1 equilibrium, 2 geometric, 3, auto, 4 dynamic(later)

General solve algorithm which can do presolve.

Fill in array (at least lengthNames+1 long) with a column name.

Fill in array (at least lengthNames+1 long) with a row name.

length of names (0 means no names)

Amount of print out: 0 - none 1 - just final 2 - just factorizations 3 - as 2 plus a bit more 4 - verbose above that 8,16,32 etc just for selective debug.

Unset Callback function.

User pointer for whatever reason.

Copy in status vector.

Return address of status array (char[numberRows+numberColumns])

See if status array exists (partly for OsiClip)

Infeasibility/unbounded ray (NULL returned if none/wrong) Up to user to use delete [] on these arrays.

Objective value.

Element values in matrix.

Column vector lengths in matrix.

Row indices in matrix.

Column starts in matrix.

Number of elements in matrix.

Column Upper.

Column Lower.

Objective.

Row upper.

Row lower.

Reduced costs.

Dual row solution.

Primal column solution.

Primal row solution.

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

Secondary status of problem - may get extended 0 - none 1 - primal infeasible because dual limit reached 2 - scaled problem optimal - unscaled has primal infeasibilities 3 - scaled problem optimal - unscaled has dual infeasibilities 4 - scaled problem optimal - unscaled has both dual and primal infeasibilities.

Set problem status.

Status of problem: 0 - optimal 1 - primal infeasible 2 - dual infeasible 3 - stopped on iterations etc 4 - stopped due to errors.

Returns true if hit maximum iterations (or time)

Maximum time in seconds (from when set called)

Maximum number of nodes.

Maximum number of iterations.

Number of iterations.

Sets problem name.

Fills in array with problem name.

Objective offset.

Dual objective limit.

Dual tolerance to use.

Primal tolerance to use.

Number of columns.

Number of rows.

Drops names - makes lengthnames 0 and names empty.

Deletes columns.

Deletes rows.

Resizes rim part of model.

Drop integer informations.

Copy in integer information.



Integer information.

Write an mps file from the given filename.

Read an mps file from the given filename.

Cbc\_Model Destructor.

array must be a null-terminated string.

See ClpSolve.hpp for options

if gap between bounds  $\leq$  "gap" variables can be flipped

If "pivot" is 0 No pivoting (so will just be choice of algorithm) 1 Simple pivoting e.g. gub 2 Mini iterations

This is designed for use outside algorithms so does not save iterating arrays etc. It does not save any messaging information. Does not save scaling values. It does not know about all types of virtual functions.

Definition at line 39 of file Cbc\_C\_Interface.h.

### 8.3 /home/ted/COIN/trunk/Cbc/src/CbcBranchActual.hpp File Reference

```
#include "CbcBranchBase.hpp"
#include "CoinPackedMatrix.hpp"
#include "CbcClique.hpp"
#include "CbcSOS.hpp"
#include "CbcSimpleInteger.hpp"
#include "CbcNWay.hpp"
#include "CbcSimpleIntegerPseudoCost.hpp"
#include "CbcBranchDefaultDecision.hpp"
#include "CbcFollowOn.hpp"
#include "CbcFixVariable.hpp"
#include "CbcDummyBranchingObject.hpp"
#include "CbcGeneral.hpp"
#include "CbcGeneralDepth.hpp"
#include "CbcSubProblem.hpp"
```

### 8.4 /home/ted/COIN/trunk/Cbc/src/CbcBranchAllDifferent.hpp File Reference

```
#include "CbcBranchBase.hpp"
#include "OsiRowCut.hpp"
#include "CoinPackedMatrix.hpp"
#include "CbcBranchCut.hpp"
```

#### Classes

- class [CbcBranchAllDifferent](#)

*Define a branch class that branches so that it is only satisfied if all members have different values So cut is  $x \leq y-1$  or  $x \geq y+1$ .*

## 8.5 /home/ted/COIN/trunk/Cbc/src/CbcBranchBase.hpp File Reference

```
#include <string>
#include <vector>
#include "OsiBranchingObject.hpp"
#include "CbcObject.hpp"
#include "CbcBranchingObject.hpp"
#include "CbcBranchDecision.hpp"
#include "CbcConsequence.hpp"
#include "CbcObjectUpdateData.hpp"
```

### Enumerations

- enum [CbcRangeCompare](#) {  
[CbcRangeSame](#), [CbcRangeDisjoint](#), [CbcRangeSubset](#), [CbcRangeSuperset](#),  
[CbcRangeOverlap](#) }

### Functions

- static [CbcRangeCompare](#) [CbcCompareRanges](#) (double \*thisBd, const double \*otherBd, const bool replaceIfOverlap)  
*Compare two ranges.*

### 8.5.1 Enumeration Type Documentation

#### 8.5.1.1 enum CbcRangeCompare

##### Enumerator

***CbcRangeSame***  
***CbcRangeDisjoint***  
***CbcRangeSubset***  
***CbcRangeSuperset***  
***CbcRangeOverlap***

Definition at line 13 of file CbcBranchBase.hpp.

### 8.5.2 Function Documentation

**8.5.2.1 static CbcRangeCompare CbcCompareRanges ( double \* thisBd, const double \* otherBd, const bool replaceIfOverlap )** *[inline], [static]*

Compare two ranges.

The two bounds arrays are both of size two and describe closed intervals. Return the appropriate CbcRangeCompare value (first argument being the sub/superset if that's the case). In case of overlap (and if `replaceIfOverlap` is true) replace the content of `thisBd` with the intersection of the ranges.

Definition at line 36 of file CbcBranchBase.hpp.

## 8.6 /home/ted/COIN/trunk/Cbc/src/CbcBranchCut.hpp File Reference

```
#include "CbcBranchBase.hpp"
#include "OsiRowCut.hpp"
#include "CoinPackedMatrix.hpp"
```

### Classes

- class [CbcBranchCut](#)  
*Define a cut branching class.*
- class [CbcCutBranchingObject](#)  
*Cut branching object.*

## 8.7 /home/ted/COIN/trunk/Cbc/src/CbcBranchDecision.hpp File Reference

```
#include "CbcBranchBase.hpp"
```

### Classes

- class [CbcBranchDecision](#)

## 8.8 /home/ted/COIN/trunk/Cbc/src/CbcBranchDefaultDecision.hpp File Reference

```
#include "CbcBranchBase.hpp"
```

### Classes

- class [CbcBranchDefaultDecision](#)  
*Branching decision default class.*

## 8.9 /home/ted/COIN/trunk/Cbc/src/CbcBranchDynamic.hpp File Reference

```
#include "CoinPackedMatrix.hpp"
#include "CbcSimpleIntegerDynamicPseudoCost.hpp"
#include "CbcBranchActual.hpp"
```

### Classes

- class [CbcBranchDynamicDecision](#)  
*Branching decision dynamic class.*
- class [CbcDynamicPseudoCostBranchingObject](#)  
*Simple branching object for an integer variable with pseudo costs.*

## 8.10 /home/ted/COIN/trunk/Cbc/src/CbcBranchingObject.hpp File Reference

```
#include <string>
#include <vector>
#include "CbcBranchBase.hpp"
#include "OsiBranchingObject.hpp"
```

### Classes

- class [CbcBranchingObject](#)  
*Abstract branching object base class Now just difference with OsiBranchingObject.*

### Enumerations

- enum [CbcBranchObjType](#) {  
[SimpleIntegerBranchObj](#) = 100, [SimpleIntegerDynamicPseudoCostBranchObj](#) = 101, [CliqueBranchObj](#) = 102,  
[LongCliqueBranchObj](#) = 103,  
[SoSBranchObj](#) = 104, [NWayBranchObj](#) = 105, [FollowOnBranchObj](#) = 106, [DummyBranchObj](#) = 107,  
[GeneralDepthBranchObj](#) = 108, [OneGeneralBranchingObj](#) = 110, [CutBranchingObj](#) = 200, [LotsizeBranchObj](#) =  
300,  
[DynamicPseudoCostBranchObj](#) = 400 }

#### 8.10.1 Enumeration Type Documentation

##### 8.10.1.1 enum CbcBranchObjType

#### Enumerator

***SimpleIntegerBranchObj***  
***SimpleIntegerDynamicPseudoCostBranchObj***  
***CliqueBranchObj***  
***LongCliqueBranchObj***  
***SoSBranchObj***  
***NWayBranchObj***  
***FollowOnBranchObj***  
***DummyBranchObj***  
***GeneralDepthBranchObj***  
***OneGeneralBranchingObj***  
***CutBranchingObj***  
***LotsizeBranchObj***  
***DynamicPseudoCostBranchObj***

Definition at line 18 of file CbcBranchingObject.hpp.

## 8.11 /home/ted/COIN/trunk/Cbc/src/CbcBranchLotsize.hpp File Reference

```
#include "CbcBranchBase.hpp"
```

### Classes

- class [CbcLotsize](#)  
*Lotsize class.*
- class [CbcLotsizeBranchingObject](#)  
*Lotsize branching object.*

## 8.12 /home/ted/COIN/trunk/Cbc/src/CbcBranchToFixLots.hpp File Reference

```
#include "CbcBranchCut.hpp"  
#include "CbcBranchBase.hpp"  
#include "OsiRowCut.hpp"  
#include "CoinPackedMatrix.hpp"
```

### Classes

- class [CbcBranchToFixLots](#)  
*Define a branch class that branches so that one way variables are fixed while the other way cuts off that solution.*

## 8.13 /home/ted/COIN/trunk/Cbc/src/CbcClique.hpp File Reference

### Classes

- class [CbcClique](#)  
*Branching object for cliques.*
- class [CbcCliqueBranchingObject](#)  
*Branching object for unordered cliques.*
- class [CbcLongCliqueBranchingObject](#)  
*Unordered Clique Branching Object class.*

## 8.14 /home/ted/COIN/trunk/Cbc/src/CbcCompare.hpp File Reference

### Classes

- class [CbcCompare](#)

## 8.15 /home/ted/COIN/trunk/Cbc/src/CbcCompareActual.hpp File Reference

```
#include "CbcNode.hpp"  
#include "CbcCompareBase.hpp"  
#include "CbcCompare.hpp"  
#include "CbcCompareDepth.hpp"  
#include "CbcCompareDefault.hpp"
```

### 8.16 /home/ted/COIN/trunk/Cbc/src/CbcCompareBase.hpp File Reference

```
#include "CbcNode.hpp"  
#include "CbcConfig.h"
```

#### Classes

- class [CbcCompareBase](#)

### 8.17 /home/ted/COIN/trunk/Cbc/src/CbcCompareDefault.hpp File Reference

```
#include "CbcNode.hpp"  
#include "CbcCompareBase.hpp"  
#include "CbcCompare.hpp"
```

#### Classes

- class [CbcCompareDefault](#)

### 8.18 /home/ted/COIN/trunk/Cbc/src/CbcCompareDepth.hpp File Reference

```
#include "CbcNode.hpp"  
#include "CbcCompareBase.hpp"  
#include "CbcCompare.hpp"
```

#### Classes

- class [CbcCompareDepth](#)

### 8.19 /home/ted/COIN/trunk/Cbc/src/CbcCompareEstimate.hpp File Reference

```
#include "CbcNode.hpp"  
#include "CbcCompareBase.hpp"  
#include "CbcCompare.hpp"
```

#### Classes

- class [CbcCompareEstimate](#)

### 8.20 /home/ted/COIN/trunk/Cbc/src/CbcCompareObjective.hpp File Reference

```
#include "CbcNode.hpp"  
#include "CbcCompareBase.hpp"  
#include "CbcCompare.hpp"
```

## Classes

- class [CbcCompareObjective](#)

## 8.21 /home/ted/COIN/trunk/Cbc/src/CbcConfig.h File Reference

```
#include "config_cbc_default.h"
```

## 8.22 /home/ted/COIN/trunk/Cbc/src/CbcConsequence.hpp File Reference

### Classes

- class [CbcConsequence](#)  
*Abstract base class for consequent bounds.*

## 8.23 /home/ted/COIN/trunk/Cbc/src/CbcCountRowCut.hpp File Reference

### Classes

- class [CbcCountRowCut](#)  
*OsiRowCut augmented with bookkeeping.*
- struct [CoinHashLink](#)  
*Really for Conflict cuts to - a) stop duplicates b) allow half baked cuts The whichRow\_ field in OsiRowCut2 is used for a type 0 - normal 1 - processed cut 2 - unprocessed cut i.e.*
- class [CbcRowCuts](#)

## 8.24 /home/ted/COIN/trunk/Cbc/src/CbcCutGenerator.hpp File Reference

```
#include "OsiSolverInterface.hpp"  
#include "OsiCuts.hpp"  
#include "CglCutGenerator.hpp"  
#include "CbcCutModifier.hpp"
```

### Classes

- class [CbcCutGenerator](#)  
*Interface between Cbc and Cut Generation Library.*

### Macros

- #define [SCANCUTS](#) 1000
- #define [SCANCUTS\\_PROBING](#) 1000

### 8.24.1 Macro Definition Documentation

#### 8.24.1.1 `#define SCANCUTS 1000`

Definition at line 464 of file CbcCutGenerator.hpp.

#### 8.24.1.2 `#define SCANCUTS_PROBING 1000`

Definition at line 466 of file CbcCutGenerator.hpp.

## 8.25 `/home/ted/COIN/trunk/Cbc/src/CbcCutModifier.hpp` File Reference

```
#include "OsiSolverInterface.hpp"
#include "OsiCuts.hpp"
#include "CglCutGenerator.hpp"
```

### Classes

- class [CbcCutModifier](#)

*Abstract cut modifier base class.*

## 8.26 `/home/ted/COIN/trunk/Cbc/src/CbcCutSubsetModifier.hpp` File Reference

```
#include "OsiSolverInterface.hpp"
#include "OsiCuts.hpp"
#include "CglCutGenerator.hpp"
#include "CbcCutModifier.hpp"
```

### Classes

- class [CbcCutSubsetModifier](#)

*Simple cut modifier base class.*

## 8.27 `/home/ted/COIN/trunk/Cbc/src/CbcDummyBranchingObject.hpp` File Reference

```
#include "CbcBranchBase.hpp"
```

### Classes

- class [CbcDummyBranchingObject](#)

*Dummy branching object.*



## 8.28 /home/ted/COIN/trunk/Cbc/src/CbcEventHandler.hpp File Reference

Event handling for cbc.

```
#include <map>
```

### Classes

- class [CbcEventHandler](#)  
*Base class for Cbc event handling.*

#### 8.28.1 Detailed Description

Event handling for cbc. This file contains the declaration of [CbcEventHandler](#), used for event handling in cbc.

The central method is [CbcEventHandler::event\(\)](#). The default semantics of this call are 'ask for the action to take in response to this event'. The call is made at the point in the code where the event occurs (*e.g.*, when a solution is found, or when a node is added to or removed from the search tree). The return value specifies the action to perform in response to the event (*e.g.*, continue, or stop).

This is a lazy class. Initially, it knows nothing about specific events, and returns `dfItAction_` for any event. This makes for a trivial constructor and fast startup. The only place where the list of known events or actions is hardwired is in the enum definitions for `CbcEvent` and `CbcAction`, respectively.

At the first call to `setAction`, a map is created to hold (Event,Action) pairs, and this map will be consulted ever after. Events not in the map will still return the default value.

For serious extensions, derive a subclass and replace `event()` with a function that suits you better. The function has access to the [CbcModel](#) via a pointer held in the [CbcEventHandler](#) object, and can do as much thinking as it likes before returning an answer. You can also print as much information as you want. The model is held as a `const`, however, so you can't alter reality.

The design of the class deliberately matches `ClpEventHandler`, so that other solvers can participate in cbc without breaking the patterns set by clp-specific code.

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

## 8.29 /home/ted/COIN/trunk/Cbc/src/CbcFathom.hpp File Reference

```
#include "CbcConfig.h"  
#include "OsiClpSolverInterface.hpp"
```

### Classes

- class [CbcFathom](#)  
*Fathom base class.*
- class [CbcOsiSolver](#)  
*This is for codes where solver needs to know about [CbcModel](#) Seems to provide only one value-added feature, a [CbcModel](#) object.*

### 8.30 /home/ted/COIN/trunk/Cbc/src/CbcFathomDynamicProgramming.hpp File Reference

```
#include "CbcFathom.hpp"
```

#### Classes

- class [CbcFathomDynamicProgramming](#)  
*FathomDynamicProgramming class.*

### 8.31 /home/ted/COIN/trunk/Cbc/src/CbcFeasibilityBase.hpp File Reference

#### Classes

- class [CbcFeasibilityBase](#)

### 8.32 /home/ted/COIN/trunk/Cbc/src/CbcFixVariable.hpp File Reference

```
#include "CbcBranchBase.hpp"
```

#### Classes

- class [CbcFixVariable](#)  
*Class for consequent bounds.*

### 8.33 /home/ted/COIN/trunk/Cbc/src/CbcFollowOn.hpp File Reference

```
#include "CbcBranchBase.hpp"  
#include "OsiRowCut.hpp"  
#include "CoinHelperFunctions.hpp"  
#include "CoinPackedMatrix.hpp"
```

#### Classes

- class [CbcFollowOn](#)  
*Define a follow on class.*
- class [CbcFixingBranchingObject](#)  
*General Branching Object class.*
- class [CbcIdiotBranch](#)  
*Define an idiotic idea class.*

## 8.34 /home/ted/COIN/trunk/Cbc/src/CbcFullNodeInfo.hpp File Reference

```
#include <string>
#include <vector>
#include "CoinWarmStartBasis.hpp"
#include "CoinSearchTree.hpp"
#include "CbcBranchBase.hpp"
#include "CbcNodeInfo.hpp"
```

### Classes

- class [CbcFullNodeInfo](#)

*Information required to recreate the subproblem at this node.*

## 8.35 /home/ted/COIN/trunk/Cbc/src/CbcGenCbcParam.hpp File Reference

### Classes

- class [CbcCbcParam](#)

*Class for control parameters that act on a [CbcModel](#) object.*

### Namespaces

- [CbcCbcParamUtils](#)

### Functions

- void [CbcCbcParamUtils::addCbcCbcParams](#) (int &numParams, CoinParamVec &paramVec, [CbcModel](#) \*model)
- void [CbcCbcParamUtils::loadCbcParamObj](#) (const CoinParamVec paramVec, int first, int last, [CbcModel](#) \*model)
- void [CbcCbcParamUtils::setCbcModelDefaults](#) ([CbcModel](#) \*model)
- int [CbcCbcParamUtils::pushCbcCbcDbI](#) (CoinParam \*param)
- int [CbcCbcParamUtils::pushCbcCbcInt](#) (CoinParam \*param)

## 8.36 /home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp File Reference

```
#include "CoinParam.hpp"
```

```

#include "CoinMessageHandler.hpp"
#include "CglCutGenerator.hpp"
#include "CglProbing.hpp"
#include "CglClique.hpp"
#include "CglFlowCover.hpp"
#include "CglGomory.hpp"
#include "CglKnapsackCover.hpp"
#include "CglMixedIntegerRounding2.hpp"
#include "CglOddHole.hpp"
#include "CglRedSplit.hpp"
#include "CglTwomir.hpp"
#include "CbcModel.hpp"
#include "CbcHeuristic.hpp"
#include "CbcHeuristicFPump.hpp"
#include "CbcHeuristicGreedy.hpp"
#include "CbcHeuristicLocal.hpp"
#include "CbcTreeLocal.hpp"
#include "CbcGenMessages.hpp"

```

## Classes

- class [CbcGenCtlBlk](#)
- struct [CbcGenCtlBlk::genParamsInfo\\_struct](#)  
*Start and end of cbc-generic parameters in parameter vector.*
- struct [CbcGenCtlBlk::cbcParamsInfo\\_struct](#)  
*Start and end of [CbcModel](#) parameters in parameter vector.*
- struct [CbcGenCtlBlk::osiParamsInfo\\_struct](#)  
*Start and end of [OsiSolverInterface](#) parameters in parameter vector.*
- struct [CbcGenCtlBlk::debugSolInfo\\_struct](#)  
*Array of primal variable values for debugging.*
- struct [CbcGenCtlBlk::babState\\_struct](#)  
*State of branch-and-cut.*
- struct [CbcGenCtlBlk::djFixCtl\\_struct](#)  
*Control use of reduced cost fixing prior to B&C.*
- struct [CbcGenCtlBlk::chooseStrongCtl\\_struct](#)  
*Control variables for a strong branching method.*

## Namespaces

- [CbcGenParamUtils](#)

## Macros

- `#define CBC\_GENERIC\_VERSION "00.01.00"`

## Functions

- void [CbcGenParamUtils::addCbcGenParams](#) (int &numParams, CoinParamVec &paramVec, [CbcGenCtlBlk](#) \*ctlBlk)

### 8.36.1 Macro Definition Documentation

#### 8.36.1.1 #define CBC\_GENERIC\_VERSION "00.01.00"

Definition at line 53 of file CbcGenCtlBlk.hpp.

## 8.37 /home/ted/COIN/trunk/Cbc/src/CbcGeneral.hpp File Reference

```
#include "CbcBranchBase.hpp"
```

### Classes

- class [CbcGeneral](#)

*Define a catch all class.*

## 8.38 /home/ted/COIN/trunk/Cbc/src/CbcGeneralDepth.hpp File Reference

```
#include "CbcGeneral.hpp"  
#include "CbcBranchBase.hpp"  
#include "CbcSubProblem.hpp"
```

## 8.39 /home/ted/COIN/trunk/Cbc/src/CbcGenMessages.hpp File Reference

This file contains the enum that defines symbolic names for for cbc-generic messages.

### Enumerations

- enum [CbcGenMsgCode](#) { [CBCGEN\\_TEST\\_MSG](#) = 1, [CBCGEN\\_NEW\\_SOLVER](#), [CBCGEN\\_CONFUSION](#), [CBCGEN\\_DUMMY\\_END](#) }

*Symbolic names for cbc-generic messages.*

### 8.39.1 Detailed Description

This file contains the enum that defines symbolic names for for cbc-generic messages.

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

### 8.39.2 Enumeration Type Documentation

#### 8.39.2.1 enum CbcGenMsgCode

Symbolic names for cbc-generic messages.

These are the 'internal IDs' for cbc-generic messages.

### Enumerator

***CBCGEN\_TEST\_MSG***

**CBCGEN\_NEW\_SOLVER**

**CBCGEN\_CONFUSION**

**CBCGEN\_DUMMY\_END**

Definition at line 36 of file CbcGenMessages.hpp.

## 8.40 /home/ted/COIN/trunk/Cbc/src/CbcGenOsiParam.hpp File Reference

### Classes

- class [CbcOsiParam](#)  
*Class for control parameters that act on a OsiSolverInterface object.*

### Namespaces

- [CbcOsiParamUtils](#)

### Functions

- void [CbcOsiParamUtils::addCbcOsiParams](#) (int &numParams, CoinParamVec &paramVec, OsiSolverInterface \*osi)
- void [CbcOsiParamUtils::loadOsiParamObj](#) (const CoinParamVec paramVec, [CbcGenCtlBlk](#) \*ctlBlk)
- void [CbcOsiParamUtils::setOsiSolverInterfaceDefaults](#) (OsiSolverInterface \*osi)
- int [CbcOsiParamUtils::pushCbcOsiLogLevel](#) (CoinParam \*param)
- int [CbcOsiParamUtils::pushCbcOsiInt](#) (CoinParam \*param)
- int [CbcOsiParamUtils::pushCbcOsiDbf](#) (CoinParam \*param)
- int [CbcOsiParamUtils::pushCbcOsiKwd](#) (CoinParam \*param)
- int [CbcOsiParamUtils::pushCbcOsiHint](#) (CoinParam \*param)

## 8.41 /home/ted/COIN/trunk/Cbc/src/CbcGenParam.hpp File Reference

### Classes

- class [CbcGenParam](#)  
*Class for cbc-generic control parameters.*

### Namespaces

- [CbcGenParamUtils](#)

### Functions

- void [CbcGenParamUtils::addCbcGenParams](#) (int &numParams, CoinParamVec &paramVec, [CbcGenCtlBlk](#) \*ctlBlk)
- void [CbcGenParamUtils::loadGenParamObj](#) (const CoinParamVec paramVec, int first, int last, [CbcGenCtlBlk](#) \*ctlBlk)
- void [CbcGenParamUtils::saveSolution](#) (const OsiSolverInterface \*osi, std::string fileName)

- bool [CbcGenParamUtils::readSolution](#) (std::string fileName, int &numRows, int &numCols, double &objVal, double \*\*rowActivity, double \*\*dualVars, double \*\*primalVars, double \*\*reducedCosts)
- int [CbcGenParamUtils::doBaCParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doDebugParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doExitParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doHelpParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doImportParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doPrintMaskParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doNothingParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doSolutionParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doUnimplementedParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::doVersionParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::pushCbcGenDblParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::pushCbcGenIntParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::pushCbcGenKwdParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::pushCbcGenStrParam](#) (CoinParam \*param)
- int [CbcGenParamUtils::pushCbcGenCutParam](#) (CoinParam \*param)

## 8.42 /home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp File Reference

```
#include <string>
#include <vector>
#include "CoinPackedMatrix.hpp"
#include "OsiCuts.hpp"
#include "CoinHelperFunctions.hpp"
#include "OsiBranchingObject.hpp"
```

### Classes

- class [CbcHeuristicNode](#)  
*A class describing the branching decisions that were made to get to the node where a heuristic was invoked from.*
- class [CbcHeuristicNodeList](#)
- class [CbcHeuristic](#)  
*Heuristic base class.*
- class [CbcRounding](#)  
*Rounding class.*
- class [CbcHeuristicPartial](#)  
*Partial solution class If user knows a partial solution this tries to get an integer solution it uses hotstart information.*
- class [CbcSerendipity](#)  
*heuristic - just picks up any good solution found by solver - see OsiBabSolver*
- class [CbcHeuristicJustOne](#)  
*Just One class - this chooses one at random.*

## 8.43 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDINS.hpp File Reference

```
#include "CbcHeuristic.hpp"
```

**Classes**

- class [CbcHeuristicDINS](#)

**8.44 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDive.hpp File Reference**

```
#include "CbcHeuristic.hpp"
```

**Classes**

- struct [PseudoReducedCost](#)
- class [CbcHeuristicDive](#)

*Dive class.*

**8.45 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveCoefficient.hpp File Reference**

```
#include "CbcHeuristicDive.hpp"
```

**Classes**

- class [CbcHeuristicDiveCoefficient](#)

*DiveCoefficient class.*

**8.46 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveFractional.hpp File Reference**

```
#include "CbcHeuristicDive.hpp"
```

**Classes**

- class [CbcHeuristicDiveFractional](#)

*DiveFractional class.*

**8.47 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveGuided.hpp File Reference**

```
#include "CbcHeuristicDive.hpp"
```

**Classes**

- class [CbcHeuristicDiveGuided](#)

*DiveGuided class.*



## 8.48 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveLineSearch.hpp File Reference

```
#include "CbcHeuristicDive.hpp"
```

### Classes

- class [CbcHeuristicDiveLineSearch](#)  
*DiveLineSearch class.*

## 8.49 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDivePseudoCost.hpp File Reference

```
#include "CbcHeuristicDive.hpp"
```

### Classes

- class [CbcHeuristicDivePseudoCost](#)  
*DivePseudoCost class.*

## 8.50 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveVectorLength.hpp File Reference

```
#include "CbcHeuristicDive.hpp"
```

### Classes

- class [CbcHeuristicDiveVectorLength](#)  
*DiveVectorLength class.*

## 8.51 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDW.hpp File Reference

```
#include "CbcHeuristic.hpp"
```

### Classes

- class [CbcHeuristicDW](#)  
*This is unlike the other heuristics in that it is very very compute intensive.*

## 8.52 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicFPump.hpp File Reference

```
#include "CbcHeuristic.hpp"  
#include "OsiClpSolverInterface.hpp"
```

### Classes

- class [CbcHeuristicFPump](#)  
*Feasibility Pump class.*

## 8.53 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicGreedy.hpp File Reference

```
#include "CbcHeuristic.hpp"
```

### Classes

- class [CbcHeuristicGreedyCover](#)  
*Greedy heuristic classes.*
- class [CbcHeuristicGreedyEquality](#)
- class [CbcHeuristicGreedySOS](#)  
*Greedy heuristic for SOS and L rows (and positive elements)*

## 8.54 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicLocal.hpp File Reference

```
#include "CbcHeuristic.hpp"
```

### Classes

- class [CbcHeuristicLocal](#)  
*LocalSearch class.*
- class [CbcHeuristicProximity](#)
- class [CbcHeuristicNaive](#)  
*Naive class a) Fix all ints as close to zero as possible b) Fix all ints with nonzero costs and < large to zero c) Put bounds round continuous and UIs and maximize.*
- class [CbcHeuristicCrossover](#)  
*Crossover Search class.*

## 8.55 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicPivotAndFix.hpp File Reference

```
#include "CbcHeuristic.hpp"
```

### Classes

- class [CbcHeuristicPivotAndFix](#)  
*LocalSearch class.*

## 8.56 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicRandRound.hpp File Reference

```
#include "CbcHeuristic.hpp"
```

## Classes

- class [CbcHeuristicRandRound](#)  
*LocalSearch class.*

## 8.57 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicRENS.hpp File Reference

```
#include "CbcHeuristic.hpp"
```

## Classes

- class [CbcHeuristicRENS](#)  
*LocalSearch class.*

## 8.58 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicRINS.hpp File Reference

```
#include "CbcHeuristic.hpp"  
#include "CbcHeuristicRENS.hpp"  
#include "CbcHeuristicDINS.hpp"  
#include "CbcHeuristicVND.hpp"
```

## Classes

- class [CbcHeuristicRINS](#)  
*LocalSearch class.*

## 8.59 /home/ted/COIN/trunk/Cbc/src/CbcHeuristicVND.hpp File Reference

```
#include "CbcHeuristic.hpp"
```

## Classes

- class [CbcHeuristicVND](#)  
*LocalSearch class.*

## 8.60 /home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp File Reference

```
#include "CoinModel.hpp"  
#include "OsiClpSolverInterface.hpp"  
#include "OsiChooseVariable.hpp"  
#include "CbcFathom.hpp"  
#include "CbcHeuristic.hpp"  
#include "OsiBranchingObject.hpp"  
#include <string>  
#include "CglStored.hpp"
```

## Classes

- class [OsiSolverLink](#)  
*This is to allow the user to replace initialSolve and resolve This version changes coefficients.*
- class [OsiLinkedBound](#)  
*List of bounds which depend on other bounds.*
- class [CbcHeuristicDynamic3](#)  
*heuristic - just picks up any good solution*
- class [OsiOldLink](#)
- class [OsiOldLinkBranchingObject](#)  
*Branching object for Linked ordered sets.*
- class [OsiOneLink](#)  
*Define data for one link.*
- class [OsiLink](#)  
*Define Special Linked Ordered Sets.*
- class [OsiLinkBranchingObject](#)  
*Branching object for Linked ordered sets.*
- class [OsiBiLinear](#)  
*Define BiLinear objects.*
- class [OsiBiLinearBranchingObject](#)  
*Branching object for BiLinear objects.*
- class [OsiBiLinearEquality](#)  
*Define Continuous BiLinear objects for an == bound.*
- class [OsiSimpleFixedInteger](#)  
*Define a single integer class - but one where you keep branching until fixed even if satisfied.*
- class [OsiUsesBiLinear](#)  
*Define a single variable class which is involved with [OsiBiLinear](#) objects.*
- class [OsiChooseStrongSubset](#)  
*This class chooses a variable to branch on.*
- class [CglTemporary](#)  
*Stored Temporary Cut Generator Class - destroyed after first use.*
- class [OsiSolverLinearizedQuadratic](#)  
*This is to allow the user to replace initialSolve and resolve.*

## Functions

- `ClpSimplex * approximateSolution (CoinModel &coinModel, int numberPasses, double deltaTolerance, int mode=0)`  
*Return an approximate solution to a CoinModel.*

### 8.60.1 Function Documentation

#### 8.60.1.1 `ClpSimplex* approximateSolution ( CoinModel & coinModel, int numberPasses, double deltaTolerance, int mode = 0 )`

Return an approximate solution to a CoinModel.

Lots of bounds may be odd to force a solution. mode = 0 just tries to get a continuous solution

## 8.61 /home/ted/COIN/trunk/Cbc/src/CbcMessage.hpp File Reference

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

## Classes

- class [CbcMessage](#)

## Enumerations

- enum [CBC\\_Message](#) {  
[CBC\\_END\\_GOOD](#), [CBC\\_MAXNODES](#), [CBC\\_MAXTIME](#), [CBC\\_MAXSOLS](#),  
[CBC\\_EVENT](#), [CBC\\_MAXITERS](#), [CBC\\_SOLUTION](#), [CBC\\_END\\_SOLUTION](#),  
[CBC\\_SOLUTION2](#), [CBC\\_END](#), [CBC\\_INFEAS](#), [CBC\\_STRONG](#),  
[CBC\\_SOLINDIVIDUAL](#), [CBC\\_INTEGERINCREMENT](#), [CBC\\_STATUS](#), [CBC\\_GAP](#),  
[CBC\\_ROUNDING](#), [CBC\\_TREE\\_SOL](#), [CBC\\_ROOT](#), [CBC\\_GENERATOR](#),  
[CBC\\_BRANCH](#), [CBC\\_STRONGSOL](#), [CBC\\_NOINT](#), [CBC\\_VUB\\_PASS](#),  
[CBC\\_VUB\\_END](#), [CBC\\_NOTFEAS1](#), [CBC\\_NOTFEAS2](#), [CBC\\_NOTFEAS3](#),  
[CBC\\_CUTOFF\\_WARNING1](#), [CBC\\_ITERATE\\_STRONG](#), [CBC\\_PRIORITY](#), [CBC\\_WARNING\\_STRONG](#),  
[CBC\\_START\\_SUB](#), [CBC\\_END\\_SUB](#), [CBC\\_THREAD\\_STATS](#), [CBC\\_CUTS\\_STATS](#),  
[CBC\\_STRONG\\_STATS](#), [CBC\\_UNBOUNDED](#), [CBC\\_OTHER\\_STATS](#), [CBC\\_HEURISTICS\\_OFF](#),  
[CBC\\_STATUS2](#), [CBC\\_FPUMP1](#), [CBC\\_FPUMP2](#), [CBC\\_STATUS3](#),  
[CBC\\_OTHER\\_STATS2](#), [CBC\\_RELAXED1](#), [CBC\\_RELAXED2](#), [CBC\\_RESTART](#),  
[CBC\\_GENERAL](#), [CBC\\_ROOT\\_DETAIL](#), [CBC\\_FATHOM\\_CHANGE](#), [CBC\\_DUMMY\\_END](#) }

*This deals with Cbc messages (as against Clp messages etc).*

## 8.61.1 Enumeration Type Documentation

8.61.1.1 enum [CBC\\_Message](#)

This deals with Cbc messages (as against Clp messages etc).

CoinMessageHandler.hpp is the general part of message handling. All it has are enum's for the various messages. CbcMessage.cpp has text in various languages.

It is trivial to use the .hpp and .cpp file as a basis for messages for other components.

## Enumerator

***CBC\_END\_GOOD***  
***CBC\_MAXNODES***  
***CBC\_MAXTIME***  
***CBC\_MAXSOLS***  
***CBC\_EVENT***  
***CBC\_MAXITERS***  
***CBC\_SOLUTION***  
***CBC\_END\_SOLUTION***  
***CBC\_SOLUTION2***  
***CBC\_END***  
***CBC\_INFEAS***

***CBC\_STRONG***  
***CBC\_SOLINDIVIDUAL***  
***CBC\_INTEGERINCREMENT***  
***CBC\_STATUS***  
***CBC\_GAP***  
***CBC\_ROUNDING***  
***CBC\_TREE\_SOL***  
***CBC\_ROOT***  
***CBC\_GENERATOR***  
***CBC\_BRANCH***  
***CBC\_STRONGSOL***  
***CBC\_NOINT***  
***CBC\_VUB\_PASS***  
***CBC\_VUB\_END***  
***CBC\_NOTFEAS1***  
***CBC\_NOTFEAS2***  
***CBC\_NOTFEAS3***  
***CBC\_CUTOFF\_WARNING1***  
***CBC\_ITERATE\_STRONG***  
***CBC\_PRIORITY***  
***CBC\_WARNING\_STRONG***  
***CBC\_START\_SUB***  
***CBC\_END\_SUB***  
***CBC\_THREAD\_STATS***  
***CBC\_CUTS\_STATS***  
***CBC\_STRONG\_STATS***  
***CBC\_UNBOUNDED***  
***CBC\_OTHER\_STATS***  
***CBC\_HEURISTICS\_OFF***  
***CBC\_STATUS2***  
***CBC\_FPUMP1***  
***CBC\_FPUMP2***  
***CBC\_STATUS3***  
***CBC\_OTHER\_STATS2***  
***CBC\_RELAXED1***  
***CBC\_RELAXED2***  
***CBC\_RESTART***  
***CBC\_GENERAL***  
***CBC\_ROOT\_DETAIL***  
***CBC\_FATHOM\_CHANGE***  
***CBC\_DUMMY\_END***

Definition at line 24 of file CbcMessage.hpp.

## 8.62 /home/ted/COIN/trunk/Cbc/src/CbcMipStartIO.hpp File Reference

```
#include <vector>
#include <string>
#include <utility>
```

### Functions

- int [readMIPStart](#) ([CbcModel](#) \*model, const char \*fileName, std::vector< std::pair< std::string, double > > &colValues, double &solObj)
- int [computeCompleteSolution](#) ([CbcModel](#) \*model, const std::vector< std::string > colNames, const std::vector< std::pair< std::string, double > > &colValues, double \*sol, double &obj)

### 8.62.1 Function Documentation

**8.62.1.1** int [readMIPStart](#) ( [CbcModel](#) \* model, const char \* fileName, std::vector< std::pair< std::string, double > > & colValues, double & solObj )

**8.62.1.2** int [computeCompleteSolution](#) ( [CbcModel](#) \* model, const std::vector< std::string > colNames, const std::vector< std::pair< std::string, double > > & colValues, double \* sol, double & obj )

## 8.63 /home/ted/COIN/trunk/Cbc/src/CbcModel.hpp File Reference

```
#include <string>
#include <vector>
#include "CoinMessageHandler.hpp"
#include "OsiSolverInterface.hpp"
#include "OsiBranchingObject.hpp"
#include "OsiCuts.hpp"
#include "CoinWarmStartBasis.hpp"
#include "CbcCompareBase.hpp"
#include "CbcCountRowCut.hpp"
#include "CbcMessage.hpp"
#include "CbcEventHandler.hpp"
#include "ClpDualRowPivot.hpp"
```

### Classes

- class [CbcModel](#)  
*Simple Branch and bound class.*

### Functions

- void [getIntegerInformation](#) (const [OsiObject](#) \*object, double &originalLower, double &originalUpper)  
*So we can use osiObject or CbcObject during transition.*
- int [CbcMain](#) (int argc, const char \*argv[], [OsiClpSolverInterface](#) &solver, [CbcModel](#) \*\*babSolver)
- int [CbcMain](#) (int argc, const char \*argv[], [CbcModel](#) &babSolver)
- int [callCbc](#) (const char \*input2, [OsiClpSolverInterface](#) &solver1)
- int [callCbc](#) (const char \*input2)

- int `callCbc` (const std::string input2, OsiClpSolverInterface &solver1)
- int `callCbc` (const std::string input2)
- void `CbcMain0` (CbcModel &babSolver)
- int `CbcMain1` (int argc, const char \*argv[], CbcModel &babSolver)
- int `callCbc` (const char \*input2, CbcModel &babSolver)
- int `callCbc` (const std::string input2, CbcModel &babSolver)
- int `callCbc1` (const char \*input2, CbcModel &babSolver)
- int `callCbc1` (const std::string input2, CbcModel &babSolver)
- int `callCbc1` (const char \*input2, CbcModel &babSolver, int(CbcModel \*currentSolver, int whereFrom))
- int `callCbc1` (const std::string input2, CbcModel &babSolver, int(CbcModel \*currentSolver, int whereFrom))
- int `CbcMain1` (int argc, const char \*argv[], CbcModel &babSolver, int(CbcModel \*currentSolver, int whereFrom))
- void `setCutAndHeuristicOptions` (CbcModel &model)

### 8.63.1 Function Documentation

8.63.1.1 void `getIntegerInformation` ( const OsiObject \* *object*, double & *originalLower*, double & *originalUpper* )

So we can use `osiObject` or `CbcObject` during transition.

8.63.1.2 int `CbcMain` ( int *argc*, const char \* *argv*[], OsiClpSolverInterface & *solver*, CbcModel \*\* *babSolver* )

8.63.1.3 int `CbcMain` ( int *argc*, const char \* *argv*[], CbcModel & *babSolver* )

8.63.1.4 int `callCbc` ( const char \* *input2*, OsiClpSolverInterface & *solver1* )

8.63.1.5 int `callCbc` ( const char \* *input2* )

8.63.1.6 int `callCbc` ( const std::string *input2*, OsiClpSolverInterface & *solver1* )

8.63.1.7 int `callCbc` ( const std::string *input2* )

8.63.1.8 void `CbcMain0` ( CbcModel & *babSolver* )

8.63.1.9 int `CbcMain1` ( int *argc*, const char \* *argv*[], CbcModel & *babSolver* )

8.63.1.10 int `callCbc` ( const char \* *input2*, CbcModel & *babSolver* )

8.63.1.11 int `callCbc` ( const std::string *input2*, CbcModel & *babSolver* )

8.63.1.12 int `callCbc1` ( const char \* *input2*, CbcModel & *babSolver* )

8.63.1.13 int `callCbc1` ( const std::string *input2*, CbcModel & *babSolver* )

8.63.1.14 int `callCbc1` ( const char \* *input2*, CbcModel & *babSolver*, int(CbcModel \*currentSolver, int whereFrom) )

8.63.1.15 int `callCbc1` ( const std::string *input2*, CbcModel & *babSolver*, int(CbcModel \*currentSolver, int whereFrom) )

8.63.1.16 int `CbcMain1` ( int *argc*, const char \* *argv*[], CbcModel & *babSolver*, int(CbcModel \*currentSolver, int whereFrom) )

8.63.1.17 void `setCutAndHeuristicOptions` ( CbcModel & *model* )



## 8.64 /home/ted/COIN/trunk/Cbc/src/CbcNode.hpp File Reference

```
#include <string>
#include <vector>
#include "CoinWarmStartBasis.hpp"
#include "CoinSearchTree.hpp"
#include "CbcBranchBase.hpp"
#include "CbcNodeInfo.hpp"
#include "CbcFullNodeInfo.hpp"
#include "CbcPartialNodeInfo.hpp"
```

### Classes

- class [CbcNode](#)

*Information required while the node is live.*

## 8.65 /home/ted/COIN/trunk/Cbc/src/CbcNodeInfo.hpp File Reference

```
#include <string>
#include <vector>
#include "CoinWarmStartBasis.hpp"
#include "CoinSearchTree.hpp"
#include "CbcBranchBase.hpp"
```

### Classes

- class [CbcNodeInfo](#)

*Information required to recreate the subproblem at this node.*

## 8.66 /home/ted/COIN/trunk/Cbc/src/CbcNWay.hpp File Reference

### Classes

- class [CbcNWay](#)  
*Define an n-way class for variables.*
- class [CbcNWayBranchingObject](#)  
*N way branching Object class.*

## 8.67 /home/ted/COIN/trunk/Cbc/src/CbcObject.hpp File Reference

```
#include <string>
#include <vector>
#include "OsiBranchingObject.hpp"
```



```

M_STR_MESSAGES,
CLP_PARAM_STR_AUTOSCALE, CLP_PARAM_STR_CHOLESKY, CLP_PARAM_STR_KKT, CLP_PARAM_
STR_BARRIERSCALE,
CLP_PARAM_STR_GAMMA, CLP_PARAM_STR_CROSSOVER, CLP_PARAM_STR_PFI, CLP_PARAM_NO-
TUSED_ALGORITHM,
CBC_PARAM_STR_NODESTRATEGY = 251, CBC_PARAM_STR_BRANCHSTRATEGY, CBC_PARAM_NOT-
USED_ADDCUTSSTRATEGY, CBC_PARAM_STR_GOMORYCUTS,
CBC_PARAM_STR_PROBINGCUTS, CBC_PARAM_STR_KNAPSACKCUTS, CBC_PARAM_NOTUSED_OD-
DHOECUTS, CBC_PARAM_STR_ROUNDING,
CBC_PARAM_STR_SOLVER, CBC_PARAM_STR_CLIQUERCUTS, CBC_PARAM_STR_COSTSTRATEGY, C-
BC_PARAM_STR_FLOWCUTS,
CBC_PARAM_STR_MIXEDCUTS, CBC_PARAM_STR_TWOMIRCUTS, CBC_PARAM_STR_PREPROCESS,
CLP_PARAM_ACTION_DIRECTORY = 301,
CLP_PARAM_ACTION_IMPORT, CLP_PARAM_ACTION_EXPORT, CLP_PARAM_ACTION_RESTORE, CLP-
_PARAM_ACTION_SAVE,
CLP_PARAM_ACTION_DUALSIMPLEX, CLP_PARAM_ACTION_PRIMALSIMPLEX, CLP_PARAM_ACTION_
MAXIMIZE, CLP_PARAM_ACTION_MINIMIZE,
CLP_PARAM_ACTION_EXIT, CLP_PARAM_ACTION_STDIN, CLP_PARAM_ACTION_UNITTEST, CLP_PAR-
AM_ACTION_NETLIB_DUAL,
CLP_PARAM_ACTION_NETLIB_PRIMAL, CLP_PARAM_ACTION_SOLUTION, CLP_PARAM_ACTION_TIGH-
TEN, CLP_PARAM_ACTION_FAKEBOUND,
CLP_PARAM_ACTION_HELP, CLP_PARAM_ACTION_PLUSMINUS, CLP_PARAM_ACTION_NETWORK, CL-
P_PARAM_ACTION_ALLSLACK,
CLP_PARAM_ACTION_REVERSE, CLP_PARAM_ACTION_BARRIER, CLP_PARAM_ACTION_NETLIB_BAR-
RIER, CLP_PARAM_ACTION_REALLY_SCALE,
CLP_PARAM_ACTION_BASISIN, CLP_PARAM_ACTION_BASISOUT, CLP_PARAM_ACTION_SOLVECONTI-
NUOUS, CBC_PARAM_ACTION_BAB,
CBC_PARAM_ACTION_MIPLIB, CLP_PARAM_ACTION_CLEARCUTS, CLP_VERSION_NOTUSED_PRINTV-
ERSION, CBC_PARAM_NOTUSED_OSLSTUFF = 401,
CBC_PARAM_NOTUSED_CBCSTUFF, CBC_PARAM_NOTUSED_INVALID = 1000 }

```

*Parameter codes.*

### 8.69.1 Enumeration Type Documentation

#### 8.69.1.1 enum CbcParameterType

Parameter codes.

Parameter type ranges are allocated as follows

- 1 – 100 double parameters
- 101 – 200 integer parameters
- 201 – 250 string parameters
- 251 – 300 cuts etc(string but broken out for clarity)
- 301 – 400 ‘actions’

‘Actions’ do not necessarily invoke an immediate action; it’s just that they don’t fit neatly into the parameters array.

This coding scheme is in flux. CBC\_PARAM\_STR\_NODESTRATEGY, CBC\_PARAM\_STR\_BRANCHSTRATEGY, CBC\_PARAM\_NOTUSED\_ADDCUTSSTRATEGY, CLP\_PARAM\_ACTION\_CLEARCUTS, CBC\_PARAM\_NOTUSED\_OSLSTUFF, CBC\_PARAM\_NOTUSED\_CBCSTUFF are not used at present (03.10.24).

## Enumerator

***CBC\_PARAM\_GENERALQUERY***  
***CBC\_PARAM\_FULLGENERALQUERY***  
***CLP\_PARAM\_DBL\_PRIMALTOLERANCE***  
***CLP\_PARAM\_DBL\_DUALTOLERANCE***  
***CBC\_PARAM\_DBL\_CUTOFF***  
***CLP\_PARAM\_DBL\_TIMELIMIT***  
***CLP\_PARAM\_DBL\_DUALBOUND***  
***CLP\_PARAM\_DBL\_PRIMALWEIGHT***  
***CLP\_PARAM\_DBL\_OBJSCALE***  
***CLP\_PARAM\_DBL\_RHSSCALE***  
***CBC\_PARAM\_DBL\_INFEASIBILITYWEIGHT***  
***CBC\_PARAM\_DBL\_INTEGERTOLERANCE***  
***CBC\_PARAM\_DBL\_INCREMENT***  
***CBC\_PARAM\_DBL\_ALLOWABLEGAP***  
***CBC\_PARAM\_DBL\_DJFIX***  
***CBC\_PARAM\_DBL\_GAPRATIO***  
***CBC\_PARAM\_DBL\_TIGHTENFACTOR***  
***CLP\_PARAM\_INT\_LOGLEVEL***  
***CLP\_PARAM\_INT\_SOLVERLOGLEVEL***  
***CBC\_PARAM\_INT\_MAXNODES***  
***CBC\_PARAM\_INT\_STRONGBRANCHING***  
***CLP\_PARAM\_INT\_MAXFACTOR***  
***CLP\_PARAM\_INT\_PERTVALUE***  
***CLP\_PARAM\_INT\_MAXITERATION***  
***CLP\_PARAM\_INT\_PRESOLVEPASS***  
***CLP\_PARAM\_INT\_IDIOT***  
***CLP\_PARAM\_INT\_SPRINT***  
***CLP\_PARAM\_INT\_OUTPUTFORMAT***  
***CLP\_PARAM\_INT\_SLPVALUE***  
***CLP\_PARAM\_INT\_PRESOLVEOPTIONS***  
***CLP\_PARAM\_INT\_PRINTOPTIONS***  
***CLP\_PARAM\_INT\_SPECIALOPTIONS***  
***CLP\_PARAM\_STR\_DIRECTION***  
***CLP\_PARAM\_STR\_DUALPIVOT***  
***CLP\_PARAM\_STR\_SCALING***  
***CLP\_PARAM\_STR\_ERRORSALLOWED***  
***CLP\_PARAM\_STR\_KEEPNAMES***  
***CLP\_PARAM\_STR\_SPARSEFACTOR***  
***CLP\_PARAM\_STR\_PRIMALPIVOT***  
***CLP\_PARAM\_STR\_PRESOLVE***  
***CLP\_PARAM\_STR\_CRASH***

*CLP\_PARAM\_STR\_BIASLU*  
*CLP\_PARAM\_STR\_PERTURBATION*  
*CLP\_PARAM\_STR\_MESSAGES*  
*CLP\_PARAM\_STR\_AUTOSCALE*  
*CLP\_PARAM\_STR\_CHOLESKY*  
*CLP\_PARAM\_STR\_KKT*  
*CLP\_PARAM\_STR\_BARRIERSCALE*  
*CLP\_PARAM\_STR\_GAMMA*  
*CLP\_PARAM\_STR\_CROSSOVER*  
*CLP\_PARAM\_STR\_PFI*  
*CLP\_PARAM\_NOTUSED\_ALGORITHM*  
*CBC\_PARAM\_STR\_NODESTRATEGY*  
*CBC\_PARAM\_STR\_BRANCHSTRATEGY*  
*CBC\_PARAM\_NOTUSED\_ADDCUTSSTRATEGY*  
*CBC\_PARAM\_STR\_GOMORYCUTS*  
*CBC\_PARAM\_STR\_PROBINGCUTS*  
*CBC\_PARAM\_STR\_KNAPSACKCUTS*  
*CBC\_PARAM\_NOTUSED\_ODDHOLECUTS*  
*CBC\_PARAM\_STR\_ROUNDING*  
*CBC\_PARAM\_STR\_SOLVER*  
*CBC\_PARAM\_STR\_CLIQUECUTS*  
*CBC\_PARAM\_STR\_COSTSTRATEGY*  
*CBC\_PARAM\_STR\_FLOWCUTS*  
*CBC\_PARAM\_STR\_MIXEDCUTS*  
*CBC\_PARAM\_STR\_TWOMIRCUTS*  
*CBC\_PARAM\_STR\_PREPROCESS*  
*CLP\_PARAM\_ACTION\_DIRECTORY*  
*CLP\_PARAM\_ACTION\_IMPORT*  
*CLP\_PARAM\_ACTION\_EXPORT*  
*CLP\_PARAM\_ACTION\_RESTORE*  
*CLP\_PARAM\_ACTION\_SAVE*  
*CLP\_PARAM\_ACTION\_DUALSIMPLEX*  
*CLP\_PARAM\_ACTION\_PRIMALSIMPLEX*  
*CLP\_PARAM\_ACTION\_MAXIMIZE*  
*CLP\_PARAM\_ACTION\_MINIMIZE*  
*CLP\_PARAM\_ACTION\_EXIT*  
*CLP\_PARAM\_ACTION\_STDIN*  
*CLP\_PARAM\_ACTION\_UNITTEST*  
*CLP\_PARAM\_ACTION\_NETLIB\_DUAL*  
*CLP\_PARAM\_ACTION\_NETLIB\_PRIMAL*  
*CLP\_PARAM\_ACTION\_SOLUTION*  
*CLP\_PARAM\_ACTION\_TIGHTEN*

***CLP\_PARAM\_ACTION\_FAKEBOUND***  
***CLP\_PARAM\_ACTION\_HELP***  
***CLP\_PARAM\_ACTION\_PLUSMINUS***  
***CLP\_PARAM\_ACTION\_NETWORK***  
***CLP\_PARAM\_ACTION\_ALLSLACK***  
***CLP\_PARAM\_ACTION\_REVERSE***  
***CLP\_PARAM\_ACTION\_BARRIER***  
***CLP\_PARAM\_ACTION\_NETLIB\_BARRIER***  
***CLP\_PARAM\_ACTION\_REALLY\_SCALE***  
***CLP\_PARAM\_ACTION\_BASISIN***  
***CLP\_PARAM\_ACTION\_BASISOUT***  
***CLP\_PARAM\_ACTION\_SOLVECONTINUOUS***  
***CBC\_PARAM\_ACTION\_BAB***  
***CBC\_PARAM\_ACTION\_MIPLIB***  
***CLP\_PARAM\_ACTION\_CLEARCUTS***  
***CLP\_VERSION\_NOTUSED\_PRINTVERSION***  
***CBC\_PARAM\_NOTUSED\_OSLSTUFF***  
***CBC\_PARAM\_NOTUSED\_CBCSTUFF***  
***CBC\_PARAM\_NOTUSED\_INVALID***

Definition at line 35 of file CbcParam.hpp.

## 8.70 /home/ted/COIN/trunk/Cbc/src/CbcPartialNodeInfo.hpp File Reference

```

#include <string>
#include <vector>
#include "CoinWarmStartBasis.hpp"
#include "CoinSearchTree.hpp"
#include "CbcBranchBase.hpp"
#include "CbcNodeInfo.hpp"

```

### Classes

- class [CbcPartialNodeInfo](#)

*Holds information for recreating a subproblem by incremental change from the parent.*

## 8.71 /home/ted/COIN/trunk/Cbc/src/CbcSimpleInteger.hpp File Reference

```

#include "CbcBranchingObject.hpp"

```

## Classes

- class [CbcIntegerBranchingObject](#)  
*Simple branching object for an integer variable.*
- class [CbcSimpleInteger](#)  
*Define a single integer class.*

## 8.72 /home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerDynamicPseudoCost.hpp File Reference

```
#include "CbcSimpleInteger.hpp"
```

## Classes

- class [CbcSimpleIntegerDynamicPseudoCost](#)  
*Define a single integer class but with dynamic pseudo costs.*
- class [CbcIntegerPseudoCostBranchingObject](#)  
*Simple branching object for an integer variable with pseudo costs.*

## Macros

- `#define` [TYPERATIO](#) 0.9
- `#define` [MINIMUM\\_MOVEMENT](#) 0.1
- `#define` [TYPE2](#) 0
- `#define` [INFEAS](#) 1
- `#define` [MOD\\_SHADOW](#) 1
- `#define` [WEIGHT\\_AFTER](#) 0.8
- `#define` [WEIGHT\\_BEFORE](#) 0.1
- `#define` [WEIGHT\\_PRODUCT](#)

### 8.72.1 Macro Definition Documentation

#### 8.72.1.1 `#define` TYPERATIO 0.9

Definition at line 13 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

#### 8.72.1.2 `#define` MINIMUM\_MOVEMENT 0.1

Definition at line 14 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

#### 8.72.1.3 `#define` TYPE2 0

Definition at line 15 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

#### 8.72.1.4 `#define` INFEAS 1

Definition at line 17 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

#### 8.72.1.5 `#define` MOD\_SHADOW 1

Definition at line 18 of file CbcSimpleIntegerDynamicPseudoCost.hpp.

#### 8.72.1.6 `#define WEIGHT_AFTER 0.8`

Definition at line 20 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

#### 8.72.1.7 `#define WEIGHT_BEFORE 0.1`

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

#### 8.72.1.8 `#define WEIGHT_PRODUCT`

Definition at line 23 of file `CbcSimpleIntegerDynamicPseudoCost.hpp`.

### 8.73 `/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerPseudoCost.hpp` File Reference

```
#include "CbcSimpleInteger.hpp"
```

#### Classes

- class [CbcSimpleIntegerPseudoCost](#)  
*Define a single integer class but with pseudo costs.*

### 8.74 `/home/ted/COIN/trunk/Cbc/src/CbcSolver.hpp` File Reference

Defines [CbcSolver](#), the proposed top-level class for the new-style cbc solver.

```
#include <string>
#include <vector>
#include "CoinMessageHandler.hpp"
#include "OsiClpSolverInterface.hpp"
#include "CbcModel.hpp"
#include "CbcOrClpParam.hpp"
```

#### Classes

- class [CbcSolver](#)  
*This allows the use of the standalone solver in a flexible manner.*
- struct [CbcSolverUsefulData](#)  
*Structure to hold useful arrays.*
- class [CbcUser](#)  
*A class to allow the use of unknown user functionality.*
- class [CbcStopNow](#)  
*Support the use of a call back class to decide whether to stop.*

#### 8.74.1 Detailed Description

Defines [CbcSolver](#), the proposed top-level class for the new-style cbc solver. This class is currently an orphan. With the removal of all code flagged with the `NEWS_STYLE_SOLVER`, this class is never instantiated (and cannot be instantiated). It is available to be coopted as a top-level object wrapping the current `CbcMain0` and `CbcMain1`, should that appear to be a desirable path forward. – lh, 091211 –



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

## 8.75 /home/ted/COIN/trunk/Cbc/src/CbcSolverAnalyze.hpp File Reference

Look to see if a constraint is all-integer (variables & coeffs), or could be all integer.

### Functions

- int \* [analyze](#) (OsiClpSolverInterface \*solverMod, int &numberChanged, double &increment, bool changeInt, CoinMessageHandler \*generalMessageHandler, bool noPrinting)

#### 8.75.1 Detailed Description

Look to see if a constraint is all-integer (variables & coeffs), or could be all integer.

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

#### 8.75.2 Function Documentation

8.75.2.1 int\* [analyze](#) ( OsiClpSolverInterface \* *solverMod*, int & *numberChanged*, double & *increment*, bool *changeInt*, CoinMessageHandler \* *generalMessageHandler*, bool *noPrinting* )

## 8.76 /home/ted/COIN/trunk/Cbc/src/CbcSolverExpandKnapsack.hpp File Reference

Expanding possibilities of x\*y, where x\*y are both integers, constructing a knapsack constraint.

### Functions

- OsiSolverInterface \* [expandKnapsack](#) (CoinModel &model, int \*whichColumn, int \*knapsackStart, int \*knapsackRow, int &numberKnapsack, CglStored &stored, int logLevel, int fixedPriority, int SOSPriority, CoinModel &tightenedModel)
- void [afterKnapsack](#) (const CoinModel &coinModel2, const int \*whichColumn, const int \*knapsackStart, const int \*knapsackRow, int numberKnapsack, const double \*knapsackSolution, double \*solution, int logLevel)

#### 8.76.1 Detailed Description

Expanding possibilities of x\*y, where x\*y are both integers, constructing a knapsack constraint. Results in a tighter model.

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

#### 8.76.2 Function Documentation

8.76.2.1 OsiSolverInterface\* [expandKnapsack](#) ( CoinModel & *model*, int \* *whichColumn*, int \* *knapsackStart*, int \* *knapsackRow*, int & *numberKnapsack*, CglStored & *stored*, int *logLevel*, int *fixedPriority*, int *SOSPriority*, CoinModel & *tightenedModel* )

8.76.2.2 void [afterKnapsack](#) ( const CoinModel & *coinModel2*, const int \* *whichColumn*, const int \* *knapsackStart*, const int \* *knapsackRow*, int *numberKnapsack*, const double \* *knapsackSolution*, double \* *solution*, int *logLevel* )

## 8.77 /home/ted/COIN/trunk/Cbc/src/CbcSolverHeuristics.hpp File Reference

Routines for doing heuristics.

### Functions

- void [crunchIt](#) (ClpSimplex \*model)
- OsiClpSolverInterface \* [fixVubs](#) ([CbcModel](#) &model, int skipZero2, int &doAction, CoinMessageHandler \*, const double \*lastSolution, double dextra[6], int extra[5])
- int [doHeuristics](#) ([CbcModel](#) \*model, int type, CbcOrClpParam \*parameters\_, int numberParameters\_, int noPrinting\_, int initialPumpTune)
  - 1 - add heuristics to model 2 - do heuristics (and set cutoff and best solution) 3 - for miplib test so skip some (out model later)*

### 8.77.1 Detailed Description

Routines for doing heuristics.

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

### 8.77.2 Function Documentation

#### 8.77.2.1 void crunchIt ( ClpSimplex \* model )

#### 8.77.2.2 OsiClpSolverInterface\* fixVubs ( CbcModel & model, int skipZero2, int & doAction, CoinMessageHandler \* , const double \* lastSolution, double dextra[6], int extra[5] )

#### 8.77.2.3 int doHeuristics ( CbcModel \* model, int type, CbcOrClpParam \* parameters\_, int numberParameters\_, int noPrinting\_, int initialPumpTune )

1 - add heuristics to model 2 - do heuristics (and set cutoff and best solution) 3 - for miplib test so skip some (out model later)

## 8.78 /home/ted/COIN/trunk/Cbc/src/CbcSOS.hpp File Reference

### Classes

- class [CbcSOS](#)
  - Branching object for Special Ordered Sets of type 1 and 2.*
- class [CbcSOSBranchingObject](#)
  - Branching object for Special ordered sets.*

## 8.79 /home/ted/COIN/trunk/Cbc/src/CbcStatistics.hpp File Reference

```
#include "CbcModel.hpp"
```

## Classes

- class [CbcStatistics](#)  
*For gathering statistics.*

## 8.80 /home/ted/COIN/trunk/Cbc/src/CbcStrategy.hpp File Reference

```
#include "CbcModel.hpp"
```

## Classes

- class [CbcStrategy](#)  
*Strategy base class.*
- class [CbcStrategyNull](#)  
*Null class.*
- class [CbcStrategyDefault](#)  
*Default class.*
- class [CbcStrategyDefaultSubTree](#)  
*Default class for sub trees.*

## 8.81 /home/ted/COIN/trunk/Cbc/src/CbcSubProblem.hpp File Reference

## 8.82 /home/ted/COIN/trunk/Cbc/src/CbcThread.hpp File Reference

```
#include "CbcModel.hpp"  
#include "CbcNode.hpp"
```

## Classes

- class [CbcThread](#)  
*A class to encapsulate thread stuff.*
- class [CbcBaseModel](#)  
*Base model.*

## 8.83 /home/ted/COIN/trunk/Cbc/src/CbcTree.hpp File Reference

```
#include <vector>  
#include <algorithm>  
#include <cmath>  
#include "CoinHelperFunctions.hpp"  
#include "CbcCompare.hpp"
```

## Classes

- class [CbcTree](#)

*Using MS heap implementation.*

### 8.84 /home/ted/COIN/trunk/Cbc/src/CbcTreeLocal.hpp File Reference

```
#include "CbcTree.hpp"
#include "CbcNode.hpp"
#include "OsiRowCut.hpp"
```

## Classes

- class [CbcTreeLocal](#)
- class [CbcTreeVariable](#)

### 8.85 /home/ted/COIN/trunk/Cbc/src/ClpAmplObjective.hpp File Reference

```
#include "ClpObjective.hpp"
#include "CoinPackedMatrix.hpp"
```

## Classes

- class [ClpAmplObjective](#)  
*Ampl Objective Class.*

### 8.86 /home/ted/COIN/trunk/Cbc/src/ClpConstraintAmpl.hpp File Reference

```
#include "ClpConstraint.hpp"
```

## Classes

- class [ClpConstraintAmpl](#)  
*Ampl Constraint Class.*

### 8.87 /home/ted/COIN/trunk/Cbc/src/config\_cbc\_default.h File Reference

## Macros

- #define [CBC\\_VERSION](#) "trunk"
- #define [CBC\\_VERSION\\_MAJOR](#) 9999
- #define [CBC\\_VERSION\\_MINOR](#) 9999
- #define [CBC\\_VERSION\\_RELEASE](#) 9999

### 8.87.1 Macro Definition Documentation

#### 8.87.1.1 `#define CBC_VERSION "trunk"`

Definition at line 8 of file config\_cbc\_default.h.

#### 8.87.1.2 `#define CBC_VERSION_MAJOR 9999`

Definition at line 11 of file config\_cbc\_default.h.

#### 8.87.1.3 `#define CBC_VERSION_MINOR 9999`

Definition at line 14 of file config\_cbc\_default.h.

#### 8.87.1.4 `#define CBC_VERSION_RELEASE 9999`

Definition at line 17 of file config\_cbc\_default.h.

## 8.88 /home/ted/COIN/trunk/Cbc/src/config\_default.h File Reference

```
#include "configall_system.h"
#include "config_cbc_default.h"
```

### Macros

- `#define COIN_CBC_CHECKLEVEL 0`
- `#define COIN_CBC_VERBOSITY 0`
- `#define COIN_HAS_CGL 1`
- `#define COIN_HAS_CLP 1`
- `#define COIN_HAS_COINUTILS 1`
- `#define COIN_HAS_OSI 1`
- `#define COIN_HAS_VOL 1`

### 8.88.1 Macro Definition Documentation

#### 8.88.1.1 `#define COIN_CBC_CHECKLEVEL 0`

Definition at line 14 of file config\_default.h.

#### 8.88.1.2 `#define COIN_CBC_VERBOSITY 0`

Definition at line 17 of file config\_default.h.

#### 8.88.1.3 `#define COIN_HAS_CGL 1`

Definition at line 20 of file config\_default.h.

#### 8.88.1.4 `#define COIN_HAS_CLP 1`

Definition at line 23 of file config\_default.h.

#### 8.88.1.5 `#define COIN_HAS_COINUTILS 1`

Definition at line 26 of file `config_default.h`.

#### 8.88.1.6 `#define COIN_HAS_OSI 1`

Definition at line 29 of file `config_default.h`.

#### 8.88.1.7 `#define COIN_HAS_VOL 1`

Definition at line 32 of file `config_default.h`.

### 8.89 `/home/ted/COIN/trunk/Cbc/src/OsiCbc/OsiCbcSolverInterface.hpp` File Reference

```
#include <string>
#include <cfloat>
#include <map>
#include "CbcModel.hpp"
#include "CoinPackedMatrix.hpp"
#include "OsiSolverInterface.hpp"
#include "CbcStrategy.hpp"
#include "CoinWarmStartBasis.hpp"
```

#### Classes

- class [OsiCbcSolverInterface](#)

*Cbc Solver Interface.*

#### Functions

- bool [OsiCbcHasNDEBUG](#) ()
- void [OsiCbcSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)

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

#### Variables

- static const double [OsiCbcInfinity](#) = COIN\_DBL\_MAX

#### 8.89.1 Function Documentation

##### 8.89.1.1 `bool OsiCbcHasNDEBUG ( )`

##### 8.89.1.2 `void OsiCbcSolverInterfaceUnitTest ( const std::string & mpsDir, const std::string & netlibDir )`

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

#### 8.89.2 Variable Documentation

8.89.2.1 `const double OsiCbcInfinity = COIN_DBL_MAX` `[static]`

Definition at line 20 of file OsiCbcSolverInterface.hpp.

## Index

### Symbols

- ~CbcBaseModel
  - CbcBaseModel, [28](#)
- ~CbcBranchAllDifferent
  - CbcBranchAllDifferent, [29](#)
- ~CbcBranchCut
  - CbcBranchCut, [31](#)
- ~CbcBranchDecision
  - CbcBranchDecision, [34](#)
- ~CbcBranchDefaultDecision
  - CbcBranchDefaultDecision, [37](#)
- ~CbcBranchDynamicDecision
  - CbcBranchDynamicDecision, [39](#)
- ~CbcBranchToFixLots
  - CbcBranchToFixLots, [48](#)
- ~CbcBranchingObject
  - CbcBranchingObject, [43](#)
- ~CbcCbcParam
  - CbcCbcParam, [53](#)
- ~CbcClique
  - CbcClique, [56](#)
- ~CbcCliqueBranchingObject
  - CbcCliqueBranchingObject, [59](#)
- ~CbcCompare
  - CbcCompare, [61](#)
- ~CbcCompareBase
  - CbcCompareBase, [62](#)
- ~CbcCompareDefault
  - CbcCompareDefault, [66](#)
- ~CbcCompareDepth
  - CbcCompareDepth, [70](#)
- ~CbcCompareEstimate
  - CbcCompareEstimate, [71](#)
- ~CbcCompareObjective
  - CbcCompareObjective, [72](#)
- ~CbcConsequence
  - CbcConsequence, [73](#)
- ~CbcCountRowCut
  - CbcCountRowCut, [75](#)
- ~CbcCutBranchingObject
  - CbcCutBranchingObject, [78](#)
- ~CbcCutGenerator
  - CbcCutGenerator, [83](#)
- ~CbcCutModifier
  - CbcCutModifier, [91](#)
- ~CbcCutSubsetModifier
  - CbcCutSubsetModifier, [92](#)
- ~CbcDummyBranchingObject
  - CbcDummyBranchingObject, [94](#)
- ~CbcDynamicPseudoCostBranchingObject
  - CbcDynamicPseudoCostBranchingObject, [97](#)
- ~CbcEventHandler
  - CbcEventHandler, [101](#)
- ~CbcFathom
  - CbcFathom, [104](#)
- ~CbcFathomDynamicProgramming
  - CbcFathomDynamicProgramming, [107](#)
- ~CbcFeasibilityBase
  - CbcFeasibilityBase, [110](#)
- ~CbcFixVariable
  - CbcFixVariable, [114](#)
- ~CbcFixingBranchingObject
  - CbcFixingBranchingObject, [112](#)
- ~CbcFollowOn
  - CbcFollowOn, [117](#)
- ~CbcFullNodeInfo
  - CbcFullNodeInfo, [120](#)
- ~CbcGenCtlBlk
  - CbcGenCtlBlk, [129](#)
- ~CbcGenParam
  - CbcGenParam, [143](#)
- ~CbcGeneral
  - CbcGeneral, [138](#)
- ~CbcHeuristic
  - CbcHeuristic, [148](#)
- ~CbcHeuristicCrossover
  - CbcHeuristicCrossover, [156](#)
- ~CbcHeuristicDINS
  - CbcHeuristicDINS, [159](#)
- ~CbcHeuristicDW
  - CbcHeuristicDW, [181](#)
- ~CbcHeuristicDive
  - CbcHeuristicDive, [163](#)
- ~CbcHeuristicDiveCoefficient
  - CbcHeuristicDiveCoefficient, [167](#)
- ~CbcHeuristicDiveFractional
  - CbcHeuristicDiveFractional, [169](#)
- ~CbcHeuristicDiveGuided
  - CbcHeuristicDiveGuided, [170](#)
- ~CbcHeuristicDiveLineSearch
  - CbcHeuristicDiveLineSearch, [172](#)
- ~CbcHeuristicDivePseudoCost
  - CbcHeuristicDivePseudoCost, [174](#)
- ~CbcHeuristicDiveVectorLength
  - CbcHeuristicDiveVectorLength, [176](#)
- ~CbcHeuristicDynamic3
  - CbcHeuristicDynamic3, [191](#)
- ~CbcHeuristicFPump
  - CbcHeuristicFPump, [195](#)
- ~CbcHeuristicGreedyCover
  - CbcHeuristicGreedyCover, [202](#)
- ~CbcHeuristicGreedyEquality



- CbcHeuristicGreedyEquality, 205
- ~CbcHeuristicGreedySOS
  - CbcHeuristicGreedySOS, 208
- ~CbcHeuristicJustOne
  - CbcHeuristicJustOne, 211
- ~CbcHeuristicLocal
  - CbcHeuristicLocal, 213
- ~CbcHeuristicNaive
  - CbcHeuristicNaive, 216
- ~CbcHeuristicNode
  - CbcHeuristicNode, 217
- ~CbcHeuristicNodeList
  - CbcHeuristicNodeList, 218
- ~CbcHeuristicPartial
  - CbcHeuristicPartial, 220
- ~CbcHeuristicPivotAndFix
  - CbcHeuristicPivotAndFix, 222
- ~CbcHeuristicProximity
  - CbcHeuristicProximity, 224
- ~CbcHeuristicRENS
  - CbcHeuristicRENS, 228
- ~CbcHeuristicRINS
  - CbcHeuristicRINS, 231
- ~CbcHeuristicRandRound
  - CbcHeuristicRandRound, 226
- ~CbcHeuristicVND
  - CbcHeuristicVND, 234
- ~CbcIdiotBranch
  - CbcIdiotBranch, 237
- ~CbcIntegerBranchingObject
  - CbcIntegerBranchingObject, 240
- ~CbcIntegerPseudoCostBranchingObject
  - CbcIntegerPseudoCostBranchingObject, 244
- ~CbcLongCliqueBranchingObject
  - CbcLongCliqueBranchingObject, 246
- ~CbcLotsize
  - CbcLotsize, 249
- ~CbcLotsizeBranchingObject
  - CbcLotsizeBranchingObject, 253
- ~CbcModel
  - CbcModel, 273
- ~CbcNWay
  - CbcNWay, 328
- ~CbcNWayBranchingObject
  - CbcNWayBranchingObject, 331
- ~CbcNode
  - CbcNode, 313
- ~CbcNodeInfo
  - CbcNodeInfo, 321
- ~CbcObject
  - CbcObject, 334
- ~CbcObjectUpdateData
  - CbcObjectUpdateData, 340
- ~CbcOsiParam, 345
- ~CbcOsiSolver
  - CbcOsiSolver, 347
- ~CbcParam
  - CbcParam, 350
- ~CbcPartialNodeInfo
  - CbcPartialNodeInfo, 355
- ~CbcRounding
  - CbcRounding, 358
- ~CbcRowCuts
  - CbcRowCuts, 360
- ~CbcSOS
  - CbcSOS, 392
- ~CbcSOSBranchingObject
  - CbcSOSBranchingObject, 396
- ~CbcSerendipity
  - CbcSerendipity, 361
- ~CbcSimpleInteger
  - CbcSimpleInteger, 364
- ~CbcSimpleIntegerDynamicPseudoCost
  - CbcSimpleIntegerDynamicPseudoCost, 371
- ~CbcSimpleIntegerPseudoCost
  - CbcSimpleIntegerPseudoCost, 382
- ~CbcSolver
  - CbcSolver, 386
- ~CbcStatistics
  - CbcStatistics, 398
- ~CbcStopNow
  - CbcStopNow, 401
- ~CbcStrategy
  - CbcStrategy, 403
- ~CbcStrategyDefault
  - CbcStrategyDefault, 407
- ~CbcStrategyDefaultSubTree
  - CbcStrategyDefaultSubTree, 409
- ~CbcStrategyNull
  - CbcStrategyNull, 411
- ~CbcThread
  - CbcThread, 414
- ~CbcTree
  - CbcTree, 417
- ~CbcTreeLocal
  - CbcTreeLocal, 423
- ~CbcTreeVariable
  - CbcTreeVariable, 426
- ~CbcUser
  - CbcUser, 430
- ~CglTemporary
  - CglTemporary, 432
- ~ClpAmplObjective
  - ClpAmplObjective, 435
- ~ClpConstraintAmpl
  - ClpConstraintAmpl, 437
- ~OsiBiLinear

- OsiBiLinear, [445](#)
- ~OsiBiLinearBranchingObject
  - OsiBiLinearBranchingObject, [452](#)
- ~OsiBiLinearEquality
  - OsiBiLinearEquality, [454](#)
- ~OsiCbcSolverInterface
  - OsiCbcSolverInterface, [461](#)
- ~OsiChooseStrongSubset
  - OsiChooseStrongSubset, [474](#)
- ~OsiLink
  - OsiLink, [477](#)
- ~OsiLinkBranchingObject
  - OsiLinkBranchingObject, [478](#)
- ~OsiLinkedBound
  - OsiLinkedBound, [480](#)
- ~OsiOldLink
  - OsiOldLink, [482](#)
- ~OsiOldLinkBranchingObject
  - OsiOldLinkBranchingObject, [484](#)
- ~OsiOneLink
  - OsiOneLink, [485](#)
- ~OsiSimpleFixedInteger
  - OsiSimpleFixedInteger, [487](#)
- ~OsiSolverLinearizedQuadratic
  - OsiSolverLinearizedQuadratic, [489](#)
- ~OsiSolverLink
  - OsiSolverLink, [495](#)
- ~OsiUsesBiLinear
  - OsiUsesBiLinear, [502](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchActual.hpp, [509](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchAllDifferent.-hpp, [509](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchBase.hpp, [510](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchCut.hpp, [511](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchDecision.hpp, [511](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchDefaultDecision.-hpp, [511](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchDynamic.hpp, [511](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchLotsize.hpp, [512](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchToFixLots.hpp, [513](#)
- /home/ted/COIN/trunk/Cbc/src/CbcBranchingObject.hpp, [512](#)
- /home/ted/COIN/trunk/Cbc/src/CbcClique.hpp, [513](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCompare.hpp, [513](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCompareActual.hpp, [513](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCompareBase.hpp, [514](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCompareDefault.hpp, [514](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCompareDepth.hpp, [514](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCompareEstimate.-hpp, [514](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCompareObjective.-hpp, [514](#)
- /home/ted/COIN/trunk/Cbc/src/CbcConfig.h, [515](#)
- /home/ted/COIN/trunk/Cbc/src/CbcConsequence.hpp, [515](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCountRowCut.hpp, [515](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCutGenerator.hpp, [515](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCutModifier.hpp, [516](#)
- /home/ted/COIN/trunk/Cbc/src/CbcCutSubsetModifier.-hpp, [516](#)
- /home/ted/COIN/trunk/Cbc/src/CbcDummyBranching-Object.hpp, [516](#)
- /home/ted/COIN/trunk/Cbc/src/CbcEventHandler.hpp, [517](#)
- /home/ted/COIN/trunk/Cbc/src/CbcFathom.hpp, [517](#)
- /home/ted/COIN/trunk/Cbc/src/CbcFathomDynamic-Programming.hpp, [518](#)
- /home/ted/COIN/trunk/Cbc/src/CbcFeasibilityBase.hpp, [518](#)
- /home/ted/COIN/trunk/Cbc/src/CbcFixVariable.hpp, [518](#)
- /home/ted/COIN/trunk/Cbc/src/CbcFollowOn.hpp, [518](#)
- /home/ted/COIN/trunk/Cbc/src/CbcFullNodeInfo.hpp, [519](#)
- /home/ted/COIN/trunk/Cbc/src/CbcGenCbcParam.hpp, [519](#)
- /home/ted/COIN/trunk/Cbc/src/CbcGenCtlBlk.hpp, [519](#)
- /home/ted/COIN/trunk/Cbc/src/CbcGenMessages.hpp, [521](#)
- /home/ted/COIN/trunk/Cbc/src/CbcGenOsiParam.hpp, [522](#)
- /home/ted/COIN/trunk/Cbc/src/CbcGenParam.hpp, [522](#)
- /home/ted/COIN/trunk/Cbc/src/CbcGeneral.hpp, [521](#)
- /home/ted/COIN/trunk/Cbc/src/CbcGeneralDepth.hpp, [521](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristic.hpp, [523](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDINS.hpp, [523](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDW.hpp, [525](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDive.hpp, [524](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveCoefficient.-hpp, [524](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveFractional.-hpp, [524](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveGuided.-hpp, [524](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveLine-Search.hpp, [525](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDivePseudo-Cost.hpp, [525](#)
- /home/ted/COIN/trunk/Cbc/src/CbcHeuristicDiveVector-Length.hpp, [525](#)

- [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicFPump.hpp, 525](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicGreedy.hpp, 526](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicLocal.hpp, 526](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicPivotAndFix.-hpp, 526](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRENS.hpp, 527](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRINS.hpp, 527](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicRandRound.-hpp, 526](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcHeuristicVND.hpp, 527](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcLinked.hpp, 527](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcMessage.hpp, 529](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcMipStartIO.hpp, 531](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcModel.hpp, 531](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcNWay.hpp, 533](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcNode.hpp, 533](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcNodeInfo.hpp, 533](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcObject.hpp, 533](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcObjectUpdateData.-hpp, 534](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcParam.hpp, 534](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcPartialNodeInfo.hpp, 538](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSOS.hpp, 542](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSimpleInteger.hpp, 538](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerDynamic-PseudoCost.hpp, 539](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSimpleIntegerPseudo-Cost.hpp, 540](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSolver.hpp, 540](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSolverAnalyze.hpp, 541](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSolverExpandKnapsack.-hpp, 541](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSolverHeuristics.hpp, 542](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcStatistics.hpp, 542](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcStrategy.hpp, 543](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcSubProblem.hpp, 543](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcThread.hpp, 543](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcTree.hpp, 543](#)
  - [/home/ted/COIN/trunk/Cbc/src/CbcTreeLocal.hpp, 544](#)
  - [/home/ted/COIN/trunk/Cbc/src/Cbc\\_C\\_Interface.h, 504](#)
  - [/home/ted/COIN/trunk/Cbc/src/Cbc\\_ampl.h, 504](#)
  - [/home/ted/COIN/trunk/Cbc/src/ClpAmplObjective.hpp, 544](#)
  - [/home/ted/COIN/trunk/Cbc/src/ClpConstraintAmpl.hpp, 544](#)
  - [/home/ted/COIN/trunk/Cbc/src/OsiCbc/OsiCbcSolver-Interface.hpp, 546](#)
  - [/home/ted/COIN/trunk/Cbc/src/config\\_cbc\\_default.h, 544](#)
  - [/home/ted/COIN/trunk/Cbc/src/config\\_default.h, 545](#)
- A
- ALGORITHM
    - [CbcOsiParam, 343](#)
  - ALLOWABLEGAP
    - [CbcCbcParam, 51](#)
  - ALLSLACK
    - [CbcOsiParam, 343](#)
  - AUTOSCALE
    - [CbcOsiParam, 343](#)
  - absoluteIncrement
    - [CbcHeuristicFPump, 196](#)
  - absoluteIncrement\_
    - [CbcHeuristicFPump, 199](#)
  - accumulate
    - [CbcHeuristicFPump, 198](#)
  - accumulate\_
    - [CbcHeuristicFPump, 200](#)
  - action\_
    - [CbcGenCtlBlk::djFixCtl\\_struct, 440](#)
  - active
    - [CbcNode, 317](#)
  - active\_
    - [CbcNodeInfo, 326](#)
  - addCuts
    - [CbcEventHandler, 101](#)
  - addBiLinearObjects
    - [OsiUsesBiLinear, 503](#)
  - addBoundModifier
    - [OsiLinkedBound, 480](#)
    - [OsiSolverLink, 495](#)
  - addBranchingInformation
    - [CbcTree, 420](#)
  - addCbcCbcParams
    - [CbcCbcParamUtils, 20](#)
  - addCbcGenParams
    - [CbcGenParamUtils, 21](#)
  - addCbcOsiParams
    - [CbcOsiParamUtils, 22](#)
  - addCol
    - [OsiCbcSolverInterface, 469](#)
  - addCols
    - [OsiCbcSolverInterface, 469](#)
  - addCutGenerator
    - [CbcModel, 296](#)
    - [CbcSolver, 387](#)
  - addCutIfNotDuplicate
    - [CbcRowCuts, 360](#)
  - addCutIfNotDuplicateWhenGreedy
    - [CbcRowCuts, 360](#)

- addCuts
  - CbcModel, 306
  - CbcNodeInfo, 323
  - CbcRowCuts, 360
- addCuts1
  - CbcModel, 306
- addExtraRow
  - OsiBiLinear, 449
- addHelp
  - CbcParam, 350
- addHeuristic
  - CbcHeuristicJustOne, 212
  - CbcModel, 297
- AddIntegers
  - CbcModel, 275
- addObjects
  - CbcModel, 276
- addRow
  - OsiCbcSolverInterface, 469
- addRows
  - OsiCbcSolverInterface, 469
- addStatistics
  - CbcCutGenerator, 89
- addTighterConstraints
  - OsiSolverLink, 496
- addToSumDownChange
  - CbcSimpleIntegerDynamicPseudoCost, 374
- addToSumDownCost
  - CbcSimpleIntegerDynamicPseudoCost, 374
- addToSumDownDecrease
  - CbcSimpleIntegerDynamicPseudoCost, 375
- addToSumUpChange
  - CbcSimpleIntegerDynamicPseudoCost, 375
- addToSumUpCost
  - CbcSimpleIntegerDynamicPseudoCost, 374
- addToSumUpDecrease
  - CbcSimpleIntegerDynamicPseudoCost, 375
- addUpdateInformation
  - CbcModel, 274
- addUserFunction
  - CbcSolver, 387
- addedCuts
  - CbcModel, 308
- adjustHeuristics
  - CbcModel, 307
- affinity\_
  - CbcHeuristicDW, 188
- afterHeuristic
  - CbcEventHandler, 101
- afterKnapsack
  - CbcSolverExpandKnapsack.hpp, 541
- afterNodeNumber\_
  - CbcCompareDefault, 69
- algorithm
  - CbcHeuristicGreedyCover, 203
  - CbcHeuristicGreedyEquality, 206
  - CbcHeuristicGreedySOS, 209
- algorithm\_
  - CbcFathomDynamicProgramming, 110
  - CbcHeuristicGreedyCover, 203
  - CbcHeuristicGreedyEquality, 206
  - CbcHeuristicGreedySOS, 209
- allActivated
  - CbcNodeInfo, 325
- allBranchesGone
  - CbcNodeInfo, 322
- allDynamic
  - CbcModel, 310
- allowImportErrors\_
  - CbcGenCtlBlk, 135
- alternateTest
  - CbcCompare, 61
  - CbcCompareBase, 64
- alwaysCreate\_
  - CbcBranchToFixLots, 50
- ampl\_info, 22
  - arguments, 26
  - branchDirection, 25
  - buffer, 26
  - columnLower, 24
  - columnStatus, 25
  - columnUpper, 24
  - cut, 26
  - direction, 24
  - dualSolution, 25
  - elements, 25
  - logLevel, 26
  - nonLinear, 26
  - numberArguments, 24
  - numberBinary, 24
  - numberColumns, 23
  - numberElements, 24
  - numberIntegers, 24
  - numberRows, 23
  - numberSos, 24
  - objValue, 24
  - objective, 24
  - offset, 24
  - primalSolution, 25
  - priorities, 25
  - problemStatus, 24
  - pseudoDown, 25
  - pseudoUp, 25
  - rowLower, 24
  - rowStatus, 25
  - rowUpper, 24
  - rows, 25
  - sosIndices, 26

- sosPriority, [25](#)
- sosReference, [26](#)
- sosStart, [25](#)
- sosType, [25](#)
- special, [26](#)
- starts, [25](#)
- ampl\_obj\_prec
  - Cbc\_ampl.h, [504](#)
- analyze
  - CbcNode, [315](#)
  - CbcSolver, [387](#)
  - CbcSolverAnalyze.hpp, [541](#)
- analyzeObjective
  - CbcModel, [275](#)
- analyzeObjects
  - OsiSolverLink, [496](#)
- answerSolver\_
  - CbcGenCtIBlk::babState\_struct, [27](#)
- append
  - CbcHeuristicNodeList, [218](#)
  - CbcParam, [350](#)
- applyBounds
  - CbcFullNodeInfo, [120](#)
  - CbcNodeInfo, [322](#)
  - CbcPartialNodeInfo, [355](#)
- applyColCut
  - OsiCbcSolverInterface, [473](#)
- applyConsequence
  - CbcNWay, [328](#)
- applyRowCut
  - OsiCbcSolverInterface, [473](#)
- applyRowCuts
  - OsiCbcSolverInterface, [469](#)
- applyToModel
  - CbcFullNodeInfo, [120](#)
  - CbcNodeInfo, [321](#)
  - CbcPartialNodeInfo, [355](#)
- applyToSolver
  - CbcConsequence, [74](#)
  - CbcFixVariable, [115](#)
- approximateSolution
  - CbcLinked.hpp, [528](#)
- arguments
  - ampl\_info, [26](#)
- artificialCost
  - CbcHeuristicFPump, [197](#)
- artificialCost\_
  - CbcHeuristicFPump, [200](#)
- assignProblem
  - OsiCbcSolverInterface, [470](#)
- assignSolver
  - CbcModel, [302](#)
- atSolution
  - CbcCutGenerator, [85](#)
- attempts\_
  - CbcHeuristicCrossover, [157](#)
- avgDistance
  - CbcHeuristicNode, [218](#)
- B
- BAB
  - CbcGenParam, [141](#)
- BACAbandon
  - CbcGenCtIBlk, [128](#)
- BACFinish
  - CbcGenCtIBlk, [128](#)
- BACInvalid
  - CbcGenCtIBlk, [128](#)
- BACNotRun
  - CbcGenCtIBlk, [128](#)
- BACStop
  - CbcGenCtIBlk, [128](#)
- BACUser
  - CbcGenCtIBlk, [128](#)
- BACmFinish
  - CbcGenCtIBlk, [129](#)
- BACmGap
  - CbcGenCtIBlk, [129](#)
- BACmInfeas
  - CbcGenCtIBlk, [129](#)
- BACmInvalid
  - CbcGenCtIBlk, [129](#)
- BACmNodeLimit
  - CbcGenCtIBlk, [129](#)
- BACmOther
  - CbcGenCtIBlk, [129](#)
- BACmSolnLimit
  - CbcGenCtIBlk, [129](#)
- BACmTimeLimit
  - CbcGenCtIBlk, [129](#)
- BACmUbnd
  - CbcGenCtIBlk, [129](#)
- BACmUser
  - CbcGenCtIBlk, [129](#)
- BACwBAC
  - CbcGenCtIBlk, [129](#)
- BACwBareRoot
  - CbcGenCtIBlk, [129](#)
- BACwIPP
  - CbcGenCtIBlk, [129](#)
- BACwIPPRelax
  - CbcGenCtIBlk, [129](#)
- BACwInvalid
  - CbcGenCtIBlk, [129](#)
- BACwNotStarted
  - CbcGenCtIBlk, [129](#)
- BARRIER
  - CbcOsiParam, [343](#)

- BARRIERSCALE
  - CbcOsiParam, [343](#)
- BASISIN
  - CbcOsiParam, [343](#)
- BASISOUT
  - CbcOsiParam, [343](#)
- BIASLU
  - CbcOsiParam, [343](#)
- BPCost
  - CbcGenCtlBlk, [127](#)
- BPExt
  - CbcGenCtlBlk, [128](#)
- BPOff
  - CbcGenCtlBlk, [127](#)
- BPOrder
  - CbcGenCtlBlk, [128](#)
- BACMajor
  - CbcGenCtlBlk, [128](#)
- BACMinor
  - CbcGenCtlBlk, [128](#)
- BACWhere
  - CbcGenCtlBlk, [129](#)
- BPControl
  - CbcGenCtlBlk, [127](#)
- bab\_
  - CbcGenCtlBlk, [137](#)
- babModel
  - CbcSolver, [388](#)
- back
  - CbcFathomDynamicProgramming, [108](#)
- back\_
  - CbcFathomDynamicProgramming, [109](#)
- backwardRow\_
  - CbcHeuristicDW, [187](#)
- baseSolution
  - CbcHeuristicVND, [235](#)
- baseSolution\_
  - CbcHeuristicVND, [236](#)
- basis\_
  - CbcFullNodeInfo, [121](#)
- basisDiff
  - CbcPartialNodeInfo, [355](#)
- basisDiff\_
  - CbcPartialNodeInfo, [356](#)
- beforeSolution1
  - CbcEventHandler, [101](#)
- beforeSolution2
  - CbcEventHandler, [101](#)
- bestAlternate
  - CbcTree, [419](#)
- bestBranch
  - CbcBranchDecision, [34](#)
  - CbcBranchDefaultDecision, [37](#)
- bestNode
  - CbcTree, [418](#)
- bestObjective
  - CbcHeuristicDW, [182](#)
- bestObjective\_
  - CbcHeuristicDW, [186](#)
- bestObjectiveValue
  - OsiSolverLinearizedQuadratic, [490](#)
  - OsiSolverLink, [496](#)
- bestObjectiveValue\_
  - OsiSolverLinearizedQuadratic, [490](#)
  - OsiSolverLink, [500](#)
- bestPossible\_
  - CbcCompareDefault, [68](#)
- bestSolution
  - CbcHeuristicDW, [182](#)
  - CbcModel, [291](#)
  - OsiSolverLinearizedQuadratic, [490](#)
  - OsiSolverLink, [496](#)
- bestSolution\_
  - CbcHeuristicDW, [186](#)
  - OsiSolverLinearizedQuadratic, [491](#)
  - OsiSolverLink, [500](#)
- betterBranch
  - CbcBranchDecision, [34](#)
  - CbcBranchDefaultDecision, [37](#)
  - CbcBranchDynamicDecision, [39](#)
- biLinearPriority
  - OsiSolverLink, [497](#)
- biLinearPriority\_
  - OsiSolverLink, [500](#)
- binVarIndex\_
  - CbcHeuristicDive, [166](#)
- bitPattern\_
  - CbcFathomDynamicProgramming, [110](#)
- bound
  - CbcLotsize, [251](#)
- boundBranch
  - CbcBranchCut, [31](#)
  - CbcCutBranchingObject, [78](#)
  - OsiBiLinear, [446](#)
  - OsiBiLinearBranchingObject, [453](#)
  - OsiLink, [477](#)
  - OsiOldLink, [483](#)
- boundType
  - OsiBiLinear, [448](#)
- boundType\_
  - OsiBiLinear, [450](#)
- branch
  - CbcBranchingObject, [44](#)
  - CbcCliqueBranchingObject, [59](#)
  - CbcCutBranchingObject, [78](#)
  - CbcDummyBranchingObject, [94](#)
  - CbcDynamicPseudoCostBranchingObject, [98](#)
  - CbcFixingBranchingObject, [112](#)

- CbcIntegerBranchingObject, [241](#)
- CbcIntegerPseudoCostBranchingObject, [244](#)
- CbcLongCliqueBranchingObject, [247](#)
- CbcLotsizeBranchingObject, [253](#)
- CbcNode, [315](#)
- CbcNWayBranchingObject, [331](#)
- CbcSOSBranchingObject, [396](#)
- OsiBiLinearBranchingObject, [453](#)
- OsiLinkBranchingObject, [479](#)
- OsiOldLinkBranchingObject, [484](#)
- branchAndBound
  - CbcModel, [273](#)
  - OsiCbcSolverInterface, [462](#)
- branchDirection
  - ampl\_info, [25](#)
- branchDirection\_
  - CbcSolverUsefulData, [390](#)
- branched
  - CbcTree, [420](#)
- branched\_
  - CbcTree, [421](#)
- branchedOn
  - CbcNodeInfo, [323](#)
- branchingMethod
  - CbcModel, [295](#)
- branchingObject
  - CbcNode, [317](#)
- branchingStrategy
  - OsiBiLinear, [448](#)
- branchingStrategy\_
  - OsiBiLinear, [450](#)
- branchingValue\_
  - CbcObjectUpdateData, [341](#)
- breadthDepth\_
  - CbcCompareDefault, [68](#)
- breakEven
  - CbcSimpleInteger, [366](#)
- breakEven\_
  - CbcSimpleInteger, [366](#)
- buffer
  - ampl\_info, [26](#)
- buildCut
  - CbcIdiotBranch, [238](#)
- buildRowBasis
  - CbcFullNodeInfo, [120](#)
  - CbcNodeInfo, [322](#)
  - CbcPartialNodeInfo, [355](#)
- C
- CBC\_BRANCH
  - CbcMessage.hpp, [530](#)
- CBC\_CUTOFF\_WARNING1
  - CbcMessage.hpp, [530](#)
- CBC\_CUTS\_STATS
  - CbcMessage.hpp, [530](#)
- CBC\_DUMMY\_END
  - CbcMessage.hpp, [530](#)
- CBC\_END
  - CbcMessage.hpp, [529](#)
- CBC\_END\_GOOD
  - CbcMessage.hpp, [529](#)
- CBC\_END\_SOLUTION
  - CbcMessage.hpp, [529](#)
- CBC\_END\_SUB
  - CbcMessage.hpp, [530](#)
- CBC\_EVENT
  - CbcMessage.hpp, [529](#)
- CBC\_FATHOM\_CHANGE
  - CbcMessage.hpp, [530](#)
- CBC\_FPUMP1
  - CbcMessage.hpp, [530](#)
- CBC\_FPUMP2
  - CbcMessage.hpp, [530](#)
- CBC\_GAP
  - CbcMessage.hpp, [530](#)
- CBC\_GENERAL
  - CbcMessage.hpp, [530](#)
- CBC\_GENERATOR
  - CbcMessage.hpp, [530](#)
- CBC\_HEURISTICS\_OFF
  - CbcMessage.hpp, [530](#)
- CBC\_INFEAS
  - CbcMessage.hpp, [529](#)
- CBC\_INTEGERINCREMENT
  - CbcMessage.hpp, [530](#)
- CBC\_ITERATE\_STRONG
  - CbcMessage.hpp, [530](#)
- CBC\_MAXITERS
  - CbcMessage.hpp, [529](#)
- CBC\_MAXNODES
  - CbcMessage.hpp, [529](#)
- CBC\_MAXSOLS
  - CbcMessage.hpp, [529](#)
- CBC\_MAXTIME
  - CbcMessage.hpp, [529](#)
- CBC\_NOINT
  - CbcMessage.hpp, [530](#)
- CBC\_NOTFEAS1
  - CbcMessage.hpp, [530](#)
- CBC\_NOTFEAS2
  - CbcMessage.hpp, [530](#)
- CBC\_NOTFEAS3
  - CbcMessage.hpp, [530](#)
- CBC\_OTHER\_STATS
  - CbcMessage.hpp, [530](#)
- CBC\_OTHER\_STATS2
  - CbcMessage.hpp, [530](#)
- CBC\_PARAM\_ACTION\_BAB



- CbcParam.hpp, [538](#)
- CBC\_PARAM\_ACTION\_MIPLIB
  - CbcParam.hpp, [538](#)
- CBC\_PARAM\_DBL\_ALLOWABLEGAP
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_DBL\_CUTOFF
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_DBL\_DJFIX
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_DBL\_GAPRATIO
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_DBL\_INCREMENT
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_DBL\_INFEASIBILITYWEIGHT
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_DBL\_INTEGERTOLERANCE
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_DBL\_TIGHTENFACTOR
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_FULLGENERALQUERY
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_GENERALQUERY
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_INT\_MAXNODES
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_INT\_STRONGBRANCHING
  - CbcParam.hpp, [536](#)
- CBC\_PARAM\_NOTUSED\_ADDCUTSSTRATEGY
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_NOTUSED\_CBCSTUFF
  - CbcParam.hpp, [538](#)
- CBC\_PARAM\_NOTUSED\_INVALID
  - CbcParam.hpp, [538](#)
- CBC\_PARAM\_NOTUSED\_ODDHOLECUTS
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_NOTUSED\_OSLSTUFF
  - CbcParam.hpp, [538](#)
- CBC\_PARAM\_STR\_BRANCHSTRATEGY
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_CLIQUECUTS
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_COSTSTRATEGY
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_FLOWCUTS
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_GOMORYCUTS
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_KNAPSACKCUTS
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_MIXEDCUTS
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_NODESTRATEGY
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_PREPROCESS
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_PROBINGCUTS
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_ROUNDING
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_SOLVER
  - CbcParam.hpp, [537](#)
- CBC\_PARAM\_STR\_TWOMIRCUTS
  - CbcParam.hpp, [537](#)
- CBC\_PRIORITY
  - CbcMessage.hpp, [530](#)
- CBC\_RELAXED1
  - CbcMessage.hpp, [530](#)
- CBC\_RELAXED2
  - CbcMessage.hpp, [530](#)
- CBC\_RESTART
  - CbcMessage.hpp, [530](#)
- CBC\_ROOT
  - CbcMessage.hpp, [530](#)
- CBC\_ROOT\_DETAIL
  - CbcMessage.hpp, [530](#)
- CBC\_ROUNDING
  - CbcMessage.hpp, [530](#)
- CBC\_SOLINDIVIDUAL
  - CbcMessage.hpp, [530](#)
- CBC\_SOLUTION
  - CbcMessage.hpp, [529](#)
- CBC\_SOLUTION2
  - CbcMessage.hpp, [529](#)
- CBC\_START\_SUB
  - CbcMessage.hpp, [530](#)
- CBC\_STATUS
  - CbcMessage.hpp, [530](#)
- CBC\_STATUS2
  - CbcMessage.hpp, [530](#)
- CBC\_STATUS3
  - CbcMessage.hpp, [530](#)
- CBC\_STRONG
  - CbcMessage.hpp, [529](#)
- CBC\_STRONG\_STATS
  - CbcMessage.hpp, [530](#)
- CBC\_STRONGSOL
  - CbcMessage.hpp, [530](#)
- CBC\_THREAD\_STATS
  - CbcMessage.hpp, [530](#)
- CBC\_TREE\_SOL
  - CbcMessage.hpp, [530](#)
- CBC\_UNBOUNDED
  - CbcMessage.hpp, [530](#)
- CBC\_VUB\_END
  - CbcMessage.hpp, [530](#)
- CBC\_VUB\_PASS
  - CbcMessage.hpp, [530](#)
- CBC\_WARNING\_STRONG



CbcMessage.hpp, [530](#)  
CBCCBC\_FIRSTPARAM  
    CbcCbcParam, [51](#)  
CBCCBC\_LASTPARAM  
    CbcCbcParam, [52](#)  
CBCGEN\_CONFUSION  
    CbcGenMessages.hpp, [522](#)  
CBCGEN\_DUMMY\_END  
    CbcGenMessages.hpp, [522](#)  
CBCGEN\_FIRSTPARAM  
    CbcGenParam, [140](#)  
CBCGEN\_LASTPARAM  
    CbcGenParam, [142](#)  
CBCGEN\_NEW\_SOLVER  
    CbcGenMessages.hpp, [521](#)  
CBCGEN\_TEST\_MSG  
    CbcGenMessages.hpp, [521](#)  
CBCOSI\_FIRSTPARAM  
    CbcOsiParam, [343](#)  
CBCOSI\_LASTPARAM  
    CbcOsiParam, [344](#)  
CGForceBut  
    CbcGenCtlBlk, [127](#)  
CGForceOn  
    CbcGenCtlBlk, [127](#)  
CGIfMove  
    CbcGenCtlBlk, [127](#)  
CGMarker  
    CbcGenCtlBlk, [127](#)  
CGOff  
    CbcGenCtlBlk, [127](#)  
CGOn  
    CbcGenCtlBlk, [127](#)  
CGRoot  
    CbcGenCtlBlk, [127](#)  
CHOLESKY  
    CbcOsiParam, [343](#)  
CLEARCUTS  
    CbcGenParam, [141](#)  
CLIQUECUTS  
    CbcGenParam, [141](#)  
CLP\_PARAM\_ACTION\_ALLSLACK  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_BARRIER  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_BASISIN  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_BASISOUT  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_CLEARCUTS  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_DIRECTORY  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_DUALSIMPLEX  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_EXIT  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_EXPORT  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_FAKEBOUND  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_HELP  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_IMPORT  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_MAXIMIZE  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_MINIMIZE  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_NETLIB\_BARRIER  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_NETLIB\_DUAL  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_NETLIB\_PRIMAL  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_NETWORK  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_PLUSMINUS  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_PRIMALSIMPLEX  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_REALLY\_SCALE  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_RESTORE  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_REVERSE  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_SAVE  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_SOLUTION  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_SOLVECONTINUOUS  
    CbcParam.hpp, [538](#)  
CLP\_PARAM\_ACTION\_STDIN  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_TIGHTEN  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_ACTION\_UNITTEST  
    CbcParam.hpp, [537](#)  
CLP\_PARAM\_DBL\_DUALBOUND  
    CbcParam.hpp, [536](#)  
CLP\_PARAM\_DBL\_DUALTOLERANCE  
    CbcParam.hpp, [536](#)  
CLP\_PARAM\_DBL\_OBJSCALE  
    CbcParam.hpp, [536](#)  
CLP\_PARAM\_DBL\_PRIMALTOLERANCE  
    CbcParam.hpp, [536](#)  
CLP\_PARAM\_DBL\_PRIMALWEIGHT

CbcParam.hpp, [536](#)  
 CLP\_PARAM\_DBL\_RHSSCALE  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_DBL\_TIMELIMIT  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_IDIOT  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_LOGLEVEL  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_MAXFACTOR  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_MAXITERATION  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_OUTPUTFORMAT  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_PERTVALUE  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_PREOLVEOPTIONS  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_PREOLVEPASS  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_PRINTOPTIONS  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_SLPVALUE  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_SOLVERLOGLEVEL  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_SPECIALOPTIONS  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_INT\_SPRINT  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_NOTUSED\_ALGORITHM  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_AUTOSCALE  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_BARRIERSCALE  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_BIASLU  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_CHOLESKY  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_CRASH  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_CROSSOVER  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_DIRECTION  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_DUALPIVOT  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_ERRORALLOWED  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_GAMMA  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_KEEPNAMES

CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_KKT  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_MESSAGES  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_PERTURBATION  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_PFI  
     CbcParam.hpp, [537](#)  
 CLP\_PARAM\_STR\_PREOLVE  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_PRIMALPIVOT  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_SCALING  
     CbcParam.hpp, [536](#)  
 CLP\_PARAM\_STR\_SPARSEFACTOR  
     CbcParam.hpp, [536](#)  
 CLP\_VERSION\_NOTUSED\_PRINTVERSION  
     CbcParam.hpp, [538](#)  
 COMBINE  
     CbcGenParam, [141](#)  
 COSTSTRATEGY  
     CbcCbcParam, [52](#)  
     CbcGenParam, [141](#)  
 CPP  
     CbcGenParam, [141](#)  
 CRASH  
     CbcOsiParam, [343](#)  
 CROSSOVER  
     CbcOsiParam, [343](#)  
 CUTDEPTH  
     CbcCbcParam, [52](#)  
     CbcGenParam, [141](#)  
 CUTOFF  
     CbcCbcParam, [52](#)  
 CUTPASS  
     CbcCbcParam, [52](#)  
 CUTSTRATEGY  
     CbcGenParam, [141](#)  
 CBC\_Message  
     CbcMessage.hpp, [529](#)  
 CBC\_VERSION  
     config\_cbc\_default.h, [545](#)  
 CBC\_VERSION\_MAJOR  
     config\_cbc\_default.h, [545](#)  
 CBC\_VERSION\_MINOR  
     config\_cbc\_default.h, [545](#)  
 CGControl  
     CbcGenCtlBlk, [127](#)  
 COIN\_CBC\_VERBOSITY  
     config\_default.h, [545](#)  
 COIN\_HAS\_CGL  
     config\_default.h, [545](#)  
 COIN\_HAS\_CLP

- config\_default.h, [545](#)
- COIN\_HAS\_COINUTILS
  - config\_default.h, [545](#)
- COIN\_HAS\_OSI
  - config\_default.h, [546](#)
- COIN\_HAS\_VOL
  - config\_default.h, [546](#)
- COINLINKAGE
  - Cbc\_C\_Interface.h, [506](#)
- callBack
  - CbcStopNow, [401](#)
- callCbc
  - CbcModel.hpp, [532](#)
- callCbc1
  - CbcModel.hpp, [532](#)
- canDealWithOdd
  - CbcHeuristic, [151](#)
  - CbcHeuristicDynamic3, [192](#)
- canDo
  - CbcUser, [431](#)
- canDoHeuristics
  - CbcBranchToFixLots, [49](#)
  - CbcLotsize, [251](#)
  - CbcSOS, [394](#)
  - OsiBiLinear, [446](#)
  - OsiLink, [477](#)
  - OsiOldLink, [482](#)
- canDropCut
  - CbcCountRowCut, [76](#)
- canFix\_
  - CbcCutBranchingObject, [79](#)
- canHeuristicRun
  - CbcHeuristicDive, [165](#)
  - CbcHeuristicDiveGuided, [171](#)
- canStopOnGap
  - CbcModel, [280](#)
- CbcAllowableFractionGap
  - CbcModel, [272](#)
- CbcAllowableGap
  - CbcModel, [272](#)
- CbcBranchBase.hpp
  - CbcRangeDisjoint, [510](#)
  - CbcRangeOverlap, [510](#)
  - CbcRangeSame, [510](#)
  - CbcRangeSubset, [510](#)
  - CbcRangeSuperset, [510](#)
- CbcBranchingObject.hpp
  - CliqueBranchObj, [512](#)
  - CutBranchingObj, [512](#)
  - DummyBranchObj, [512](#)
  - DynamicPseudoCostBranchObj, [512](#)
  - FollowOnBranchObj, [512](#)
  - GeneralDepthBranchObj, [512](#)
  - LongCliqueBranchObj, [512](#)
  - LotsizeBranchObj, [512](#)
  - NWayBranchObj, [512](#)
  - OneGeneralBranchingObj, [512](#)
  - SimpleIntegerBranchObj, [512](#)
  - SimpleIntegerDynamicPseudoCostBranchObj, [512](#)
  - SoSBranchObj, [512](#)
- CbcCbcParam
  - ALLOWABLEGAP, [51](#)
  - CBCCBC\_FIRSTPARAM, [51](#)
  - CBCCBC\_LASTPARAM, [52](#)
  - COSTSTRATEGY, [52](#)
  - CUTDEPTH, [52](#)
  - CUTOFF, [52](#)
  - CUTPASS, [52](#)
  - DIRECTION, [52](#)
  - GAPRATIO, [52](#)
  - INCREMENT, [52](#)
  - INFEASIBILITYWEIGHT, [52](#)
  - INTEGERTOLERANCE, [52](#)
  - LOGLEVEL, [52](#)
  - MAXIMIZE, [52](#)
  - MAXNODES, [52](#)
  - MINIMIZE, [52](#)
  - MIOPTIONS, [52](#)
  - MOREMIOPTIONS, [52](#)
  - NUMBERANALYZE, [52](#)
  - NUMBERBEFORE, [52](#)
  - NUMBERMINI, [52](#)
  - STRONGBRANCHING, [52](#)
  - TIMELIMIT\_BAB, [52](#)
- CbcCurrentCutoff
  - CbcModel, [272](#)
- CbcCurrentMinimizationObjectiveValue
  - CbcModel, [273](#)
- CbcCurrentObjectiveValue
  - CbcModel, [273](#)
- CbcCutoffIncrement
  - CbcModel, [272](#)
- CbcEventHandler
  - addCuts, [101](#)
  - afterHeuristic, [101](#)
  - beforeSolution1, [101](#)
  - beforeSolution2, [101](#)
  - endSearch, [101](#)
  - heuristicPass, [101](#)
  - heuristicSolution, [101](#)
  - killSolution, [101](#)
  - noAction, [101](#)
  - node, [101](#)
  - restart, [101](#)
  - restartRoot, [101](#)
  - smallBranchAndBound, [101](#)
  - solution, [101](#)
  - stop, [101](#)

- takeAction, [101](#)
- treeStatus, [101](#)
- CbcFathomDiscipline
  - CbcModel, [272](#)
- CbcGenCtlBlk
  - BACAbandon, [128](#)
  - BACFinish, [128](#)
  - BACInvalid, [128](#)
  - BACNotRun, [128](#)
  - BACStop, [128](#)
  - BACUser, [128](#)
  - BACmFinish, [129](#)
  - BACmGap, [129](#)
  - BACmInfeas, [129](#)
  - BACmInvalid, [129](#)
  - BACmNodeLimit, [129](#)
  - BACmOther, [129](#)
  - BACmSolnLimit, [129](#)
  - BACmTimeLimit, [129](#)
  - BACmUbnd, [129](#)
  - BACmUser, [129](#)
  - BACwBAC, [129](#)
  - BACwBareRoot, [129](#)
  - BACwIPP, [129](#)
  - BACwIPPRelax, [129](#)
  - BACwInvalid, [129](#)
  - BACwNotStarted, [129](#)
  - BPCost, [127](#)
  - BPExt, [128](#)
  - BPOff, [127](#)
  - BPOrder, [128](#)
  - CGForceBut, [127](#)
  - CGForceOn, [127](#)
  - CGIfMove, [127](#)
  - CGMarker, [127](#)
  - CGOff, [127](#)
  - CGOn, [127](#)
  - CGRoot, [127](#)
  - IPPEqual, [126](#)
  - IPPEqualAll, [127](#)
  - IPPOff, [126](#)
  - IPPOn, [126](#)
  - IPPSOS, [126](#)
  - IPPSave, [126](#)
  - IPPStrategy, [127](#)
  - IPPTrySOS, [127](#)
- CbcGenMessages.hpp
  - CBCGEN\_CONFUSION, [522](#)
  - CBCGEN\_DUMMY\_END, [522](#)
  - CBCGEN\_NEW\_SOLVER, [521](#)
  - CBCGEN\_TEST\_MSG, [521](#)
- CbcGenParam
  - BAB, [141](#)
  - CBCGEN\_FIRSTPARAM, [140](#)
  - CBCGEN\_LASTPARAM, [142](#)
  - CLEARCUTS, [141](#)
  - CLIQUECUTS, [141](#)
  - COMBINE, [141](#)
  - COSTSTRATEGY, [141](#)
  - CPP, [141](#)
  - CUTDEPTH, [141](#)
  - CUTSTRATEGY, [141](#)
  - DEBUG, [141](#)
  - DIRECTORY, [141](#)
  - DJFIX, [141](#)
  - DUMMY, [141](#)
  - ERRORSALLOWED, [141](#)
  - EXIT, [141](#)
  - EXPORT, [141](#)
  - FLOWCUTS, [141](#)
  - FPUMP, [141](#)
  - FPUMPITS, [141](#)
  - FULLGENERALQUERY, [141](#)
  - GENERALQUERY, [140](#)
  - GOMORYCUTS, [141](#)
  - GREEDY, [141](#)
  - HELP, [141](#)
  - HEURISTICSTRATEGY, [141](#)
  - IMPORT, [141](#)
  - INTPRINT, [141](#)
  - KNAPSACKCUTS, [141](#)
  - LOCALTREE, [141](#)
  - LOGLEVEL, [141](#)
  - MESSAGES, [141](#)
  - MIPLIB, [141](#)
  - MIXEDCUTS, [141](#)
  - ODDHOLECUTS, [141](#)
  - OUTDUPROWS, [141](#)
  - OUTPUTFORMAT, [141](#)
  - PREPROCESS, [141](#)
  - PRINTMASK, [141](#)
  - PRINTOPTIONS, [141](#)
  - PRINTVERSION, [141](#)
  - PRIORITYIN, [141](#)
  - PROBINGCUTS, [141](#)
  - REDSPLITCUTS, [141](#)
  - ROUNDING, [141](#)
  - SHOWUNIMP, [142](#)
  - SOLUTION, [142](#)
  - SOLVECONTINUOUS, [142](#)
  - SOLVER, [142](#)
  - SOS, [142](#)
  - STDIN, [142](#)
  - STRENGTHEN, [142](#)
  - TIGHTENFACTOR, [142](#)
  - TWOMIRCUTS, [142](#)
  - UNITTEST, [142](#)
  - USERCBC, [142](#)

- USESOLUTION, [142](#)
- VERBOSE, [142](#)
- CbcHeuristicFractionGap
  - CbcModel, [273](#)
- CbcHeuristicGap
  - CbcModel, [273](#)
- CbcInfeasibilityWeight
  - CbcModel, [272](#)
- CbcIntegerTolerance
  - CbcModel, [272](#)
- CbcLargestChange
  - CbcModel, [273](#)
- CbcLastDbIParam
  - CbcModel, [273](#)
- CbcLastIntParam
  - CbcModel, [272](#)
- CbcMaxNumNode
  - CbcModel, [272](#)
- CbcMaxNumSol
  - CbcModel, [272](#)
- CbcMaximumSeconds
  - CbcModel, [272](#)
- CbcMessage.hpp
  - CBC\_BRANCH, [530](#)
  - CBC\_CUTOFF\_WARNING1, [530](#)
  - CBC\_CUTS\_STATS, [530](#)
  - CBC\_DUMMY\_END, [530](#)
  - CBC\_END, [529](#)
  - CBC\_END\_GOOD, [529](#)
  - CBC\_END\_SOLUTION, [529](#)
  - CBC\_END\_SUB, [530](#)
  - CBC\_EVENT, [529](#)
  - CBC\_FATHOM\_CHANGE, [530](#)
  - CBC\_FPUMP1, [530](#)
  - CBC\_FPUMP2, [530](#)
  - CBC\_GAP, [530](#)
  - CBC\_GENERAL, [530](#)
  - CBC\_GENERATOR, [530](#)
  - CBC\_HEURISTICS\_OFF, [530](#)
  - CBC\_INFEAS, [529](#)
  - CBC\_INTEGERINCREMENT, [530](#)
  - CBC\_ITERATE\_STRONG, [530](#)
  - CBC\_MAXITERS, [529](#)
  - CBC\_MAXNODES, [529](#)
  - CBC\_MAXSOLS, [529](#)
  - CBC\_MAXTIME, [529](#)
  - CBC\_NOINT, [530](#)
  - CBC\_NOTFEAS1, [530](#)
  - CBC\_NOTFEAS2, [530](#)
  - CBC\_NOTFEAS3, [530](#)
  - CBC\_OTHER\_STATS, [530](#)
  - CBC\_OTHER\_STATS2, [530](#)
  - CBC\_PRIORITY, [530](#)
  - CBC\_RELAXED1, [530](#)
  - CBC\_RELAXED2, [530](#)
  - CBC\_RESTART, [530](#)
  - CBC\_ROOT, [530](#)
  - CBC\_ROOT\_DETAIL, [530](#)
  - CBC\_ROUNDING, [530](#)
  - CBC\_SOLINDIVIDUAL, [530](#)
  - CBC\_SOLUTION, [529](#)
  - CBC\_SOLUTION2, [529](#)
  - CBC\_START\_SUB, [530](#)
  - CBC\_STATUS, [530](#)
  - CBC\_STATUS2, [530](#)
  - CBC\_STATUS3, [530](#)
  - CBC\_STRONG, [529](#)
  - CBC\_STRONG\_STATS, [530](#)
  - CBC\_STRONGSOL, [530](#)
  - CBC\_THREAD\_STATS, [530](#)
  - CBC\_TREE\_SOL, [530](#)
  - CBC\_UNBOUNDED, [530](#)
  - CBC\_VUB\_END, [530](#)
  - CBC\_VUB\_PASS, [530](#)
  - CBC\_WARNING\_STRONG, [530](#)
- CbcModel
  - CbcAllowableFractionGap, [272](#)
  - CbcAllowableGap, [272](#)
  - CbcCurrentCutoff, [272](#)
  - CbcCurrentMinimizationObjectiveValue, [273](#)
  - CbcCurrentObjectiveValue, [273](#)
  - CbcCutoffIncrement, [272](#)
  - CbcFathomDiscipline, [272](#)
  - CbcHeuristicFractionGap, [273](#)
  - CbcHeuristicGap, [273](#)
  - CbcInfeasibilityWeight, [272](#)
  - CbcIntegerTolerance, [272](#)
  - CbcLargestChange, [273](#)
  - CbcLastDbIParam, [273](#)
  - CbcLastIntParam, [272](#)
  - CbcMaxNumNode, [272](#)
  - CbcMaxNumSol, [272](#)
  - CbcMaximumSeconds, [272](#)
  - CbcNumberBranches, [272](#)
  - CbcOptimizationDirection, [273](#)
  - CbcPrinting, [272](#)
  - CbcSmallChange, [273](#)
  - CbcSmallestChange, [273](#)
  - CbcStartSeconds, [273](#)
  - CbcSumChange, [273](#)
- CbcNumberBranches
  - CbcModel, [272](#)
- CbcOptimizationDirection
  - CbcModel, [273](#)
- CbcOsiParam
  - ALGORITHM, [343](#)
  - ALLSLACK, [343](#)
  - AUTOSCALE, [343](#)

- BARRIER, [343](#)
- BARRIERSCALE, [343](#)
- BASISIN, [343](#)
- BASISOUT, [343](#)
- BIASLU, [343](#)
- CBCOSI\_FIRSTPARAM, [343](#)
- CBCOSI\_LASTPARAM, [344](#)
- CHOLESKY, [343](#)
- CRASH, [343](#)
- CROSSOVER, [343](#)
- DUALBOUND, [343](#)
- DUALPIVOT, [343](#)
- DUALSIMPLEX, [343](#)
- DUALTOLERANCE, [343](#)
- FAKEBOUND, [343](#)
- GAMMA, [343](#)
- IDIOT, [343](#)
- KEEPNAMES, [343](#)
- KKT, [343](#)
- MAXHOTITS, [343](#)
- MAXITERATION, [343](#)
- NETLIB\_BARRIER, [343](#)
- NETLIB\_DUAL, [343](#)
- NETLIB\_PRIMAL, [343](#)
- NETWORK, [343](#)
- OBJSCALE, [343](#)
- PERTURBATION, [343](#)
- PERTVALUE, [344](#)
- PFI, [344](#)
- PLUSMINUS, [344](#)
- PRESOLVE, [344](#)
- PRESOLVEOPTIONS, [344](#)
- PRESOLVEPASS, [344](#)
- PRIMALPIVOT, [344](#)
- PRIMALSIMPLEX, [344](#)
- PRIMALTOLERANCE, [344](#)
- REALLY\_SCALE, [344](#)
- RESTORE, [344](#)
- REVERSE, [344](#)
- RHSSCALE, [344](#)
- SAVE, [344](#)
- SCALING, [344](#)
- SLPVALUE, [344](#)
- SOLVERLOGLEVEL, [344](#)
- SPARSEFACTOR, [344](#)
- SPECIALOPTIONS, [344](#)
- SPRINT, [344](#)
- TIGHTEN, [344](#)
- CbcParam.hpp
  - CBC\_PARAM\_ACTION\_BAB, [538](#)
  - CBC\_PARAM\_ACTION\_MIPLIB, [538](#)
  - CBC\_PARAM\_DBL\_ALLOWABLEGAP, [536](#)
  - CBC\_PARAM\_DBL\_CUTOFF, [536](#)
  - CBC\_PARAM\_DBL\_DJFIX, [536](#)
  - CBC\_PARAM\_DBL\_GAPRATIO, [536](#)
  - CBC\_PARAM\_DBL\_INCREMENT, [536](#)
  - CBC\_PARAM\_DBL\_INFEASIBILITYWEIGHT, [536](#)
  - CBC\_PARAM\_DBL\_INTEGERTOLERANCE, [536](#)
  - CBC\_PARAM\_DBL\_TIGHTENFACTOR, [536](#)
  - CBC\_PARAM\_FULLGENERALQUERY, [536](#)
  - CBC\_PARAM\_GENERALQUERY, [536](#)
  - CBC\_PARAM\_INT\_MAXNODES, [536](#)
  - CBC\_PARAM\_INT\_STRONGBRANCHING, [536](#)
  - CBC\_PARAM\_NOTUSED\_ADDCUTSSTRATEGY, [537](#)
  - CBC\_PARAM\_NOTUSED\_CBCSTUFF, [538](#)
  - CBC\_PARAM\_NOTUSED\_INVALID, [538](#)
  - CBC\_PARAM\_NOTUSED\_ODDHOLECUTS, [537](#)
  - CBC\_PARAM\_NOTUSED\_OSLSTUFF, [538](#)
  - CBC\_PARAM\_STR\_BRANCHSTRATEGY, [537](#)
  - CBC\_PARAM\_STR\_CLIQUECUTS, [537](#)
  - CBC\_PARAM\_STR\_COSTSTRATEGY, [537](#)
  - CBC\_PARAM\_STR\_FLOWCUTS, [537](#)
  - CBC\_PARAM\_STR\_GOMORYCUTS, [537](#)
  - CBC\_PARAM\_STR\_KNAPSACKCUTS, [537](#)
  - CBC\_PARAM\_STR\_MIXEDCUTS, [537](#)
  - CBC\_PARAM\_STR\_NODESTRATEGY, [537](#)
  - CBC\_PARAM\_STR\_PREPROCESS, [537](#)
  - CBC\_PARAM\_STR\_PROBINGCUTS, [537](#)
  - CBC\_PARAM\_STR\_ROUNDING, [537](#)
  - CBC\_PARAM\_STR\_SOLVER, [537](#)
  - CBC\_PARAM\_STR\_TWOMIRCUTS, [537](#)
  - CLP\_PARAM\_ACTION\_ALLSLACK, [538](#)
  - CLP\_PARAM\_ACTION\_BARRIER, [538](#)
  - CLP\_PARAM\_ACTION\_BASISIN, [538](#)
  - CLP\_PARAM\_ACTION\_BASISOUT, [538](#)
  - CLP\_PARAM\_ACTION\_CLEARCUTS, [538](#)
  - CLP\_PARAM\_ACTION\_DIRECTORY, [537](#)
  - CLP\_PARAM\_ACTION\_DUALSIMPLEX, [537](#)
  - CLP\_PARAM\_ACTION\_EXIT, [537](#)
  - CLP\_PARAM\_ACTION\_EXPORT, [537](#)
  - CLP\_PARAM\_ACTION\_FAKEBOUND, [537](#)
  - CLP\_PARAM\_ACTION\_HELP, [538](#)
  - CLP\_PARAM\_ACTION\_IMPORT, [537](#)
  - CLP\_PARAM\_ACTION\_MAXIMIZE, [537](#)
  - CLP\_PARAM\_ACTION\_MINIMIZE, [537](#)
  - CLP\_PARAM\_ACTION\_NETLIB\_BARRIER, [538](#)
  - CLP\_PARAM\_ACTION\_NETLIB\_DUAL, [537](#)
  - CLP\_PARAM\_ACTION\_NETLIB\_PRIMAL, [537](#)
  - CLP\_PARAM\_ACTION\_NETWORK, [538](#)
  - CLP\_PARAM\_ACTION\_PLUSMINUS, [538](#)
  - CLP\_PARAM\_ACTION\_PRIMALSIMPLEX, [537](#)
  - CLP\_PARAM\_ACTION\_REALLY\_SCALE, [538](#)
  - CLP\_PARAM\_ACTION\_RESTORE, [537](#)
  - CLP\_PARAM\_ACTION\_REVERSE, [538](#)
  - CLP\_PARAM\_ACTION\_SAVE, [537](#)
  - CLP\_PARAM\_ACTION\_SOLUTION, [537](#)
  - CLP\_PARAM\_ACTION\_SOLVECONTINUOUS, [538](#)

- CLP\_PARAM\_ACTION\_STDIN, [537](#)
- CLP\_PARAM\_ACTION\_TIGHTEN, [537](#)
- CLP\_PARAM\_ACTION\_UNITTEST, [537](#)
- CLP\_PARAM\_DBL\_DUALBOUND, [536](#)
- CLP\_PARAM\_DBL\_DUALTOLERANCE, [536](#)
- CLP\_PARAM\_DBL\_OBJSCALE, [536](#)
- CLP\_PARAM\_DBL\_PRIMALTOLERANCE, [536](#)
- CLP\_PARAM\_DBL\_PRIMALWEIGHT, [536](#)
- CLP\_PARAM\_DBL\_RHSSCALE, [536](#)
- CLP\_PARAM\_DBL\_TIMELIMIT, [536](#)
- CLP\_PARAM\_INT\_IDIOT, [536](#)
- CLP\_PARAM\_INT\_LOGLEVEL, [536](#)
- CLP\_PARAM\_INT\_MAXFACTOR, [536](#)
- CLP\_PARAM\_INT\_MAXITERATION, [536](#)
- CLP\_PARAM\_INT\_OUTPUTFORMAT, [536](#)
- CLP\_PARAM\_INT\_PERTVALUE, [536](#)
- CLP\_PARAM\_INT\_PREOLVEOPTIONS, [536](#)
- CLP\_PARAM\_INT\_PREOLVEPASS, [536](#)
- CLP\_PARAM\_INT\_PRINTOPTIONS, [536](#)
- CLP\_PARAM\_INT\_SLPVALUE, [536](#)
- CLP\_PARAM\_INT\_SOLVERLOGLEVEL, [536](#)
- CLP\_PARAM\_INT\_SPECIALOPTIONS, [536](#)
- CLP\_PARAM\_INT\_SPRINT, [536](#)
- CLP\_PARAM\_NOTUSED\_ALGORITHM, [537](#)
- CLP\_PARAM\_STR\_AUTOSCALE, [537](#)
- CLP\_PARAM\_STR\_BARRIERSCALE, [537](#)
- CLP\_PARAM\_STR\_BIASLU, [536](#)
- CLP\_PARAM\_STR\_CHOLESKY, [537](#)
- CLP\_PARAM\_STR\_CRASH, [536](#)
- CLP\_PARAM\_STR\_CROSSOVER, [537](#)
- CLP\_PARAM\_STR\_DIRECTION, [536](#)
- CLP\_PARAM\_STR\_DUALPIVOT, [536](#)
- CLP\_PARAM\_STR\_ERRORSALLOWED, [536](#)
- CLP\_PARAM\_STR\_GAMMA, [537](#)
- CLP\_PARAM\_STR\_KEEPNAMES, [536](#)
- CLP\_PARAM\_STR\_KKT, [537](#)
- CLP\_PARAM\_STR\_MESSAGES, [537](#)
- CLP\_PARAM\_STR\_PERTURBATION, [537](#)
- CLP\_PARAM\_STR\_PFI, [537](#)
- CLP\_PARAM\_STR\_PREOLVE, [536](#)
- CLP\_PARAM\_STR\_PRIMALPIVOT, [536](#)
- CLP\_PARAM\_STR\_SCALING, [536](#)
- CLP\_PARAM\_STR\_SPARSEFACTOR, [536](#)
- CLP\_VERSION\_NOTUSED\_PRINTVERSION, [538](#)
- CbcPrinting
  - CbcModel, [272](#)
- CbcRangeDisjoint
  - CbcBranchBase.hpp, [510](#)
- CbcRangeOverlap
  - CbcBranchBase.hpp, [510](#)
- CbcRangeSame
  - CbcBranchBase.hpp, [510](#)
- CbcRangeSubset
  - CbcBranchBase.hpp, [510](#)
- CbcRangeSuperset
  - CbcBranchBase.hpp, [510](#)
- CbcSmallChange
  - CbcModel, [273](#)
- CbcSmallestChange
  - CbcModel, [273](#)
- CbcStartSeconds
  - CbcModel, [273](#)
- CbcSumChange
  - CbcModel, [273](#)
- Cbc\_C\_Interface.h
  - COINLINKAGE, [506](#)
  - Cbc\_addColumns, [505](#)
  - Cbc\_addRows, [505](#)
  - Cbc\_addSOS\_Dense, [506](#)
  - Cbc\_addSOS\_Sparse, [506](#)
  - Cbc\_copyNames, [506](#)
  - Cbc\_getVersion, [505](#)
  - Cbc\_loadProblem, [505](#)
  - Cbc\_registerCallBack, [506](#)
- Cbc\_addColumns
  - Cbc\_C\_Interface.h, [505](#)
- Cbc\_addRows
  - Cbc\_C\_Interface.h, [505](#)
- Cbc\_addSOS\_Dense
  - Cbc\_C\_Interface.h, [506](#)
- Cbc\_addSOS\_Sparse
  - Cbc\_C\_Interface.h, [506](#)
- Cbc\_ampl.h
  - ampl\_obj\_prec, [504](#)
  - freeArgs, [504](#)
  - freeArrays1, [504](#)
  - freeArrays2, [504](#)
  - readAmpl, [504](#)
  - writeAmpl, [504](#)
- Cbc\_copyNames
  - Cbc\_C\_Interface.h, [506](#)
- Cbc\_getVersion
  - Cbc\_C\_Interface.h, [505](#)
- Cbc\_loadProblem
  - Cbc\_C\_Interface.h, [505](#)
- Cbc\_registerCallBack
  - Cbc\_C\_Interface.h, [506](#)
- CbcAction
  - CbcEventHandler, [101](#)
- CbcBaseModel, [27](#)
  - ~CbcBaseModel, [28](#)
  - CbcBaseModel, [27](#)
  - CbcBaseModel, [27](#)
- CbcBranchAllDifferent, [28](#)
  - ~CbcBranchAllDifferent, [29](#)
  - CbcBranchAllDifferent, [29](#)
  - CbcBranchAllDifferent, [29](#)
  - clone, [29](#)



- createCbcBranch, 29
- infeasibility, 29
- numberInSet\_, 29
- operator=, 29
- which\_, 29
- CbcBranchBase.hpp
  - CbcCompareRanges, 510
  - CbcRangeCompare, 510
- CbcBranchCut, 30
  - ~CbcBranchCut, 31
  - boundBranch, 31
  - CbcBranchCut, 31
  - CbcBranchCut, 31
  - clone, 31
  - createCbcBranch, 31
  - feasibleRegion, 31
  - infeasibility, 31
  - notPreferredNewFeasible, 32
  - operator=, 31
  - preferredNewFeasible, 32
  - resetBounds, 32
- CbcBranchDecision, 32
  - ~CbcBranchDecision, 34
  - bestBranch, 34
  - betterBranch, 34
  - CbcBranchDecision, 34
  - cbcModel, 35
  - CbcBranchDecision, 34
  - chooseMethod, 35
  - chooseMethod\_, 36
  - clone, 34
  - generateCpp, 35
  - getBestCriterion, 35
  - initialize, 34
  - model\_, 35
  - object\_, 35
  - saveBranchingObject, 34
  - setBestCriterion, 35
  - setChooseMethod, 35
  - updateInformation, 35
  - whichMethod, 34
- CbcBranchDefaultDecision, 36
  - ~CbcBranchDefaultDecision, 37
  - bestBranch, 37
  - betterBranch, 37
  - CbcBranchDefaultDecision, 37
  - CbcBranchDefaultDecision, 37
  - clone, 37
  - getBestCriterion, 37
  - initialize, 37
  - setBestCriterion, 37
- CbcBranchDynamicDecision, 38
  - ~CbcBranchDynamicDecision, 39
  - betterBranch, 39
  - CbcBranchDynamicDecision, 39
  - CbcBranchDynamicDecision, 39
  - clone, 39
  - getBestCriterion, 39
  - initialize, 39
  - saveBranchingObject, 40
  - setBestCriterion, 39
  - updateInformation, 40
  - whichMethod, 40
- CbcBranchObjType
  - CbcBranchingObject.hpp, 512
- CbcBranchToFixLots, 47
  - ~CbcBranchToFixLots, 48
  - alwaysCreate\_, 50
  - canDoHeuristics, 49
  - CbcBranchToFixLots, 48
  - CbcBranchToFixLots, 48
  - clone, 48
  - createCbcBranch, 49
  - depth\_, 50
  - djTolerance\_, 49
  - fractionFixed\_, 49
  - infeasibility, 49
  - mark\_, 49
  - matrixByRow\_, 49
  - numberClean\_, 50
  - operator=, 48
  - redoSequenceEtc, 49
  - shallWe, 49
- CbcBranchingObject, 40
  - ~CbcBranchingObject, 43
  - branch, 44
  - CbcBranchingObject, 43
  - CbcBranchingObject, 43
  - clone, 43
  - compareBranchingObject, 46
  - compareOriginalObject, 46
  - fillStrongInfo, 43
  - fix, 44
  - model, 45
  - model\_, 46
  - object, 45
  - operator=, 43
  - originalCbcObject\_, 46
  - previousBranch, 44
  - print, 45
  - resetNumberBranchesLeft, 44
  - setModel, 45
  - setNumberBranches, 44
  - setOriginalObject, 45
  - tighten, 44
  - type, 46
  - variable, 45
  - variable\_, 46



- way, 45
- way\_, 46
- CbcBranchingObject.hpp
  - CbcBranchObjType, 512
- CbcCbcParam, 50
  - ~CbcCbcParam, 53
  - CbcCbcParam, 52, 53
  - CbcCbcParamCode, 51
  - CbcCbcParam, 52, 53
  - clone, 53
  - obj, 53
  - operator=, 53
  - paramCode, 53
  - setObj, 53
  - setParamCode, 53
- CbcCbcParamCode
  - CbcCbcParam, 51
- CbcCbcParamUtils, 20
  - addCbcCbcParams, 20
  - loadCbcParamObj, 20
  - pushCbcCbcDbl, 21
  - pushCbcCbcInt, 21
  - setCbcModelDefaults, 20
- CbcClique, 54
  - ~CbcClique, 56
  - CbcClique, 56
  - CbcClique, 56
  - cliqueType, 57
  - cliqueType\_, 58
  - clone, 56
  - createCbcBranch, 56
  - feasibleRegion, 56
  - infeasibility, 56
  - members, 57
  - members\_, 57
  - numberMembers, 56
  - numberMembers\_, 57
  - numberNonSOSMembers, 56
  - numberNonSOSMembers\_, 57
  - operator=, 56
  - redoSequenceEtc, 57
  - slack\_, 58
  - type, 57
  - type\_, 57
- CbcCliqueBranchingObject, 58
  - ~CbcCliqueBranchingObject, 59
  - branch, 59
  - CbcCliqueBranchingObject, 59
  - CbcCliqueBranchingObject, 59
  - clone, 59
  - compareBranchingObject, 60
  - compareOriginalObject, 60
  - operator=, 59
  - print, 60
  - type, 60
- CbcCompare, 60
  - ~CbcCompare, 61
  - alternateTest, 61
  - CbcCompare, 61
  - CbcCompare, 61
  - compareNodes, 61
  - comparisonObject, 61
  - operator(), 61
  - test\_, 61
- CbcCompareBase, 61
  - ~CbcCompareBase, 62
  - alternateTest, 64
  - CbcCompareBase, 62, 63
  - CbcCompareBase, 62, 63
  - clone, 63
  - equalityTest, 64
  - every1000Nodes, 63
  - fullScan, 63
  - generateCpp, 63
  - newSolution, 63
  - operator(), 64
  - operator=, 63
  - sayThreaded, 64
  - test, 64
  - test\_, 64
  - threaded\_, 64
- CbcCompareDefault, 64
  - ~CbcCompareDefault, 66
  - afterNodeNumber\_, 69
  - bestPossible\_, 68
  - breadthDepth\_, 68
  - CbcCompareDefault, 66
  - CbcCompareDefault, 66
  - cleanDive, 68
  - clone, 66
  - cutoff\_, 68
  - every1000Nodes, 67
  - generateCpp, 67
  - getBestPossible, 67
  - getCutoff, 67
  - getWeight, 67
  - newSolution, 67
  - numberSolutions\_, 68
  - operator=, 66
  - saveWeight\_, 68
  - setBestPossible, 67
  - setBreadthDepth, 67
  - setCutoff, 67
  - setWeight, 67
  - setupForDiving\_, 69
  - startDive, 68
  - startNodeNumber\_, 68
  - test, 67

- treeSize\_, 68
- weight\_, 68
- CbcCompareDepth, 69
  - ~CbcCompareDepth, 70
  - CbcCompareDepth, 70
  - CbcCompareDepth, 70
  - clone, 70
  - generateCpp, 70
  - operator=, 70
  - test, 70
- CbcCompareEstimate, 70
  - ~CbcCompareEstimate, 71
  - CbcCompareEstimate, 71
  - CbcCompareEstimate, 71
  - clone, 71
  - generateCpp, 71
  - operator=, 71
  - test, 71
- CbcCompareObjective, 71
  - ~CbcCompareObjective, 72
  - CbcCompareObjective, 72
  - CbcCompareObjective, 72
  - clone, 72
  - generateCpp, 72
  - operator=, 72
  - test, 72
- CbcCompareRanges
  - CbcBranchBase.hpp, 510
- CbcConsequence, 73
  - ~CbcConsequence, 73
  - applyToSolver, 74
  - CbcConsequence, 73
  - CbcConsequence, 73
  - clone, 73
  - operator=, 73
- CbcCountRowCut, 74
  - ~CbcCountRowCut, 75
  - canDropCut, 76
  - CbcCountRowCut, 75
  - CbcCountRowCut, 75
  - decrement, 75
  - increment, 75
  - numberPointingToThis, 76
  - setInfo, 76
  - whichCutGenerator, 76
- CbcCutBranchingObject, 76
  - ~CbcCutBranchingObject, 78
  - boundBranch, 78
  - branch, 78
  - canFix\_, 79
  - CbcCutBranchingObject, 77
  - CbcCutBranchingObject, 77
  - clone, 78
  - compareBranchingObject, 78
  - compareOriginalObject, 78
  - down\_, 79
  - operator=, 78
  - print, 78
  - type, 78
  - up\_, 79
- CbcCutGenerator, 79
  - ~CbcCutGenerator, 83
  - addStatistics, 89
  - atSolution, 85
  - CbcCutGenerator, 82
  - CbcCutGenerator, 82
  - cutGeneratorName, 83
  - generateCuts, 83
  - generateTuning, 83
  - generator, 86
  - globalCuts, 89
  - globalCutsAtRoot, 89
  - howOften, 83
  - howOftenInSub, 84
  - inaccuracy, 84
  - incrementNumberColumnCuts, 87
  - incrementNumberCutsActive, 87
  - incrementNumberCutsInTotal, 86
  - incrementNumberElementsInTotal, 86
  - incrementNumberTimesEntered, 86
  - incrementTimeInCutGenerator, 85
  - ineffectualCuts, 88
  - maximumTries, 84
  - mustCallAgain, 87
  - needsOptimalBasis, 87
  - normal, 85
  - numberActiveCutsAtRoot, 89
  - numberColumnCuts, 86
  - numberCutsActive, 87
  - numberCutsAtRoot, 88
  - numberCutsInTotal, 86
  - numberElementsInTotal, 86
  - numberShortCutsAtRoot, 89
  - numberTimesEntered, 86
  - operator=, 83
  - refreshModel, 83
  - scaleBackStatistics, 89
  - setAtSolution, 85
  - setGlobalCuts, 89
  - setGlobalCutsAtRoot, 89
  - setHowOften, 83
  - setInaccuracy, 84
  - setIneffectualCuts, 88
  - setMaximumTries, 84
  - setModel, 89
  - setMustCallAgain, 87
  - setNeedsOptimalBasis, 87
  - setNormal, 85

- setNumberActiveCutsAtRoot, 89
- setNumberColumnCuts, 86
- setNumberCutsActive, 87
- setNumberCutsAtRoot, 88
- setNumberCutsInTotal, 86
- setNumberElementsInTotal, 86
- setNumberShortCutsAtRoot, 89
- setNumberTimesEntered, 86
- setSwitchOffIfLessThan, 87
- setSwitchedOff, 87
- setTiming, 85
- setWhatDepth, 84
- setWhatDepthInSub, 84
- setWhenInfeasible, 85
- setWhetherCallAtEnd, 88
- setWhetherInMustCallAgainMode, 88
- setWhetherToUse, 88
- switchOffIfLessThan, 87
- switchedOff, 87
- switches, 84
- timeInCutGenerator, 85
- timing, 85
- whatDepth, 84
- whatDepthInSub, 84
- whenInfeasible, 85
- whetherCallAtEnd, 88
- whetherInMustCallAgainMode, 88
- whetherToUse, 88
- CbcCutGenerator.hpp
  - SCANCUTS, 516
  - SCANCUTS\_PROBING, 516
- CbcCutModifier, 90
  - ~CbcCutModifier, 91
  - CbcCutModifier, 90
  - CbcCutModifier, 90
  - clone, 91
  - generateCpp, 91
  - modify, 91
  - operator=, 91
- CbcCutSubsetModifier, 91
  - ~CbcCutSubsetModifier, 92
  - CbcCutSubsetModifier, 92
  - CbcCutSubsetModifier, 92
  - clone, 92
  - firstOdd\_, 93
  - generateCpp, 93
  - modify, 93
  - operator=, 92
- CbcDblParam
  - CbcModel, 272
- CbcDummyBranchingObject, 93
  - ~CbcDummyBranchingObject, 94
  - branch, 94
  - CbcDummyBranchingObject, 94
  - CbcDummyBranchingObject, 94
  - clone, 94
  - compareBranchingObject, 95
  - compareOriginalObject, 95
  - operator=, 94
  - print, 95
  - type, 95
- CbcDynamicPseudoCostBranchingObject, 95
  - ~CbcDynamicPseudoCostBranchingObject, 97
  - branch, 98
  - CbcDynamicPseudoCostBranchingObject, 97
  - CbcDynamicPseudoCostBranchingObject, 97
  - changeInGuessed, 98
  - changeInGuessed\_, 99
  - clone, 97
  - fillPart, 98
  - fillStrongInfo, 98
  - object, 98
  - object\_, 99
  - operator=, 97
  - setChangeInGuessed, 98
  - setObject, 98
  - type, 98
- CbcEvent
  - CbcEventHandler, 101
- CbcEventHandler, 99
  - ~CbcEventHandler, 101
  - CbcAction, 101
  - CbcEvent, 101
  - CbcEventHandler, 101
  - CbcEventHandler, 101
  - clone, 102
  - dfltAction\_, 102
  - eaMap\_, 103
  - eaMapPair, 100
  - event, 102
  - getModel, 102
  - model\_, 102
  - operator=, 102
  - setAction, 102
  - setDfltAction, 102
  - setModel, 102
- CbcFathom, 103
  - ~CbcFathom, 104
  - CbcFathom, 104
  - CbcFathom, 104
  - clone, 104
  - fathom, 104
  - model\_, 104
  - possible, 104
  - possible\_, 105
  - resetModel, 104
  - setModel, 104
- CbcFathomDynamicProgramming, 105

- ~CbcFathomDynamicProgramming, 107
- algorithm\_, 110
- back, 108
- back\_, 109
- bitPattern\_, 110
- CbcFathomDynamicProgramming, 107
- CbcFathomDynamicProgramming, 107
- checkPossible, 108
- clone, 107
- coefficients\_, 109
- cost, 108
- cost\_, 108
- fathom, 107
- indices\_, 109
- lookup\_, 109
- maximumSize, 107
- maximumSizeAllowed\_, 109
- numberActive\_, 109
- numberBits\_, 109
- numberNonOne\_, 110
- resetModel, 107
- rhs\_, 109
- setAlgorithm, 108
- setMaximumSize, 108
- setModel, 107
- setTarget, 108
- size\_, 108
- startBit\_, 109
- target, 108
- target\_, 109
- tryColumn, 108
- type\_, 108
- CbcFeasibilityBase, 110
  - ~CbcFeasibilityBase, 110
  - CbcFeasibilityBase, 110, 111
  - CbcFeasibilityBase, 110, 111
  - clone, 111
  - feasible, 111
  - operator=, 111
- CbcFixVariable, 113
  - ~CbcFixVariable, 114
  - applyToSolver, 115
  - CbcFixVariable, 114
  - CbcFixVariable, 114
  - clone, 114
  - newBound\_, 115
  - numberStates\_, 115
  - operator=, 114
  - startLower\_, 115
  - startUpper\_, 115
  - states\_, 115
  - variable\_, 115
- CbcFixingBranchingObject, 111
  - ~CbcFixingBranchingObject, 112
  - branch, 112
  - CbcFixingBranchingObject, 112
  - CbcFixingBranchingObject, 112
  - clone, 112
  - compareBranchingObject, 113
  - compareOriginalObject, 113
  - operator=, 112
  - print, 112
  - type, 112
- CbcFollowOn, 115
  - ~CbcFollowOn, 117
  - CbcFollowOn, 117
  - CbcFollowOn, 117
  - clone, 117
  - createCbcBranch, 117
  - feasibleRegion, 117
  - gutsOfFollowOn, 117
  - infeasibility, 117
  - matrix\_, 117
  - matrixByRow\_, 117
  - operator=, 117
  - rhs\_, 117
- CbcFullNodeInfo, 118
  - ~CbcFullNodeInfo, 120
  - applyBounds, 120
  - applyToModel, 120
  - basis\_, 121
  - buildRowBasis, 120
  - CbcFullNodeInfo, 119
  - CbcFullNodeInfo, 119
  - clone, 120
  - lower, 120
  - lower\_, 121
  - mutableLower, 120
  - mutableUpper, 121
  - numberIntegers\_, 121
  - setColLower, 120
  - setColUpper, 121
  - upper, 120
  - upper\_, 121
- CbcGenCtlBlk, 121
  - ~CbcGenCtlBlk, 129
  - allowImportErrors\_, 135
  - BACMajor, 128
  - BACMinor, 128
  - BACWhere, 129
  - BPCControl, 127
  - bab\_, 137
  - CGControl, 127
  - CbcGenCtlBlk, 129
  - CbcGenParamUtils::addCbcGenParams, 134
  - cbcParams\_, 135
  - CbcGenCtlBlk, 129
  - chooseStrong\_, 137

- debugCreate\_, 136
- debugFile\_, 136
- debugSol\_, 136
- defaultSettings\_, 136
- dfltDirectory\_, 134
- dfltSolver\_, 136
- djFix\_, 137
- genParams\_, 135
- getClique, 130
- getCombine, 132
- getCutDepth, 130
- getFPump, 131
- getFlow, 130
- getGomory, 131
- getGreedyCover, 132
- getGreedyEquality, 132
- getIPPAAction, 130
- getKnapsack, 131
- getMir, 131
- getProbing, 130
- getRedSplit, 131
- getRounding, 132
- getTreeLocal, 132
- getTwomir, 131
- goodModel\_, 136
- IPPControl, 126
- lastMpsIn\_, 134
- lastSolnOut\_, 135
- logLevel, 134
- message, 133
- messageHandler, 134
- model\_, 136
- osiParams\_, 135
- paramVec\_, 135
- paramsProcessed\_, 135
- passInMessageHandler, 133
- printBaBStatus, 133
- printMask\_, 135
- printMode\_, 135
- printOpt\_, 134
- priorityAction\_, 137
- setBaBStatus, 133
- setByUser\_, 136
- setCliqueAction, 130
- setCombineAction, 132
- setCutDepth, 130
- setFPumpAction, 131
- setFlowAction, 130
- setGomoryAction, 131
- setGreedyCoverAction, 132
- setGreedyEqualityAction, 132
- setIPPAAction, 130
- setKnapsackAction, 131
- setLogLevel, 134
- setMessages, 134
- setMirAction, 131
- setProbingAction, 130
- setRedSplitAction, 131
- setRoundingAction, 132
- setTreeLocalAction, 133
- setTwomirAction, 131
- totalTime\_, 136
- translateMajor, 133
- translateMinor, 133
- verbose\_, 135
- version\_, 134
- CbcGenCtlBlk::babState\_struct, 26
  - answerSolver\_, 27
  - haveAnswer\_, 27
  - majorStatus\_, 27
  - minorStatus\_, 27
  - where\_, 27
- CbcGenCtlBlk::cbcParamsInfo\_struct, 353
  - first\_, 353
  - last\_, 353
- CbcGenCtlBlk::chooseStrongCtl\_struct, 433
  - numBeforeTrust\_, 433
  - numStrong\_, 433
  - shadowPriceMode\_, 433
- CbcGenCtlBlk::debugSolInfo\_struct, 439
  - numCols\_, 440
  - values\_, 440
- CbcGenCtlBlk::djFixCtl\_struct, 440
  - action\_, 440
  - threshold\_, 440
- CbcGenCtlBlk::genParamsInfo\_struct, 441
  - first\_, 441
  - last\_, 441
- CbcGenCtlBlk::osiParamsInfo\_struct, 486
  - first\_, 486
  - last\_, 486
- CbcGenMessages.hpp
  - CbcGenMsgCode, 521
- CbcGenMsgCode
  - CbcGenMessages.hpp, 521
- CbcGenParam, 139
  - ~CbcGenParam, 143
  - CbcGenParam, 142, 143
  - CbcGenParamCode, 140
  - CbcGenParam, 142, 143
  - clone, 143
  - obj, 143
  - operator=, 143
  - paramCode, 143
  - setObj, 143
  - setParamCode, 143
- CbcGenParamCode
  - CbcGenParam, 140

- CbcGenParamUtils, 21
  - addCbcGenParams, 21
  - doBaCParam, 21
  - doDebugParam, 21
  - doExitParam, 21
  - doHelpParam, 21
  - doImportParam, 21
  - doNothingParam, 21
  - doPrintMaskParam, 21
  - doSolutionParam, 22
  - doUnimplementedParam, 22
  - doVersionParam, 22
  - loadGenParamObj, 21
  - pushCbcGenCutParam, 22
  - pushCbcGenDblParam, 22
  - pushCbcGenIntParam, 22
  - pushCbcGenKwdParam, 22
  - pushCbcGenStrParam, 22
  - readSolution, 21
  - saveSolution, 21
- CbcGenParamUtils::addCbcGenParams
  - CbcGenCtlBlk, 134
- CbcGeneral, 137
  - ~CbcGeneral, 138
  - CbcGeneral, 138
  - CbcGeneral, 138
  - clone, 138
  - createCbcBranch, 138
  - feasibleRegion, 138
  - infeasibility, 138
  - operator=, 138
  - redoSequenceEtc, 138
- CbcHeuristic, 143
  - ~CbcHeuristic, 148
  - canDealWithOdd, 151
  - CbcHeuristic, 148
  - CbcHeuristic, 148
  - clone, 148
  - cloneBut, 152
  - debugNodes, 152
  - decayFactor\_, 153
  - exitNow, 149
  - feasibilityPumpOptions, 150
  - feasibilityPumpOptions\_, 153
  - fractionSmall, 150
  - fractionSmall\_, 153
  - generateCpp, 150, 151
  - getSeed, 151
  - heuristicName, 151
  - heuristicName\_, 153
  - howOften\_, 153
  - howOftenShallow\_, 154
  - incrementNumberSolutionsFound, 150
  - inputSolution\_, 155
  - lastRunDeep\_, 154
  - minDistanceToRun\_, 154
  - model\_, 152
  - numCouldRun, 152
  - numCouldRun\_, 154
  - numInvocationsInDeep\_, 154
  - numInvocationsInShallow\_, 154
  - numRuns, 152
  - numRuns\_, 154
  - numberNodes, 149
  - numberNodes\_, 153
  - numberNodesDone\_, 155
  - numberSolutionsFound, 150
  - numberSolutionsFound\_, 155
  - operator=, 148
  - printDistanceToNodes, 152
  - randomNumberGenerator\_, 153
  - resetModel, 148
  - runNodes\_, 154
  - setDecayFactor, 151
  - setFeasibilityPumpOptions, 150
  - setFractionSmall, 150
  - setHeuristicName, 151
  - setHowOftenShallow, 152
  - setInputSolution, 151
  - setMinDistanceToRun, 152
  - setModel, 148
  - setModelOnly, 150
  - setNumberNodes, 149
  - setSeed, 151
  - setShallowDepth, 151
  - setSwitches, 149
  - setWhen, 149
  - setWhereFrom, 151
  - shallowDepth\_, 154
  - shouldHeurRun, 152
  - shouldHeurRun\_randomChoice, 152
  - smallBranchAndBound, 150
  - solution, 148
  - solution2, 148
  - switches, 149
  - switches\_, 153
  - validate, 149
  - when, 149
  - when\_, 153
  - whereFrom, 151
  - whereFrom\_, 153
- CbcHeuristicCrossover, 155
  - ~CbcHeuristicCrossover, 156
  - attempts\_, 157
  - CbcHeuristicCrossover, 156
  - CbcHeuristicCrossover, 156
  - clone, 156
  - generateCpp, 156

- numberSolutions\_, 157
- operator=, 156
- random\_, 157
- resetModel, 156
- setModel, 157
- setNumberSolutions, 157
- solution, 157
- useNumber\_, 157
- CbcHeuristicDINS, 157
  - ~CbcHeuristicDINS, 159
  - CbcHeuristicDINS, 159
  - CbcHeuristicDINS, 159
  - clone, 159
  - generateCpp, 159
  - howOften\_, 160
  - localSpace\_, 161
  - maximumKeepSolutions\_, 160
  - numberIntegers\_, 161
  - numberKeptSolutions\_, 160
  - numberSolutions\_, 160
  - numberSuccesses\_, 160
  - numberTries\_, 160
  - operator=, 159
  - resetModel, 159
  - setConstraint, 160
  - setHowOften, 160
  - setMaximumKeep, 160
  - setModel, 159
  - solution, 159
  - solutionFix, 160
  - values\_, 161
- CbcHeuristicDW, 176
  - ~CbcHeuristicDW, 181
  - affinity\_, 188
  - backwardRow\_, 187
  - bestObjective, 182
  - bestObjective\_, 186
  - bestSolution, 182
  - bestSolution\_, 186
  - CbcHeuristicDW, 181
  - CbcHeuristicDW, 181
  - clone, 181
  - columnsInBlock, 185
  - columnsInBlock\_, 188
  - continuousSolution, 183
  - continuousSolution\_, 186
  - DWModel, 182
  - doubleArray\_, 186
  - doubleArrays, 185
  - dwBlock\_, 187
  - dwSolver\_, 186
  - fingerprint\_, 188
  - fixedDj, 183
  - fixedDj\_, 186
  - fullDWEverySoOften\_, 188
  - functionPointer\_, 186
  - generateCpp, 181
  - getCurrentNumberNeeded, 184
  - getCurrentNumberNodes, 184
  - getNumberNeeded, 183
  - getNumberNodes, 184
  - heuristicCallBack, 181
  - howOften\_, 188
  - initialLower, 184
  - initialUpper, 185
  - intArray\_, 186
  - intArrays, 185
  - intsInBlock, 185
  - intsInBlock\_, 188
  - keepContinuous\_, 189
  - lastObjective\_, 186
  - maximumDW\_, 189
  - nNeeded\_, 190
  - nNeededBase\_, 190
  - nNodes\_, 190
  - nNodesBase\_, 190
  - numberBadPasses\_, 190
  - numberBlocks, 182
  - numberBlocks\_, 189
  - numberColumnsDW, 182, 183
  - numberColumnsDW\_, 187
  - numberDW\_, 189
  - numberDWTimes, 183
  - numberDWTimes\_, 189
  - numberMasterColumns\_, 189
  - numberMasterRows\_, 189
  - numberPasses\_, 188
  - objectiveDW, 183
  - objectiveDW\_, 187
  - objectiveValue, 185
  - objectiveValueWhen, 182
  - operator=, 181
  - pass, 185
  - pass\_, 189
  - passInContinuousSolution, 182
  - passInSolution, 182
  - phase, 185
  - phase\_, 189
  - random\_, 187
  - resetModel, 181
  - rowsInBlock\_, 188
  - saveLower\_, 187
  - saveUpper\_, 187
  - setCurrentNumberNeeded, 183
  - setCurrentNumberNodes, 184
  - setHowOften, 184
  - setModel, 182
  - setNumberBadPasses, 183

- setNumberNeeded, 183
- setNumberNodes, 184
- setNumberPasses, 183
- setProposalActions, 182
- setTargetObjective, 184
- sizeFingerPrint\_, 189
- solution, 182
- solveState\_, 190
- solver, 182
- solver\_, 186
- startColumnBlock, 185
- startColumnBlock\_, 188
- startRowBlock\_, 188
- targetObjective\_, 185
- weights\_, 187
- whichColumnBlock, 184
- whichColumnBlock\_, 187
- whichRowBlock, 184
- whichRowBlock\_, 187
- CbcHeuristicDive, 161
  - ~CbcHeuristicDive, 163
  - binVarIndex\_, 166
  - canHeuristicRun, 165
  - CbcHeuristicDive, 163
  - CbcHeuristicDive, 163
  - clone, 163
  - downArray\_, 166
  - downLocks\_, 165
  - fathom, 164
  - fixOtherVariables, 165
  - generateCpp, 163
  - initializeData, 165
  - matrix\_, 165
  - matrixByRow\_, 165
  - maxIterations\_, 166
  - maxSimplexIterations, 164
  - maxSimplexIterations\_, 166
  - maxSimplexIterationsAtRoot\_, 166
  - maxTime\_, 166
  - operator=, 163
  - percentageToFix\_, 166
  - reducedCostFix, 165
  - resetModel, 163
  - selectBinaryVariables, 164
  - selectVariableToBranch, 165
  - setMaxIterations, 164
  - setMaxSimplexIterations, 164
  - setMaxSimplexIterationsAtRoot, 165
  - setMaxTime, 165
  - setModel, 164
  - setPercentageToFix, 164
  - solution, 164
  - upArray\_, 166
  - upLocks\_, 166
  - validate, 164
  - vbRowIndex\_, 166
- CbcHeuristicDiveCoefficient, 166
  - ~CbcHeuristicDiveCoefficient, 167
  - CbcHeuristicDiveCoefficient, 167
  - CbcHeuristicDiveCoefficient, 167
  - clone, 167
  - generateCpp, 168
  - operator=, 167
  - selectVariableToBranch, 168
- CbcHeuristicDiveFractional, 168
  - ~CbcHeuristicDiveFractional, 169
  - CbcHeuristicDiveFractional, 169
  - CbcHeuristicDiveFractional, 169
  - clone, 169
  - generateCpp, 169
  - operator=, 169
  - selectVariableToBranch, 169
- CbcHeuristicDiveGuided, 170
  - ~CbcHeuristicDiveGuided, 170
  - canHeuristicRun, 171
  - CbcHeuristicDiveGuided, 170
  - CbcHeuristicDiveGuided, 170
  - clone, 171
  - generateCpp, 171
  - operator=, 171
  - selectVariableToBranch, 171
- CbcHeuristicDiveLineSearch, 171
  - ~CbcHeuristicDiveLineSearch, 172
  - CbcHeuristicDiveLineSearch, 172
  - CbcHeuristicDiveLineSearch, 172
  - clone, 172
  - generateCpp, 172
  - operator=, 172
  - selectVariableToBranch, 172
- CbcHeuristicDivePseudoCost, 173
  - ~CbcHeuristicDivePseudoCost, 174
  - CbcHeuristicDivePseudoCost, 174
  - CbcHeuristicDivePseudoCost, 174
  - clone, 174
  - fixOtherVariables, 174
  - generateCpp, 174
  - initializeData, 174
  - operator=, 174
  - selectVariableToBranch, 174
- CbcHeuristicDiveVectorLength, 175
  - ~CbcHeuristicDiveVectorLength, 176
  - CbcHeuristicDiveVectorLength, 175
  - CbcHeuristicDiveVectorLength, 175
  - clone, 176
  - generateCpp, 176
  - operator=, 176
  - selectVariableToBranch, 176
- CbcHeuristicDynamic3, 190



- ~CbcHeuristicDynamic3, 191
- canDealWithOdd, 192
- CbcHeuristicDynamic3, 191
- CbcHeuristicDynamic3, 191
- clone, 191
- resetModel, 192
- setModel, 191
- solution, 191
- CbcHeuristicFPump, 192
  - ~CbcHeuristicFPump, 195
  - absoluteIncrement, 196
  - absoluteIncrement\_, 199
  - accumulate, 198
  - accumulate\_, 200
  - artificialCost, 197
  - artificialCost\_, 200
  - CbcHeuristicFPump, 195
  - CbcHeuristicFPump, 195
  - clone, 195
  - defaultRounding, 197
  - defaultRounding\_, 199
  - fakeCutoff, 196
  - fakeCutoff\_, 199
  - fixOnReducedCosts, 198
  - fixOnReducedCosts\_, 200
  - generateCpp, 195
  - initialWeight, 197
  - initialWeight\_, 199
  - iterationRatio, 197
  - iterationRatio\_, 200
  - maximumPasses, 198
  - maximumPasses\_, 200
  - maximumRetries, 198
  - maximumRetries\_, 200
  - maximumTime, 196
  - maximumTime\_, 199
  - operator=, 195
  - reducedCostMultiplier, 199
  - reducedCostMultiplier\_, 200
  - relativeIncrement, 196
  - relativeIncrement\_, 199
  - resetModel, 195
  - roundExpensive\_, 200
  - setAbsoluteIncrement, 196
  - setAccumulate, 198
  - setArtificialCost, 197
  - setDefaultRounding, 196
  - setFakeCutoff, 196
  - setFixOnReducedCosts, 198
  - setInitialWeight, 197
  - setIterationRatio, 197
  - setMaximumPasses, 197
  - setMaximumRetries, 198
  - setMaximumTime, 196
  - setModel, 195
  - setReducedCostMultiplier, 198
  - setRelativeIncrement, 196
  - setWeightFactor, 197
  - solution, 195
  - startTime\_, 199
  - weightFactor, 197
  - weightFactor\_, 199
- CbcHeuristicGreedyCover, 201
  - ~CbcHeuristicGreedyCover, 202
  - algorithm, 203
  - algorithm\_, 203
  - CbcHeuristicGreedyCover, 202
  - CbcHeuristicGreedyCover, 202
  - clone, 202
  - generateCpp, 202
  - gutsOfConstructor, 203
  - matrix\_, 203
  - numberTimes, 203
  - numberTimes\_, 203
  - operator=, 202
  - originalNumberRows\_, 203
  - resetModel, 202
  - setAlgorithm, 203
  - setModel, 202
  - setNumberTimes, 203
  - solution, 202
  - validate, 202
- CbcHeuristicGreedyEquality, 203
  - ~CbcHeuristicGreedyEquality, 205
  - algorithm, 206
  - algorithm\_, 206
  - CbcHeuristicGreedyEquality, 205
  - CbcHeuristicGreedyEquality, 205
  - clone, 205
  - fraction, 206
  - fraction\_, 206
  - generateCpp, 205
  - gutsOfConstructor, 206
  - matrix\_, 206
  - numberTimes, 206
  - numberTimes\_, 206
  - operator=, 205
  - originalNumberRows\_, 206
  - resetModel, 205
  - setAlgorithm, 206
  - setFraction, 206
  - setModel, 205
  - setNumberTimes, 206
  - solution, 205
  - validate, 205
- CbcHeuristicGreedySOS, 207
  - ~CbcHeuristicGreedySOS, 208
  - algorithm, 209

- algorithm\_, 209
- CbcHeuristicGreedySOS, 208
- CbcHeuristicGreedySOS, 208
- clone, 208
- generateCpp, 208
- gutsOfConstructor, 209
- matrix\_, 209
- numberTimes, 209
- numberTimes\_, 209
- operator=, 208
- originalNumberRows\_, 209
- originalRhs\_, 209
- resetModel, 209
- setAlgorithm, 209
- setModel, 208
- setNumberTimes, 209
- solution, 208
- validate, 208
- CbcHeuristicJustOne, 210
  - ~CbcHeuristicJustOne, 211
  - addHeuristic, 212
  - CbcHeuristicJustOne, 211
  - CbcHeuristicJustOne, 211
  - clone, 211
  - generateCpp, 211
  - heuristic\_, 212
  - normalizeProbabilities, 212
  - numberHeuristics\_, 212
  - operator=, 211
  - probabilities\_, 212
  - resetModel, 211
  - selectVariableToBranch, 211
  - setModel, 211
  - solution, 211
  - validate, 212
- CbcHeuristicLocal, 212
  - ~CbcHeuristicLocal, 213
  - CbcHeuristicLocal, 213
  - CbcHeuristicLocal, 213
  - clone, 213
  - generateCpp, 214
  - matrix\_, 214
  - numberSolutions\_, 214
  - operator=, 213
  - resetModel, 214
  - setModel, 214
  - setSearchType, 214
  - solution, 214
  - solutionFix, 214
  - swap\_, 214
  - used, 214
  - used\_, 215
- CbcHeuristicNaive, 215
  - ~CbcHeuristicNaive, 216
- CbcHeuristicNaive, 216
- CbcHeuristicNaive, 216
- clone, 216
- generateCpp, 216
- large\_, 217
- largeValue, 217
- operator=, 216
- resetModel, 216
- setLargeValue, 217
- setModel, 216
- solution, 216
- CbcHeuristicNode, 217
  - ~CbcHeuristicNode, 217
  - avgDistance, 218
  - CbcHeuristicNode, 217
  - CbcHeuristicNode, 217
  - distance, 218
  - minDistance, 218
  - minDistanceIsSmall, 218
- CbcHeuristicNodeList, 218
  - ~CbcHeuristicNodeList, 218
  - append, 218
  - CbcHeuristicNodeList, 218
  - CbcHeuristicNodeList, 218
  - node, 219
  - operator=, 218
  - size, 219
- CbcHeuristicPartial, 219
  - ~CbcHeuristicPartial, 220
  - CbcHeuristicPartial, 220
  - CbcHeuristicPartial, 220
  - clone, 220
  - fixPriority\_, 221
  - generateCpp, 220
  - operator=, 220
  - resetModel, 220
  - setFixPriority, 221
  - setModel, 220
  - shouldHeurRun, 221
  - solution, 220
  - validate, 221
- CbcHeuristicPivotAndFix, 221
  - ~CbcHeuristicPivotAndFix, 222
  - CbcHeuristicPivotAndFix, 222
  - CbcHeuristicPivotAndFix, 222
  - clone, 222
  - generateCpp, 222
  - operator=, 222
  - resetModel, 222
  - setModel, 222
  - solution, 223
- CbcHeuristicProximity, 223
  - ~CbcHeuristicProximity, 224
  - CbcHeuristicProximity, 224

- CbcHeuristicProximity, 224
- clone, 224
- feasibilityPump\_, 225
- generateCpp, 224
- increment\_, 225
- numberSolutions\_, 225
- operator=, 224
- resetModel, 224
- setIncrement, 225
- setModel, 224
- solution, 224
- used, 225
- used\_, 225
- CbcHeuristicRENS, 227
  - ~CbcHeuristicRENS, 228
  - CbcHeuristicRENS, 228
  - CbcHeuristicRENS, 228
  - clone, 228
  - numberTries\_, 229
  - operator=, 228
  - rensType\_, 229
  - resetModel, 228
  - setModel, 228
  - setRensType, 229
  - solution, 229
- CbcHeuristicRINS, 229
  - ~CbcHeuristicRINS, 231
  - CbcHeuristicRINS, 231
  - CbcHeuristicRINS, 231
  - clone, 231
  - generateCpp, 231
  - howOften\_, 232
  - lastNode\_, 232
  - numberSolutions\_, 232
  - numberSuccesses\_, 232
  - numberTries\_, 232
  - operator=, 231
  - resetModel, 231
  - setHowOften, 231
  - setLastNode, 232
  - setModel, 231
  - setSolutionCount, 232
  - solution, 231
  - solutionFix, 231
  - stateOfFixing\_, 232
  - used, 231
  - used\_, 232
- CbcHeuristicRandRound, 225
  - ~CbcHeuristicRandRound, 226
  - CbcHeuristicRandRound, 226
  - CbcHeuristicRandRound, 226
  - clone, 226
  - generateCpp, 227
  - operator=, 226
  - resetModel, 227
  - setModel, 227
  - solution, 227
- CbcHeuristicVND, 233
  - ~CbcHeuristicVND, 234
  - baseSolution, 235
  - baseSolution\_, 236
  - CbcHeuristicVND, 234
  - CbcHeuristicVND, 234
  - clone, 234
  - generateCpp, 234
  - howOften\_, 235
  - k\_, 236
  - kmax\_, 236
  - lastNode\_, 235
  - nDifferent\_, 236
  - numberSolutions\_, 235
  - numberSuccesses\_, 235
  - numberTries\_, 235
  - operator=, 234
  - resetModel, 234
  - setHowOften, 235
  - setModel, 235
  - solution, 235
  - solutionFix, 235
  - stepSize\_, 236
- CbcIdiotBranch, 236
  - ~CbcIdiotBranch, 237
  - buildCut, 238
  - CbcIdiotBranch, 237
  - CbcIdiotBranch, 237
  - clone, 237
  - createCbcBranch, 238
  - feasibleRegion, 238
  - infeasibility, 238
  - initializeForBranching, 238
  - operator=, 238
  - randomNumberGenerator\_, 238
  - savedRandomNumberGenerator\_, 238
- CbcIntParam
  - CbcModel, 272
- CbcIntegerBranchingObject, 239
  - ~CbcIntegerBranchingObject, 240
  - branch, 241
  - CbcIntegerBranchingObject, 240
  - CbcIntegerBranchingObject, 240
  - clone, 241
  - compareBranchingObject, 242
  - down\_, 242
  - downBounds, 241
  - fillPart, 241
  - fix, 241
  - operator=, 241
  - print, 241

- setDownBounds, [241](#)
  - setUpBounds, [242](#)
  - tighten, [241](#)
  - type, [242](#)
  - up\_, [242](#)
  - upBounds, [241](#)
- CbcIntegerPseudoCostBranchingObject, [242](#)
  - ~CbcIntegerPseudoCostBranchingObject, [244](#)
  - branch, [244](#)
  - CbcIntegerPseudoCostBranchingObject, [244](#)
  - CbcIntegerPseudoCostBranchingObject, [244](#)
  - changeInGuessed, [245](#)
  - changeInGuessed\_, [245](#)
  - clone, [244](#)
  - compareBranchingObject, [245](#)
  - operator=, [244](#)
  - setChangeInGuessed, [245](#)
  - type, [245](#)
- CbcLinked.hpp
  - approximateSolution, [528](#)
- CbcLongCliqueBranchingObject, [245](#)
  - ~CbcLongCliqueBranchingObject, [246](#)
  - branch, [247](#)
  - CbcLongCliqueBranchingObject, [246](#)
  - CbcLongCliqueBranchingObject, [246](#)
  - clone, [247](#)
  - compareBranchingObject, [247](#)
  - compareOriginalObject, [247](#)
  - operator=, [247](#)
  - print, [247](#)
  - type, [247](#)
- CbcLotsize, [247](#)
  - ~CbcLotsize, [249](#)
  - bound, [251](#)
  - canDoHeuristics, [251](#)
  - CbcLotsize, [249](#)
  - CbcLotsize, [249](#)
  - clone, [249](#)
  - columnNumber, [250](#)
  - createCbcBranch, [249](#)
  - feasibleRegion, [249](#)
  - findRange, [250](#)
  - floorCeiling, [250](#)
  - infeasibility, [249](#)
  - modelSequence, [250](#)
  - notPreferredNewFeasible, [250](#)
  - numberRanges, [251](#)
  - operator=, [249](#)
  - originalLowerBound, [250](#)
  - originalUpperBound, [251](#)
  - preferredNewFeasible, [250](#)
  - rangeType, [251](#)
  - resetBounds, [250](#)
  - setModelSequence, [250](#)
- CbcLotsizeBranchingObject, [251](#)
  - ~CbcLotsizeBranchingObject, [253](#)
  - branch, [253](#)
  - CbcLotsizeBranchingObject, [252](#), [253](#)
  - CbcLotsizeBranchingObject, [252](#), [253](#)
  - clone, [253](#)
  - compareBranchingObject, [253](#)
  - down\_, [254](#)
  - operator=, [253](#)
  - print, [253](#)
  - type, [253](#)
  - up\_, [254](#)
- CbcMain
  - CbcModel.hpp, [532](#)
- CbcMain0
  - CbcModel.hpp, [532](#)
- CbcMain1
  - CbcModel.hpp, [532](#)
- CbcMessage, [254](#)
  - CbcMessage, [255](#)
  - CbcMessage, [255](#)
- CbcMessage.hpp
  - CBC\_Message, [529](#)
- CbcMipStartIO.hpp
  - computeCompleteSolution, [531](#)
  - readMIPStart, [531](#)
- CbcModel, [255](#)
  - ~CbcModel, [273](#)
  - addCutGenerator, [296](#)
  - addCuts, [306](#)
  - addCuts1, [306](#)
  - addHeuristic, [297](#)
  - AddIntegers, [275](#)
  - addObjects, [276](#)
  - addUpdateInformation, [274](#)
  - addedCuts, [308](#)
  - adjustHeuristics, [307](#)
  - allDynamic, [310](#)
  - analyzeObjective, [275](#)
  - assignSolver, [302](#)
  - bestSolution, [291](#)
  - branchAndBound, [273](#)
  - branchingMethod, [295](#)
  - canStopOnGap, [280](#)
  - CbcDbIParam, [272](#)
  - CbcIntParam, [272](#)
  - CbcModel, [273](#)
  - CbcModel, [273](#)
  - checkModel, [302](#)
  - checkSolution, [289](#)
  - chooseBranch, [306](#)
  - clearContinuousSolver, [303](#)
  - clearNumberGlobalViolations, [293](#)
  - cliquePseudoCosts, [307](#)

clone, 302  
conflictCut, 283  
continuousPriority, 309  
continuousSolution, 288  
continuousSolver, 303  
convertToDynamic, 307  
createContinuousSolver, 303  
currentDepth, 282  
currentNode, 308  
currentNumberCuts, 308  
currentSolution, 289  
cutGenerator, 296  
cutGenerators, 296  
cutModifier, 295  
dealWithEventHandler, 289  
defaultHandler, 299  
deleteObjects, 276  
deleteSavedSolution, 292  
deleteSolutions, 305  
doCutsNow, 281  
doHeuristicsAtRoot, 307  
doOneNode, 274  
fastNodeDepth, 309  
feasibleSolution, 289  
fillPseudoCosts, 307  
findCliques, 274  
findIntegers, 276  
flipModel, 276  
generateCpp, 310  
getAllowableFractionGap, 279  
getAllowableGap, 279  
getAllowablePercentageGap, 279  
getApplicationData, 298  
getBestPossibleObjValue, 290  
getCbcColLower, 288  
getCbcColSolution, 288  
getCbcColUpper, 288  
getCbcReducedCost, 288  
getCbcRowActivity, 288  
getCbcRowLower, 288  
getCbcRowPrice, 288  
getCbcRowUpper, 288  
getColLower, 286  
getColSolution, 290  
getColUpper, 286  
getContinuousInfeasibilities, 292  
getContinuousObjective, 292  
getCurrentMinimizationObjValue, 290  
getCurrentObjValue, 290  
getCurrentPassNumber, 280  
getCurrentSeconds, 278  
getCutoff, 277  
getCutoffIncrement, 280  
getDbIParam, 277  
getEmptyBasis, 306  
getEventHandler, 298  
getExtraNodeCount, 284  
getHeuristicFractionGap, 279  
getHeuristicGap, 279  
getInfeasibilityWeight, 278  
getInfinity, 288  
getIntParam, 277  
getIntegerTolerance, 278  
getIterationCount, 284  
getMIPStart, 310  
getMatrixByCol, 287  
getMatrixByRow, 287  
getMaximumCutPasses, 280  
getMaximumCutPassesAtRoot, 280  
getMaximumNodes, 277  
getMaximumSeconds, 278  
getMaximumSolutions, 277  
getMinimizationObjValue, 290  
getMinimumDrop, 280  
getMultipleRootTries, 300  
getNodeCount, 284  
getNodeCount2, 305  
getNumCols, 285  
getNumElements, 285  
getNumRows, 285  
getNumberHeuristicSolutions, 292  
getNumberThreads, 304  
getObjCoefficients, 287  
getObjSense, 287  
getObjValue, 290  
getPreferredWay, 281  
getPrintingMode, 278  
getRandomSeed, 300  
getReducedCost, 290  
getRightHandSide, 286  
getRowActivity, 290  
getRowLower, 286  
getRowPrice, 290  
getRowRange, 286  
getRowSense, 286  
getRowUpper, 287  
getSolutionCount, 291  
getSolverObjValue, 291  
getStopNumberIterations, 293  
getThreadMode, 304  
globalCuts, 308  
goToDantzig, 302  
gutsOfCopy, 303  
gutsOfDestructor, 303  
gutsOfDestructor2, 303  
haveMultiThreadSupport, 304  
heuristic, 297  
heuristicModel, 293

hotstartPriorities, 307  
hotstartSolution, 307  
howOftenGlobalScan, 283  
incrementExtra, 309  
incrementIterationCount, 284  
incrementNodeCount, 284  
incrementStrongInfo, 309  
incrementSubTreeStopped, 294  
incrementUsed, 289  
initialSolve, 273  
integerPresolve, 275  
integerPresolveThisModel, 275  
integerType, 285, 286  
integerVariable, 285  
isAbandoned, 283  
isBinary, 287  
isContinuous, 287  
isContinuousUnbounded, 283  
isFreeBinary, 287  
isInitialSolveAbandoned, 285  
isInitialSolveProvenDualInfeasible, 285  
isInitialSolveProvenOptimal, 285  
isInitialSolveProvenPrimalInfeasible, 285  
isInteger, 287  
isIntegerNonBinary, 287  
isLocked, 304  
isNodeLimitReached, 283  
isProvenDualInfeasible, 283  
isProvenInfeasible, 283  
isProvenOptimal, 283  
isSecondsLimitReached, 284  
isSolutionLimitReached, 284  
lastHeuristic, 297  
lockThread, 304  
logLevel, 299  
makeGlobalCut, 274  
makeGlobalCuts, 274  
makePartialCut, 274  
masterThread, 304  
maximumNumberIterations, 308  
maximumRows, 293  
maximumSavedSolutions, 291  
maximumSecondsReached, 278  
mergeModels, 305  
messageHandler, 299  
messages, 299  
messagesPointer, 299  
modelOwnsSolver, 302  
modifiableObject, 276  
moreSpecialOptions, 301  
moreSpecialOptions2, 301  
moveInfo, 303  
moveToModel, 305  
mutableStrongInfo, 309  
newLanguage, 299  
nodeComparison, 294  
normalSolver, 300  
numberAnalyzeIterations, 282  
numberBeforeTrust, 281  
numberCutGenerators, 296  
numberExtraIterations, 309  
numberGlobalViolations, 293  
numberHeuristics, 297  
numberIntegers, 285  
numberObjects, 276  
numberPenalties, 282  
numberRowsAtContinuous, 285  
numberSavedSolutions, 291  
numberStoppedSubTrees, 294  
numberStrong, 281  
numberStrongIterations, 308  
object, 276  
objects, 276  
operator=, 302  
originalColumns, 283  
originalModel, 275  
ownObjects, 302  
parallelMode, 304  
parentModel, 297  
passInEventHandler, 298  
passInMessageHandler, 298  
passInPriorities, 298  
passInSolverCharacteristics, 298  
passInSubTreeModel, 294  
passInTreeHandler, 294  
penaltyScaleFactor, 282  
phase, 292  
previousBounds, 306  
printFrequency, 283  
priority, 298  
probingInfo, 308  
problemFeasibility, 294  
problemType, 282  
pseudoShadow, 307  
randomNumberGenerator, 308  
redoWalkBack, 310  
reducedCostFix, 305  
referenceSolver, 303  
reserveCurrentSolution, 290  
resetModel, 303  
resetToReferenceSolver, 303  
resolve, 274, 305  
resolveAfterTakeOffCuts, 293  
rootObjectiveAfterCuts, 292  
saveBestSolution, 305  
saveExtraSolution, 305  
saveModel, 275  
saveReferenceSolver, 303

savedSolution, 291  
savedSolutionObjective, 292  
sayEventHappened, 300  
searchStrategy, 295  
secondaryStatus, 284  
setAllowableFractionGap, 279  
setAllowableGap, 278  
setAllowablePercentageGap, 279  
setApplicationData, 298  
setBestObjectiveValue, 289  
setBestSolution, 289, 291  
setBestSolutionBasis, 310  
setBranchingMethod, 295  
setContinuousInfeasibilities, 292  
setContinuousObjective, 292  
setContinuousPriority, 309  
setCurrentPassNumber, 281  
setCutModifier, 295  
setCutoff, 277  
setCutoffAsConstraint, 301  
setCutoffIncrement, 279  
setDbIParam, 277  
setDefaultHandler, 299  
setFastNodeDepth, 309  
setHeuristicFractionGap, 279  
setHeuristicGap, 279  
setHeuristicModel, 294  
setHotstartSolution, 280  
setHowOftenGlobalScan, 282  
setInfeasibilityWeight, 278  
setInfoInChild, 305  
setIntParam, 277  
setIntegerTolerance, 278  
setLanguage, 299  
setLastHeuristic, 297  
setLogLevel, 299  
setMIPStart, 310  
setMaximumCutPasses, 280  
setMaximumCutPassesAtRoot, 280  
setMaximumNodes, 277  
setMaximumNumberIterations, 309  
setMaximumSavedSolutions, 291  
setMaximumSeconds, 278  
setMaximumSolutions, 277  
setMinimizationObjValue, 290  
setMinimumDrop, 280  
setModelOwnsSolver, 302  
setMoreSpecialOptions, 300  
setMoreSpecialOptions2, 301  
setMultipleRootTries, 300  
setNextRowCut, 308  
setNodeComparison, 294  
setNumberAnalyzeIterations, 282  
setNumberBeforeTrust, 281  
setNumberHeuristicSolutions, 292  
setNumberHeuristics, 297  
setNumberObjects, 276  
setNumberPenalties, 282  
setNumberStrong, 281  
setNumberStrongIterations, 308  
setNumberThreads, 304  
setObjSense, 292  
setObjValue, 291  
setObjectiveValue, 307  
setOptionalInteger, 276  
setOriginalColumns, 283  
setParentModel, 297  
setPenaltyScaleFactor, 282  
setPointers, 305  
setPreferredWay, 281  
setPrintFrequency, 283  
setPrintingMode, 278  
setProblemFeasibility, 294  
setProblemStatus, 284  
setProblemType, 282  
setRandomSeed, 300  
setResolveAfterTakeOffCuts, 293  
setSearchStrategy, 296  
setSecondaryStatus, 285  
setSolutionCount, 291  
setSpecialOptions, 299  
setStateOfSearch, 295  
setStopNumberIterations, 293  
setStoredRowCuts, 310  
setStrategy, 297  
setStrongStrategy, 296  
setTemporaryPointer, 301  
setTestSolution, 289  
setThreadMode, 304  
setTypePresolve, 295  
setUseElapsedTime, 301  
setWhenCuts, 281  
solver, 302  
solverCharacteristics, 298  
specialOptions, 300  
splitModel, 305  
startSplitModel, 305  
stateOfSearch, 295  
status, 284  
storedRowCuts, 309  
strategy, 296  
strongInfo, 309  
strongStrategy, 296  
subTreeModel, 294  
sumChangeObjective, 293  
swapSolver, 302  
synchronizeHandlers, 305  
synchronizeModel, 276



- synchronizeNumberBeforeTrust, 307
- takeOffCuts, 306
- temporaryPointer, 301
- testSolution, 289
- tightenVubs, 275
- topOfTree, 282
- tree, 294
- typePresolve, 294
- unlockThread, 304
- useElapsedTime, 301
- usedInSolution, 289
- usefulInformation, 310
- virginCutGenerator, 296
- waitingForMiniBranchAndBound, 300
- walkback, 304
- whenCuts, 281
- whichGenerator, 274
- workingBasis, 293
- zapIntegerInformation, 307
- cbcModel
  - CbcBranchDecision, 35
  - CbcOsiSolver, 347
- CbcModel.hpp
  - callCbc, 532
  - callCbc1, 532
  - CbcMain, 532
  - CbcMain0, 532
  - CbcMain1, 532
  - getIntegerInformation, 532
  - setCutAndHeuristicOptions, 532
- cbcModel\_
  - CbcOsiSolver, 347
- CbcNWay, 326
  - ~CbcNWay, 328
  - applyConsequence, 328
  - CbcNWay, 328
  - CbcNWay, 328
  - clone, 328
  - consequence\_, 329
  - createCbcBranch, 328
  - feasibleRegion, 328
  - infeasibility, 328
  - members, 329
  - members\_, 329
  - numberMembers, 329
  - numberMembers\_, 329
  - operator=, 328
  - redoSequenceEtc, 329
  - setConsequence, 328
- CbcNWayBranchingObject, 329
  - ~CbcNWayBranchingObject, 331
  - branch, 331
  - CbcNWayBranchingObject, 331
  - CbcNWayBranchingObject, 331
- clone, 331
- compareBranchingObject, 332
- compareOriginalObject, 331
- numberBranches, 331
- operator=, 331
- print, 331
- twoWay, 331
- type, 331
- CbcNode, 310
  - ~CbcNode, 313
  - active, 317
  - analyze, 315
  - branch, 315
  - branchingObject, 317
  - CbcNode, 313
  - CbcNode, 313
  - checkInfo, 318
  - checkIsCutoff, 315
  - chooseBranch, 313
  - chooseClpBranch, 315
  - chooseDynamicBranch, 314
  - chooseOsiBranch, 314
  - createInfo, 313
  - decrementCuts, 315
  - decrementParentCuts, 315
  - depth, 316
  - getState, 317
  - guessedObjectiveValue, 316
  - initializeInfo, 315
  - modifiableBranchingObject, 317
  - nodeInfo, 315
  - nodeNumber, 317
  - nullNodeInfo, 315
  - numberBranches, 316
  - numberUnsatisfied, 316
  - objectiveValue, 316
  - onTree, 317
  - operator=, 313
  - print, 318
  - setActive, 317
  - setBranchingObject, 317
  - setDepth, 316
  - setGuessedObjectiveValue, 316
  - setNodeNumber, 317
  - setNumberUnsatisfied, 316
  - setObjectiveValue, 316
  - setOnTree, 317
  - setState, 317
  - setSumInfeasibilities, 316
  - sumInfeasibilities, 316
  - way, 316
- CbcNodeInfo, 318
  - ~CbcNodeInfo, 321
  - active\_, 326



- addCuts, 323
- allActivated, 325
- allBranchesGone, 322
- applyBounds, 322
- applyToModel, 321
- branchedOn, 323
- buildRowBasis, 322
- CbcNodeInfo, 321
- CbcNodeInfo, 321
- clone, 322
- cuts, 324
- cuts\_, 326
- deactivate, 324
- decrement, 322
- decrementCuts, 323
- decrementParentCuts, 324
- deleteCut, 323
- deleteCuts, 323
- increment, 322
- incrementCuts, 324
- incrementNumberPointingToThis, 323
- incrementParentCuts, 324
- initializeInfo, 322
- mark, 325
- marked, 325
- mutableOwner, 324
- nodeNumber, 324
- nodeNumber\_, 326
- nullOwner, 324
- nullParent, 323
- numberBranchesLeft, 322
- numberBranchesLeft\_, 326
- numberCuts, 324
- numberCuts\_, 326
- numberPointingToThis, 323
- numberPointingToThis\_, 325
- numberRows\_, 326
- owner, 324
- owner\_, 325
- parent, 323
- parent\_, 325
- parentBranch, 325
- parentBranch\_, 325
- setNodeNumber, 324
- setNumberBranchesLeft, 322
- setNumberCuts, 324
- setNumberPointingToThis, 323
- throwAway, 323
- unmark, 325
- unsetParentBasedData, 325
- CbcObject, 332
  - ~CbcObject, 334
  - CbcObject, 334
  - CbcObject, 334
- clone, 335
- createBranch, 336
- createCbcBranch, 335
- createOsiBranch, 336
- createUpdateInformation, 337
- feasibleRegion, 335
- floorCeiling, 336
- id, 337
- id\_, 338
- infeasibility, 335
- initializeForBranching, 338
- model, 338
- model\_, 338
- notPreferredNewFeasible, 336
- operator=, 335
- optionalObject, 337
- position, 337
- position\_, 338
- preferredNewFeasible, 336
- preferredWay, 338
- preferredWay\_, 338
- redoSequenceEtc, 338
- resetBounds, 336
- setId, 337
- setModel, 337
- setPosition, 337
- setPreferredWay, 338
- solverBranch, 336
- updateInformation, 337
- CbcObjectUpdateData, 339
  - ~CbcObjectUpdateData, 340
  - branchingValue\_, 341
  - CbcObjectUpdateData, 340
  - CbcObjectUpdateData, 340
  - change\_, 340
  - cutoff\_, 341
  - intDecrease\_, 340
  - object\_, 340
  - objectNumber\_, 340
  - operator=, 340
  - originalObjective\_, 341
  - status\_, 340
  - way\_, 340
- CbcOsiParam, 341
  - ~CbcOsiParam, 345
  - CbcOsiParam, 344, 345
  - CbcOsiParamCode, 343
  - CbcOsiParam, 344, 345
  - clone, 345
  - obj, 345
  - operator=, 345
  - paramCode, 345
  - setObj, 345
  - setParamCode, 345

CbcOsiParamCode  
     CbcOsiParam, 343  
 CbcOsiParamUtils, 22  
     addCbcOsiParams, 22  
     loadOsiParamObj, 22  
     pushCbcOsiDbl, 22  
     pushCbcOsiHint, 22  
     pushCbcOsiInt, 22  
     pushCbcOsiKwd, 22  
     pushCbcOsiLogLevel, 22  
     setOsiSolverInterfaceDefaults, 22  
 CbcOsiSolver, 346  
     ~CbcOsiSolver, 347  
     cbcModel, 347  
     cbcModel\_, 347  
     CbcOsiSolver, 347  
     CbcOsiSolver, 347  
     clone, 347  
     operator=, 347  
     setCbcModel, 347  
 CbcParam, 348  
     ~CbcParam, 350  
     addHelp, 350  
     append, 350  
     CbcParam, 350  
     CbcParam, 350  
     checkDoubleParameter, 351  
     currentOption, 352  
     displayThis, 352  
     doubleParameter, 351  
     doubleValue, 352  
     indexNumber, 353  
     intParameter, 351  
     intValue, 352  
     matchName, 351  
     matches, 352  
     name, 350  
     operator=, 350  
     parameterOption, 351  
     printLongHelp, 353  
     printOptions, 352  
     printString, 353  
     setCurrentOption, 352  
     setDoubleParameter, 350, 351  
     setDoubleValue, 352  
     setIntParameter, 351  
     setIntValue, 352  
     setLonghelp, 353  
     setStringValue, 352  
     shortHelp, 350  
     stringValue, 352  
     type, 352  
 CbcParam.hpp  
     CbcParameterType, 535

CbcParameterType  
     CbcParam.hpp, 535  
 cbcParams\_  
     CbcGenCtIBlk, 135  
 CbcPartialNodeInfo, 354  
     ~CbcPartialNodeInfo, 355  
     applyBounds, 355  
     applyToModel, 355  
     basisDiff, 355  
     basisDiff\_, 356  
     buildRowBasis, 355  
     CbcPartialNodeInfo, 355  
     CbcPartialNodeInfo, 355  
     clone, 355  
     newBounds, 356  
     newBounds\_, 356  
     numberChangedBounds, 356  
     numberChangedBounds\_, 356  
     variables, 355  
     variables\_, 356  
 CbcRangeCompare  
     CbcBranchBase.hpp, 510  
 CbcRounding, 356  
     ~CbcRounding, 358  
     CbcRounding, 357  
     CbcRounding, 357  
     clone, 358  
     down\_, 359  
     equal\_, 359  
     generateCpp, 358  
     matrix\_, 359  
     matrixByRow\_, 359  
     operator=, 358  
     resetModel, 358  
     seed\_, 359  
     setModel, 358  
     setSeed, 358  
     solution, 358  
     up\_, 359  
     validate, 358  
 CbcRowCuts, 359  
     ~CbcRowCuts, 360  
     addCutIfNotDuplicate, 360  
     addCutIfNotDuplicateWhenGreedy, 360  
     addCuts, 360  
     CbcRowCuts, 360  
     CbcRowCuts, 360  
     cut, 360  
     eraseRowCut, 360  
     numberCuts, 360  
     operator=, 360  
     rowCutPtr, 360  
     sizeRowCuts, 360  
 CbcSOS, 390

- ~CbcSOS, 392
- canDoHeuristics, 394
- CbcSOS, 392
- CbcSOS, 392
- clone, 392
- createCbcBranch, 392
- createUpdateInformation, 393
- feasibleRegion, 392
- infeasibility, 392
- members, 393
- mutableMembers, 394
- mutableWeights, 394
- numberMembers, 393
- numberTimesDown, 393
- numberTimesUp, 394
- operator=, 392
- osiObject, 393
- redoSequenceEtc, 393
- setIntegerValued, 394
- setNumberMembers, 394
- solverBranch, 393
- sosType, 393
- updateInformation, 393
- weights, 394
- CbcSOSBranchingObject, 394
  - ~CbcSOSBranchingObject, 396
  - branch, 396
  - CbcSOSBranchingObject, 396
  - CbcSOSBranchingObject, 396
  - clone, 396
  - compareBranchingObject, 397
  - compareOriginalObject, 396
  - computeNonzeroRange, 397
  - fix, 396
  - operator=, 396
  - previousBranch, 396
  - print, 396
  - type, 396
- CbcSerendipity, 360
  - ~CbcSerendipity, 361
  - CbcSerendipity, 361
  - CbcSerendipity, 361
  - clone, 361
  - generateCpp, 361
  - operator=, 361
  - resetModel, 362
  - setModel, 361
  - solution, 362
- CbcSimpleInteger, 362
  - ~CbcSimpleInteger, 364
  - breakEven, 366
  - breakEven\_, 366
  - CbcSimpleInteger, 364
  - CbcSimpleInteger, 364
- clone, 364
- columnNumber, 365
- columnNumber\_, 366
- createCbcBranch, 364
- feasibleRegion, 364, 365
- fillCreateBranch, 365
- infeasibility, 364
- operator=, 364
- originalLower\_, 366
- originalLowerBound, 365
- originalUpper\_, 366
- originalUpperBound, 365
- osiObject, 364
- preferredWay\_, 366
- resetBounds, 365
- resetSequenceEtc, 365
- setBreakEven, 366
- setColumnNumber, 365
- setOriginalLowerBound, 365
- setOriginalUpperBound, 365
- solverBranch, 365
- CbcSimpleIntegerDynamicPseudoCost, 366
  - ~CbcSimpleIntegerDynamicPseudoCost, 371
  - addToSumDownChange, 374
  - addToSumDownCost, 374
  - addToSumDownDecrease, 375
  - addToSumUpChange, 375
  - addToSumUpCost, 374
  - addToSumUpDecrease, 375
  - CbcSimpleIntegerDynamicPseudoCost, 371
  - CbcSimpleIntegerDynamicPseudoCost, 371
  - clone, 371
  - copySome, 372
  - createCbcBranch, 372
  - createUpdateInformation, 372
  - downDynamicPseudoCost, 373
  - downDynamicPseudoCost\_, 378
  - downEstimate, 377
  - downShadowPrice, 373
  - downShadowPrice\_, 378
  - incrementNumberBeforeTrust, 377
  - incrementNumberTimesDown, 376
  - incrementNumberTimesDownInfeasible, 376
  - incrementNumberTimesUp, 376
  - incrementNumberTimesUpInfeasible, 377
  - infeasibility, 372
  - lastDownCost\_, 379
  - lastDownDecrease\_, 379
  - lastUpCost\_, 379
  - lastUpDecrease\_, 379
  - method, 377
  - method\_, 380
  - numberBeforeTrust, 377
  - numberBeforeTrust\_, 380

numberTimesDown, 375  
 numberTimesDown\_, 379  
 numberTimesDownInfeasible, 376  
 numberTimesDownInfeasible\_, 379  
 numberTimesDownLocalFixed\_, 380  
 numberTimesDownTotalFixed\_, 380  
 numberTimesProbingTotal\_, 380  
 numberTimesUp, 376  
 numberTimesUp\_, 379  
 numberTimesUpInfeasible, 376  
 numberTimesUpInfeasible\_, 380  
 numberTimesUpLocalFixed\_, 380  
 numberTimesUpTotalFixed\_, 380  
 operator=, 372  
 print, 378  
 same, 378  
 setDownDynamicPseudoCost, 373  
 setDownInformation, 377  
 setDownShadowPrice, 373  
 setMethod, 377  
 setNumberBeforeTrust, 377  
 setNumberTimesDown, 376  
 setNumberTimesDownInfeasible, 376  
 setNumberTimesUp, 376  
 setNumberTimesUpInfeasible, 376  
 setProbingInformation, 377  
 setSumDownChange, 374  
 setSumDownCost, 374  
 setSumDownDecrease, 375  
 setSumUpChange, 375  
 setSumUpCost, 374  
 setSumUpDecrease, 375  
 setUpDownSeparator, 374  
 setUpDynamicPseudoCost, 373  
 setUpInformation, 377  
 setUpShadowPrice, 373  
 solverBranch, 372  
 sumDownChange, 374  
 sumDownChange\_, 378  
 sumDownCost, 374  
 sumDownCost\_, 378  
 sumDownDecrease, 375  
 sumDownDecrease\_, 379  
 sumUpChange, 375  
 sumUpChange\_, 378  
 sumUpCost, 374  
 sumUpCost\_, 378  
 sumUpDecrease, 375  
 sumUpDecrease\_, 379  
 upDownSeparator, 373  
 upDownSeparator\_, 378  
 upDynamicPseudoCost, 373  
 upDynamicPseudoCost\_, 378  
 upEstimate, 377

upShadowPrice, 373  
 upShadowPrice\_, 379  
 updateAfter, 372  
 updateAfterMini, 372  
 updateBefore, 372  
 updateDownDynamicPseudoCost, 373  
 updateInformation, 372  
 updateUpDynamicPseudoCost, 373  
 CbcSimpleIntegerDynamicPseudoCost.hpp  
   INFEAS, 539  
   MOD\_SHADOW, 539  
   TYPE2, 539  
   TYPERATIO, 539  
   WEIGHT\_AFTER, 539  
   WEIGHT\_BEFORE, 540  
   WEIGHT\_PRODUCT, 540  
 CbcSimpleIntegerPseudoCost, 381  
   ~CbcSimpleIntegerPseudoCost, 382  
   CbcSimpleIntegerPseudoCost, 382  
   CbcSimpleIntegerPseudoCost, 382  
   clone, 382  
   createCbcBranch, 383  
   downEstimate, 383  
   downPseudoCost, 383  
   downPseudoCost\_, 384  
   infeasibility, 383  
   method, 383  
   method\_, 384  
   operator=, 382  
   setDownPseudoCost, 383  
   setMethod, 384  
   setUpDownSeparator, 383  
   setUpPseudoCost, 383  
   upDownSeparator, 383  
   upDownSeparator\_, 384  
   upEstimate, 383  
   upPseudoCost, 383  
   upPseudoCost\_, 384  
 CbcSolver, 384  
   ~CbcSolver, 386  
   addCutGenerator, 387  
   addUserFunction, 387  
   analyze, 387  
   babModel, 388  
   CbcSolver, 386  
   CbcSolver, 386  
   cutGeneratorArray, 389  
   doubleValue, 388  
   fillParameters, 387  
   fillValuesInSolver, 387  
   intValue, 387  
   model, 388  
   numberCutGenerators, 389  
   numberUserFunctions, 388

- operator=, [387](#)
- originalCoinModel, [388](#)
- originalSolver, [388](#)
- setDoubleValue, [388](#)
- setIntValue, [388](#)
- setOriginalCoinModel, [388](#)
- setOriginalSolver, [388](#)
- setPrinting, [389](#)
- setReadMode, [389](#)
- setUserCallBack, [387](#)
- solve, [387](#)
- startTime, [389](#)
- updateModel, [387](#)
- userFunction, [388](#)
- userFunctionArray, [388](#)
- CbcSolverAnalyze.hpp
  - analyze, [541](#)
- CbcSolverExpandKnapsack.hpp
  - afterKnapsack, [541](#)
  - expandKnapsack, [541](#)
- CbcSolverHeuristics.hpp
  - crunchIt, [542](#)
  - doHeuristics, [542](#)
  - fixVubs, [542](#)
- CbcSolverUsefulData, [389](#)
  - branchDirection\_, [390](#)
  - primalSolution\_, [390](#)
  - priorities\_, [390](#)
  - pseudoDown\_, [390](#)
  - pseudoUp\_, [390](#)
  - sosPriority\_, [390](#)
- CbcStatistics, [397](#)
  - ~CbcStatistics, [398](#)
  - CbcStatistics, [398](#)
  - CbcStatistics, [398](#)
  - depth, [399](#)
  - depth\_, [400](#)
  - endOfBranch, [398](#)
  - endingInfeasibility, [399](#)
  - endingInfeasibility\_, [400](#)
  - endingObjective, [399](#)
  - endingObjective\_, [399](#)
  - id\_, [400](#)
  - node, [399](#)
  - numberIterations, [399](#)
  - numberIterations\_, [400](#)
  - operator=, [398](#)
  - parentId\_, [400](#)
  - parentNode, [399](#)
  - print, [398](#)
  - sayInfeasible, [398](#)
  - sequence\_, [400](#)
  - startingInfeasibility, [399](#)
  - startingInfeasibility\_, [400](#)
  - startingObjective, [399](#)
  - startingObjective\_, [399](#)
  - updateInfeasibility, [398](#)
  - value, [399](#)
  - value\_, [399](#)
  - way, [399](#)
  - way\_, [400](#)
- CbcStopNow, [400](#)
  - ~CbcStopNow, [401](#)
  - callBack, [401](#)
  - CbcStopNow, [401](#)
  - CbcStopNow, [401](#)
  - clone, [402](#)
  - operator=, [402](#)
- CbcStrategy, [402](#)
  - ~CbcStrategy, [403](#)
  - CbcStrategy, [403](#)
  - CbcStrategy, [403](#)
  - clone, [403](#)
  - deletePreProcess, [404](#)
  - depth\_, [405](#)
  - fullNodeInfo, [405](#)
  - generateCpp, [405](#)
  - getNested, [404](#)
  - partialNodeInfo, [405](#)
  - preProcessState, [404](#)
  - preProcessState\_, [405](#)
  - process, [404](#)
  - process\_, [405](#)
  - setNested, [404](#)
  - setPreProcessState, [404](#)
  - setupCutGenerators, [404](#)
  - setupHeuristics, [404](#)
  - setupOther, [404](#)
  - setupPrinting, [404](#)
  - status, [405](#)
- CbcStrategyDefault, [405](#)
  - ~CbcStrategyDefault, [407](#)
  - CbcStrategyDefault, [407](#)
  - CbcStrategyDefault, [407](#)
  - clone, [407](#)
  - cutsOnlyAtRoot\_, [408](#)
  - desiredPreProcess, [407](#)
  - desiredPreProcess\_, [408](#)
  - generateCpp, [407](#)
  - numberBeforeTrust\_, [408](#)
  - numberStrong\_, [408](#)
  - preProcessPasses, [407](#)
  - preProcessPasses\_, [408](#)
  - printLevel\_, [408](#)
  - setupCutGenerators, [407](#)
  - setupHeuristics, [407](#)
  - setupOther, [407](#)
  - setupPreProcessing, [407](#)

- setupPrinting, 407
- CbcStrategyDefaultSubTree, 408
  - ~CbcStrategyDefaultSubTree, 409
  - CbcStrategyDefaultSubTree, 409
  - CbcStrategyDefaultSubTree, 409
  - clone, 409
  - cutsOnlyAtRoot\_, 410
  - numberBeforeTrust\_, 410
  - numberStrong\_, 410
  - parentModel\_, 410
  - printLevel\_, 410
  - setupCutGenerators, 409
  - setupHeuristics, 409
  - setupOther, 410
  - setupPrinting, 410
- CbcStrategyNull, 410
  - ~CbcStrategyNull, 411
  - CbcStrategyNull, 411
  - CbcStrategyNull, 411
  - clone, 411
  - setupCutGenerators, 411
  - setupHeuristics, 411
  - setupOther, 412
  - setupPrinting, 412
- CbcStrongInfo, 412
  - downMovement, 413
  - finishedDown, 413
  - finishedUp, 413
  - fix, 414
  - numIntInfeasDown, 413
  - numIntInfeasUp, 413
  - numItersDown, 414
  - numItersUp, 413
  - numObjInfeasDown, 413
  - numObjInfeasUp, 413
  - objectNumber, 414
  - possibleBranch, 413
  - upMovement, 413
- CbcThread, 414
  - ~CbcThread, 414
  - CbcThread, 414
  - CbcThread, 414
- CbcTree, 414
  - ~CbcTree, 417
  - addBranchingInformation, 420
  - bestAlternate, 419
  - bestNode, 418
  - branched, 420
  - branched\_, 421
  - CbcTree, 417
  - CbcTree, 417
  - cleanTree, 419
  - clone, 418
  - comparison\_, 421
  - empty, 418
  - endSearch, 419
  - fixTop, 419
  - generateCpp, 418
  - getBestPossibleObjective, 419
  - getMaximumBranching, 420
  - getNumberBranching, 420
  - increaseSpace, 420
  - lastDepth, 420
  - lastDepth\_, 421
  - lastObjective, 420
  - lastObjective\_, 421
  - lastUnsatisfied, 420
  - lastUnsatisfied\_, 421
  - maximumBranching\_, 421
  - maximumNodeNumber, 419
  - maximumNodeNumber\_, 421
  - newBound\_, 421
  - newBounds, 420
  - nodePointer, 419
  - nodes\_, 421
  - numberBranching\_, 421
  - operator=, 418
  - pop, 418
  - push, 418
  - realpop, 419
  - realpush, 419
  - rebuild, 418
  - resetNodeNumbers, 419
  - setComparison, 418
  - setMaximumBranching, 420
  - setNumberBranching, 419
  - size, 418
  - top, 418
- CbcTreeLocal, 422
  - ~CbcTreeLocal, 423
  - CbcTreeLocal, 423
  - CbcTreeLocal, 423
  - clone, 423
  - createCut, 424
  - deleteCut, 424
  - empty, 424
  - endSearch, 424
  - generateCpp, 423
  - maxDiversification, 424
  - nodeLimit, 425
  - operator=, 423
  - passInSolution, 424
  - pop, 423
  - push, 423
  - range, 424
  - refine, 425
  - reverseCut, 424
  - setMaxDiversification, 424

- setNodeLimit, [425](#)
- setRange, [424](#)
- setRefine, [425](#)
- setTimeLimit, [425](#)
- setTypeCuts, [424](#)
- timeLimit, [424](#)
- top, [423](#)
- typeCuts, [424](#)
- CbcTreeVariable, [425](#)
  - ~CbcTreeVariable, [426](#)
  - CbcTreeVariable, [426](#)
  - CbcTreeVariable, [426](#)
  - clone, [426](#)
  - createCut, [427](#)
  - deleteCut, [427](#)
  - empty, [427](#)
  - endSearch, [427](#)
  - generateCpp, [427](#)
  - maxDiversification, [428](#)
  - nodeLimit, [428](#)
  - operator=, [426](#)
  - passInSolution, [427](#)
  - pop, [427](#)
  - push, [427](#)
  - range, [427](#)
  - refine, [428](#)
  - reverseCut, [427](#)
  - setMaxDiversification, [428](#)
  - setNodeLimit, [428](#)
  - setRange, [427](#)
  - setRefine, [428](#)
  - setTimeLimit, [428](#)
  - setTypeCuts, [428](#)
  - timeLimit, [428](#)
  - top, [427](#)
  - typeCuts, [428](#)
- CbcUser, [428](#)
  - ~CbcUser, [430](#)
  - canDo, [431](#)
  - CbcUser, [430](#)
  - CbcUser, [430](#)
  - clone, [431](#)
  - coinModel, [430](#)
  - coinModel\_, [431](#)
  - exportData, [430](#)
  - exportSolution, [430](#)
  - fillInformation, [430](#)
  - importData, [430](#)
  - name, [431](#)
  - operator=, [431](#)
  - solve, [431](#)
  - stuff, [430](#)
  - userName\_, [431](#)
- CglTemporary, [431](#)
  - ~CglTemporary, [432](#)
  - CglTemporary, [432](#)
  - CglTemporary, [432](#)
  - clone, [433](#)
  - generateCuts, [432](#)
  - operator=, [433](#)
- change\_
  - CbcObjectUpdateData, [340](#)
- changeInGuessed
  - CbcDynamicPseudoCostBranchingObject, [98](#)
  - CbcIntegerPseudoCostBranchingObject, [245](#)
- changeInGuessed\_
  - CbcDynamicPseudoCostBranchingObject, [99](#)
  - CbcIntegerPseudoCostBranchingObject, [245](#)
- checkDoubleParameter
  - CbcParam, [351](#)
- checkInfeasibility
  - OsiBiLinear, [446](#)
- checkInfo
  - CbcNode, [318](#)
- checkIsCutoff
  - CbcNode, [315](#)
- checkModel
  - CbcModel, [302](#)
- checkPossible
  - CbcFathomDynamicProgramming, [108](#)
- checkSolution
  - CbcModel, [289](#)
- chooseBranch
  - CbcModel, [306](#)
  - CbcNode, [313](#)
- chooseClpBranch
  - CbcNode, [315](#)
- chooseDynamicBranch
  - CbcNode, [314](#)
- chooseMethod
  - CbcBranchDecision, [35](#)
- chooseMethod\_
  - CbcBranchDecision, [36](#)
- chooseOsiBranch
  - CbcNode, [314](#)
- chooseStrong\_
  - CbcGenCtlBlk, [137](#)
- chooseVariable
  - OsiChooseStrongSubset, [475](#)
- chosen\_
  - OsiBiLinear, [451](#)
- cleanDive
  - CbcCompareDefault, [68](#)
- cleanMatrix
  - OsiSolverLink, [496](#)
- cleanTree
  - CbcTree, [419](#)
- clearContinuousSolver



- CbcModel, [303](#)
- clearNumberGlobalViolations
  - CbcModel, [293](#)
- CliqueBranchObj
  - CbcBranchingObject.hpp, [512](#)
- cliquePseudoCosts
  - CbcModel, [307](#)
- cliqueType
  - CbcClique, [57](#)
- cliqueType\_
  - CbcClique, [58](#)
- clone
  - CbcBranchAllDifferent, [29](#)
  - CbcBranchCut, [31](#)
  - CbcBranchDecision, [34](#)
  - CbcBranchDefaultDecision, [37](#)
  - CbcBranchDynamicDecision, [39](#)
  - CbcBranchingObject, [43](#)
  - CbcBranchToFixLots, [48](#)
  - CbcCbcParam, [53](#)
  - CbcClique, [56](#)
  - CbcCliqueBranchingObject, [59](#)
  - CbcCompareBase, [63](#)
  - CbcCompareDefault, [66](#)
  - CbcCompareDepth, [70](#)
  - CbcCompareEstimate, [71](#)
  - CbcCompareObjective, [72](#)
  - CbcConsequence, [73](#)
  - CbcCutBranchingObject, [78](#)
  - CbcCutModifier, [91](#)
  - CbcCutSubsetModifier, [92](#)
  - CbcDummyBranchingObject, [94](#)
  - CbcDynamicPseudoCostBranchingObject, [97](#)
  - CbcEventHandler, [102](#)
  - CbcFathom, [104](#)
  - CbcFathomDynamicProgramming, [107](#)
  - CbcFeasibilityBase, [111](#)
  - CbcFixingBranchingObject, [112](#)
  - CbcFixVariable, [114](#)
  - CbcFollowOn, [117](#)
  - CbcFullNodeInfo, [120](#)
  - CbcGeneral, [138](#)
  - CbcGenParam, [143](#)
  - CbcHeuristic, [148](#)
  - CbcHeuristicCrossover, [156](#)
  - CbcHeuristicDINS, [159](#)
  - CbcHeuristicDive, [163](#)
  - CbcHeuristicDiveCoefficient, [167](#)
  - CbcHeuristicDiveFractional, [169](#)
  - CbcHeuristicDiveGuided, [171](#)
  - CbcHeuristicDiveLineSearch, [172](#)
  - CbcHeuristicDivePseudoCost, [174](#)
  - CbcHeuristicDiveVectorLength, [176](#)
  - CbcHeuristicDW, [181](#)
  - CbcHeuristicDynamic3, [191](#)
  - CbcHeuristicFPump, [195](#)
  - CbcHeuristicGreedyCover, [202](#)
  - CbcHeuristicGreedyEquality, [205](#)
  - CbcHeuristicGreedySOS, [208](#)
  - CbcHeuristicJustOne, [211](#)
  - CbcHeuristicLocal, [213](#)
  - CbcHeuristicNaive, [216](#)
  - CbcHeuristicPartial, [220](#)
  - CbcHeuristicPivotAndFix, [222](#)
  - CbcHeuristicProximity, [224](#)
  - CbcHeuristicRandRound, [226](#)
  - CbcHeuristicRENS, [228](#)
  - CbcHeuristicRINS, [231](#)
  - CbcHeuristicVND, [234](#)
  - CbcIdiotBranch, [237](#)
  - CbcIntegerBranchingObject, [241](#)
  - CbcIntegerPseudoCostBranchingObject, [244](#)
  - CbcLongCliqueBranchingObject, [247](#)
  - CbcLotsize, [249](#)
  - CbcLotsizeBranchingObject, [253](#)
  - CbcModel, [302](#)
  - CbcNodeInfo, [322](#)
  - CbcNWay, [328](#)
  - CbcNWayBranchingObject, [331](#)
  - CbcObject, [335](#)
  - CbcOsiParam, [345](#)
  - CbcOsiSolver, [347](#)
  - CbcPartialNodeInfo, [355](#)
  - CbcRounding, [358](#)
  - CbcSerendipity, [361](#)
  - CbcSimpleInteger, [364](#)
  - CbcSimpleIntegerDynamicPseudoCost, [371](#)
  - CbcSimpleIntegerPseudoCost, [382](#)
  - CbcSOS, [392](#)
  - CbcSOSBranchingObject, [396](#)
  - CbcStopNow, [402](#)
  - CbcStrategy, [403](#)
  - CbcStrategyDefault, [407](#)
  - CbcStrategyDefaultSubTree, [409](#)
  - CbcStrategyNull, [411](#)
  - CbcTree, [418](#)
  - CbcTreeLocal, [423](#)
  - CbcTreeVariable, [426](#)
  - CbcUser, [431](#)
  - CglTemporary, [433](#)
  - ClpAmplObjective, [436](#)
  - ClpConstraintAmpl, [438](#)
  - OsiBiLinear, [445](#)
  - OsiBiLinearBranchingObject, [452](#)
  - OsiBiLinearEquality, [454](#)
  - OsiCbcSolverInterface, [472](#)
  - OsiChooseStrongSubset, [475](#)
  - OsiLink, [477](#)



- OsiLinkBranchingObject, 479
- OsiOldLink, 482
- OsiOldLinkBranchingObject, 484
- OsiSimpleFixedInteger, 488
- OsiSolverLinearizedQuadratic, 490
- OsiSolverLink, 495
- OsiUsesBiLinear, 502
- cloneBut
  - CbcHeuristic, 152
- ClpAmplObjective, 434
  - ~ClpAmplObjective, 435
  - clone, 436
  - ClpAmplObjective, 435
  - ClpAmplObjective, 435
  - deleteSome, 435
  - gradient, 435
  - linearObjective, 436
  - markNonlinear, 436
  - newXValues, 436
  - objectiveValue, 435
  - operator=, 436
  - reallyScale, 436
  - reducedGradient, 435
  - resize, 435
  - stepLength, 435
- ClpConstraintAmpl, 436
  - ~ClpConstraintAmpl, 437
  - clone, 438
  - ClpConstraintAmpl, 437
  - ClpConstraintAmpl, 437
  - coefficient, 439
  - column, 438
  - deleteSome, 438
  - gradient, 438
  - markNonlinear, 438
  - markNonzero, 438
  - newXValues, 438
  - numberCoefficients, 438
  - operator=, 438
  - reallyScale, 438
  - resize, 438
- coefficient
  - ClpConstraintAmpl, 439
  - OsiBiLinear, 446
- coefficient\_
  - OsiBiLinear, 449
- coefficients\_
  - CbcFathomDynamicProgramming, 109
- CoinHashLink, 439
  - index, 439
  - next, 439
- coinModel
  - CbcUser, 430
  - OsiSolverLink, 498
- coinModel\_
  - CbcUser, 431
  - OsiSolverLink, 499
- column
  - ClpConstraintAmpl, 438
- columnLower
  - ampl\_info, 24
- columnNumber
  - CbcLotsize, 250
  - CbcSimpleInteger, 365
- columnNumber\_
  - CbcSimpleInteger, 366
- columnStatus
  - ampl\_info, 25
- columnUpper
  - ampl\_info, 24
- columnsInBlock
  - CbcHeuristicDW, 185
- columnsInBlock\_
  - CbcHeuristicDW, 188
- compareBranchingObject
  - CbcBranchingObject, 46
  - CbcCliqueBranchingObject, 60
  - CbcCutBranchingObject, 78
  - CbcDummyBranchingObject, 95
  - CbcFixingBranchingObject, 113
  - CbcIntegerBranchingObject, 242
  - CbcIntegerPseudoCostBranchingObject, 245
  - CbcLongCliqueBranchingObject, 247
  - CbcLotsizeBranchingObject, 253
  - CbcNWayBranchingObject, 332
  - CbcSOSBranchingObject, 397
- compareNodes
  - CbcCompare, 61
- compareOriginalObject
  - CbcBranchingObject, 46
  - CbcCliqueBranchingObject, 60
  - CbcCutBranchingObject, 78
  - CbcDummyBranchingObject, 95
  - CbcFixingBranchingObject, 113
  - CbcLongCliqueBranchingObject, 247
  - CbcNWayBranchingObject, 331
  - CbcSOSBranchingObject, 396
- comparison\_
  - CbcTree, 421
- comparisonObject
  - CbcCompare, 61
- computeCompleteSolution
  - CbcMipStartIO.hpp, 531
- computeLambdas
  - OsiBiLinear, 449
- computeNonzeroRange
  - CbcSOSBranchingObject, 397
- config\_cbc\_default.h

- CBC\_VERSION, [545](#)
- config\_default.h
  - COIN\_HAS\_CGL, [545](#)
  - COIN\_HAS\_CLP, [545](#)
  - COIN\_HAS\_OSI, [546](#)
  - COIN\_HAS\_VOL, [546](#)
- conflictCut
  - CbcModel, [283](#)
- consequence\_
  - CbcNWay, [329](#)
- continuousPriority
  - CbcModel, [309](#)
- continuousSolution
  - CbcHeuristicDW, [183](#)
  - CbcModel, [288](#)
- continuousSolution\_
  - CbcHeuristicDW, [186](#)
- continuousSolver
  - CbcModel, [303](#)
- convertToDynamic
  - CbcModel, [307](#)
- convex\_
  - OsiSolverLink, [499](#)
- convexity\_
  - OsiBiLinear, [451](#)
- copySome
  - CbcSimpleIntegerDynamicPseudoCost, [372](#)
- cost
  - CbcFathomDynamicProgramming, [108](#)
- cost\_
  - CbcFathomDynamicProgramming, [108](#)
- createBranch
  - CbcObject, [336](#)
  - OsiBiLinear, [446](#)
  - OsiLink, [477](#)
  - OsiOldLink, [482](#)
  - OsiSimpleFixedInteger, [488](#)
  - OsiUsesBiLinear, [502](#)
- createCbcBranch
  - CbcBranchAllDifferent, [29](#)
  - CbcBranchCut, [31](#)
  - CbcBranchToFixLots, [49](#)
  - CbcCliques, [56](#)
  - CbcFollowOn, [117](#)
  - CbcGeneral, [138](#)
  - CbcIdiotBranch, [238](#)
  - CbcLotsize, [249](#)
  - CbcNWay, [328](#)
  - CbcObject, [335](#)
  - CbcSimpleInteger, [364](#)
  - CbcSimpleIntegerDynamicPseudoCost, [372](#)
  - CbcSimpleIntegerPseudoCost, [383](#)
  - CbcSOS, [392](#)
- createContinuousSolver
  - CbcModel, [303](#)
- createCut
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [427](#)
- createInfo
  - CbcNode, [313](#)
- createOsiBranch
  - CbcObject, [336](#)
- createUpdateInformation
  - CbcObject, [337](#)
  - CbcSimpleIntegerDynamicPseudoCost, [372](#)
  - CbcSOS, [393](#)
- crunchIt
  - CbcSolverHeuristics.hpp, [542](#)
- currentDepth
  - CbcModel, [282](#)
- currentNode
  - CbcModel, [308](#)
- currentNumberCuts
  - CbcModel, [308](#)
- currentOption
  - CbcParam, [352](#)
- currentSolution
  - CbcModel, [289](#)
- cut
  - ampl\_info, [26](#)
  - CbcRowCuts, [360](#)
- CutBranchingObj
  - CbcBranchingObject.hpp, [512](#)
- cutGenerator
  - CbcModel, [296](#)
- cutGeneratorArray
  - CbcSolver, [389](#)
- cutGeneratorName
  - CbcCutGenerator, [83](#)
- cutGenerators
  - CbcModel, [296](#)
- cutModifier
  - CbcModel, [295](#)
- cutoff\_
  - CbcCompareDefault, [68](#)
  - CbcObjectUpdateData, [341](#)
- cuts
  - CbcNodeInfo, [324](#)
- cuts\_
  - CbcNodeInfo, [326](#)
- cutsOnlyAtRoot\_
  - CbcStrategyDefault, [408](#)
  - CbcStrategyDefaultSubTree, [410](#)
- D
- DEBUG
  - CbcGenParam, [141](#)
- DIRECTION

- CbcCbcParam, [52](#)
- DIRECTORY
  - CbcGenParam, [141](#)
- DJFIX
  - CbcGenParam, [141](#)
- DUALBOUND
  - CbcOsiParam, [343](#)
- DUALPIVOT
  - CbcOsiParam, [343](#)
- DUALSIMPLEX
  - CbcOsiParam, [343](#)
- DUALTOLERANCE
  - CbcOsiParam, [343](#)
- DUMMY
  - CbcGenParam, [141](#)
- DWModel
  - CbcHeuristicDW, [182](#)
- deactivate
  - CbcNodeInfo, [324](#)
- dealWithEventHandler
  - CbcModel, [289](#)
- debugCreate\_
  - CbcGenCtlBlk, [136](#)
- debugFile\_
  - CbcGenCtlBlk, [136](#)
- debugNodes
  - CbcHeuristic, [152](#)
- debugSol\_
  - CbcGenCtlBlk, [136](#)
- decayFactor\_
  - CbcHeuristic, [153](#)
- decrement
  - CbcCountRowCut, [75](#)
  - CbcNodeInfo, [322](#)
- decrementCuts
  - CbcNode, [315](#)
  - CbcNodeInfo, [323](#)
- decrementParentCuts
  - CbcNode, [315](#)
  - CbcNodeInfo, [324](#)
- defaultBound
  - OsiSolverLink, [497](#)
- defaultBound\_
  - OsiSolverLink, [500](#)
- defaultHandler
  - CbcModel, [299](#)
- defaultMeshSize
  - OsiSolverLink, [497](#)
- defaultMeshSize\_
  - OsiSolverLink, [500](#)
- defaultRounding
  - CbcHeuristicFPump, [197](#)
- defaultRounding\_
  - CbcHeuristicFPump, [199](#)
- defaultSettings\_
  - CbcGenCtlBlk, [136](#)
- deleteColNames
  - OsiCbcSolverInterface, [467](#)
- deleteCols
  - OsiCbcSolverInterface, [469](#)
- deleteCut
  - CbcNodeInfo, [323](#)
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [427](#)
- deleteCuts
  - CbcNodeInfo, [323](#)
- deleteObjects
  - CbcModel, [276](#)
- deletePreProcess
  - CbcStrategy, [404](#)
- deleteRowNames
  - OsiCbcSolverInterface, [466](#)
- deleteRows
  - OsiCbcSolverInterface, [469](#)
- deleteSavedSolution
  - CbcModel, [292](#)
- deleteSolutions
  - CbcModel, [305](#)
- deleteSome
  - ClpAmplObjective, [435](#)
  - ClpConstraintAmpl, [438](#)
- depth
  - CbcNode, [316](#)
  - CbcStatistics, [399](#)
- depth\_
  - CbcBranchToFixLots, [50](#)
  - CbcStatistics, [400](#)
  - CbcStrategy, [405](#)
- desiredPreProcess
  - CbcStrategyDefault, [407](#)
- desiredPreProcess\_
  - CbcStrategyDefault, [408](#)
- dfltAction\_
  - CbcEventHandler, [102](#)
- dfltDirectory\_
  - CbcGenCtlBlk, [134](#)
- dfltRowColName
  - OsiCbcSolverInterface, [466](#)
- dfltSolver\_
  - CbcGenCtlBlk, [136](#)
- direction
  - ampl\_info, [24](#)
- displayThis
  - CbcParam, [352](#)
- distance
  - CbcHeuristicNode, [218](#)
- djFix\_
  - CbcGenCtlBlk, [137](#)

- djTolerance\_
  - CbcBranchToFixLots, [49](#)
- doAOCuts
  - OsiSolverLink, [495](#)
- doBaCParam
  - CbcGenParamUtils, [21](#)
- doCutsNow
  - CbcModel, [281](#)
- doDebugParam
  - CbcGenParamUtils, [21](#)
- doExitParam
  - CbcGenParamUtils, [21](#)
- doHelpParam
  - CbcGenParamUtils, [21](#)
- doHeuristics
  - CbcSolverHeuristics.hpp, [542](#)
- doHeuristicsAtRoot
  - CbcModel, [307](#)
- doImportParam
  - CbcGenParamUtils, [21](#)
- doNothingParam
  - CbcGenParamUtils, [21](#)
- doOneNode
  - CbcModel, [274](#)
- doPrintMaskParam
  - CbcGenParamUtils, [21](#)
- doSolutionParam
  - CbcGenParamUtils, [22](#)
- doUnimplementedParam
  - CbcGenParamUtils, [22](#)
- doVersionParam
  - CbcGenParamUtils, [22](#)
- doubleArray\_
  - CbcHeuristicDW, [186](#)
- doubleArrays
  - CbcHeuristicDW, [185](#)
- doubleParameter
  - CbcParam, [351](#)
- doubleValue
  - CbcParam, [352](#)
  - CbcSolver, [388](#)
- down\_
  - CbcCutBranchingObject, [79](#)
  - CbcIntegerBranchingObject, [242](#)
  - CbcLotsizeBranchingObject, [254](#)
  - CbcRounding, [359](#)
- downArray\_
  - CbcHeuristicDive, [166](#)
- downBounds
  - CbcIntegerBranchingObject, [241](#)
- downDynamicPseudoCost
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
- downDynamicPseudoCost\_
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
- downEstimate
  - CbcSimpleIntegerDynamicPseudoCost, [377](#)
  - CbcSimpleIntegerPseudoCost, [383](#)
- downLocks\_
  - CbcHeuristicDive, [165](#)
- downMovement
  - CbcStrongInfo, [413](#)
- downPseudoCost
  - CbcSimpleIntegerPseudoCost, [383](#)
- downPseudoCost\_
  - CbcSimpleIntegerPseudoCost, [384](#)
- downShadowPrice
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
- downShadowPrice\_
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
- dualSolution
  - ampl\_info, [25](#)
- DummyBranchObj
  - CbcBranchingObject.hpp, [512](#)
- dwBlock\_
  - CbcHeuristicDW, [187](#)
- dwSolver\_
  - CbcHeuristicDW, [186](#)
- DynamicPseudoCostBranchObj
  - CbcBranchingObject.hpp, [512](#)
- E
- ERRORSALLOWED
  - CbcGenParam, [141](#)
- EXIT
  - CbcGenParam, [141](#)
- EXPORT
  - CbcGenParam, [141](#)
- eaMap\_
  - CbcEventHandler, [103](#)
- eaMapPair
  - CbcEventHandler, [100](#)
- elements
  - ampl\_info, [25](#)
- empty
  - CbcTree, [418](#)
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [427](#)
- endSearch
  - CbcEventHandler, [101](#)
- endOfBranch
  - CbcStatistics, [398](#)
- endSearch
  - CbcTree, [419](#)
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [427](#)
- endingInfeasibility
  - CbcStatistics, [399](#)
- endingInfeasibility\_

- CbcStatistics, [400](#)
- endingObjective
  - CbcStatistics, [399](#)
- endingObjective\_
  - CbcStatistics, [399](#)
- equal\_
  - CbcRounding, [359](#)
- equalityTest
  - CbcCompareBase, [64](#)
- eraseRowCut
  - CbcRowCuts, [360](#)
- event
  - CbcEventHandler, [102](#)
- every1000Nodes
  - CbcCompareBase, [63](#)
  - CbcCompareDefault, [67](#)
- exitNow
  - CbcHeuristic, [149](#)
- expandKnapsack
  - CbcSolverExpandKnapsack.hpp, [541](#)
- exportData
  - CbcUser, [430](#)
- exportSolution
  - CbcUser, [430](#)
- extraRow\_
  - OsiBiLinear, [451](#)
- F
- FAKEBOUND
  - CbcOsiParam, [343](#)
- FLOWCUTS
  - CbcGenParam, [141](#)
- FPUMP
  - CbcGenParam, [141](#)
- FPUMPITS
  - CbcGenParam, [141](#)
- FULLGENERALQUERY
  - CbcGenParam, [141](#)
- fakeCutoff
  - CbcHeuristicFPump, [196](#)
- fakeCutoff\_
  - CbcHeuristicFPump, [199](#)
- fastNodeDepth
  - CbcModel, [309](#)
- fathom
  - CbcFathom, [104](#)
  - CbcFathomDynamicProgramming, [107](#)
  - CbcHeuristicDive, [164](#)
  - OsiSolverLink, [495](#)
- feasibilityPump\_
  - CbcHeuristicProximity, [225](#)
- feasibilityPumpOptions
  - CbcHeuristic, [150](#)
- feasibilityPumpOptions\_
  - CbcHeuristic, [153](#)
- feasible
  - CbcFeasibilityBase, [111](#)
- feasibleRegion
  - CbcBranchCut, [31](#)
  - CbcCliques, [56](#)
  - CbcFollowOn, [117](#)
  - CbcGeneral, [138](#)
  - CbcIdiotBranch, [238](#)
  - CbcLotsize, [249](#)
  - CbcNWay, [328](#)
  - CbcObject, [335](#)
  - CbcSimpleInteger, [364](#), [365](#)
  - CbcSOS, [392](#)
  - OsiBiLinear, [445](#)
  - OsiLink, [477](#)
  - OsiOldLink, [482](#)
  - OsiUsesBiLinear, [503](#)
- feasibleSolution
  - CbcModel, [289](#)
- fillCreateBranch
  - CbcSimpleInteger, [365](#)
- fillInformation
  - CbcUser, [430](#)
- fillParameters
  - CbcSolver, [387](#)
- fillPart
  - CbcDynamicPseudoCostBranchingObject, [98](#)
  - CbcIntegerBranchingObject, [241](#)
- fillPseudoCosts
  - CbcModel, [307](#)
- fillStrongInfo
  - CbcBranchingObject, [43](#)
  - CbcDynamicPseudoCostBranchingObject, [98](#)
- fillValuesInSolver
  - CbcSolver, [387](#)
- findCliques
  - CbcModel, [274](#)
- findIntegers
  - CbcModel, [276](#)
- findRange
  - CbcLotsize, [250](#)
- fingerPrint\_
  - CbcHeuristicDW, [188](#)
- finishedDown
  - CbcStrongInfo, [413](#)
- finishedUp
  - CbcStrongInfo, [413](#)
- first\_
  - CbcGenCtlBlk::cbcParamsInfo\_struct, [353](#)
  - CbcGenCtlBlk::genParamsInfo\_struct, [441](#)
  - CbcGenCtlBlk::osiParamsInfo\_struct, [486](#)
- firstLambda
  - OsiBiLinear, [447](#)

- firstLambda\_
  - OsiBiLinear, [450](#)
- firstOdd\_
  - CbcCutSubsetModifier, [93](#)
- fix
  - CbcBranchingObject, [44](#)
  - CbcIntegerBranchingObject, [241](#)
  - CbcSOSBranchingObject, [396](#)
  - CbcStrongInfo, [414](#)
- fixOnReducedCosts
  - CbcHeuristicFPump, [198](#)
- fixOnReducedCosts\_
  - CbcHeuristicFPump, [200](#)
- fixOtherVariables
  - CbcHeuristicDive, [165](#)
  - CbcHeuristicDivePseudoCost, [174](#)
- fixPriority\_
  - CbcHeuristicPartial, [221](#)
- fixTop
  - CbcTree, [419](#)
- fixVariables\_
  - OsiSolverLink, [500](#)
- fixVubs
  - CbcSolverHeuristics.hpp, [542](#)
- fixedDj
  - CbcHeuristicDW, [183](#)
- fixedDj\_
  - CbcHeuristicDW, [186](#)
- flipModel
  - CbcModel, [276](#)
- floorCeiling
  - CbcLotsize, [250](#)
  - CbcObject, [336](#)
- FollowOnBranchObj
  - CbcBranchingObject.hpp, [512](#)
- fraction
  - CbcHeuristicGreedyEquality, [206](#)
- fraction\_
  - CbcHeuristicGreedyEquality, [206](#)
- fractionFixed\_
  - CbcBranchToFixLots, [49](#)
- fractionSmall
  - CbcHeuristic, [150](#)
- fractionSmall\_
  - CbcHeuristic, [153](#)
- freeArgs
  - Cbc\_ampl.h, [504](#)
- freeArrays1
  - Cbc\_ampl.h, [504](#)
- freeArrays2
  - Cbc\_ampl.h, [504](#)
- fullDWEverySoOften\_
  - CbcHeuristicDW, [188](#)
- fullNodeInfo
  - CbcStrategy, [405](#)
- fullScan
  - CbcCompareBase, [63](#)
- function\_
  - OsiOneLink, [485](#)
- functionPointer\_
  - CbcHeuristicDW, [186](#)
- G
- GAMMA
  - CbcOsiParam, [343](#)
- GAPRATIO
  - CbcCbcParam, [52](#)
- GENERALQUERY
  - CbcGenParam, [140](#)
- GOMORYCUTS
  - CbcGenParam, [141](#)
- GREEDY
  - CbcGenParam, [141](#)
- genParams\_
  - CbcGenCtlBlk, [135](#)
- GeneralDepthBranchObj
  - CbcBranchingObject.hpp, [512](#)
- generateCpp
  - CbcBranchDecision, [35](#)
  - CbcCompareBase, [63](#)
  - CbcCompareDefault, [67](#)
  - CbcCompareDepth, [70](#)
  - CbcCompareEstimate, [71](#)
  - CbcCompareObjective, [72](#)
  - CbcCutModifier, [91](#)
  - CbcCutSubsetModifier, [93](#)
  - CbcHeuristic, [150](#), [151](#)
  - CbcHeuristicCrossover, [156](#)
  - CbcHeuristicDINS, [159](#)
  - CbcHeuristicDive, [163](#)
  - CbcHeuristicDiveCoefficient, [168](#)
  - CbcHeuristicDiveFractional, [169](#)
  - CbcHeuristicDiveGuided, [171](#)
  - CbcHeuristicDiveLineSearch, [172](#)
  - CbcHeuristicDivePseudoCost, [174](#)
  - CbcHeuristicDiveVectorLength, [176](#)
  - CbcHeuristicDW, [181](#)
  - CbcHeuristicFPump, [195](#)
  - CbcHeuristicGreedyCover, [202](#)
  - CbcHeuristicGreedyEquality, [205](#)
  - CbcHeuristicGreedySOS, [208](#)
  - CbcHeuristicJustOne, [211](#)
  - CbcHeuristicLocal, [214](#)
  - CbcHeuristicNaive, [216](#)
  - CbcHeuristicPartial, [220](#)
  - CbcHeuristicPivotAndFix, [222](#)
  - CbcHeuristicProximity, [224](#)
  - CbcHeuristicRandRound, [227](#)

- CbcHeuristicRINS, [231](#)
- CbcHeuristicVND, [234](#)
- CbcModel, [310](#)
- CbcRounding, [358](#)
- CbcSerendipity, [361](#)
- CbcStrategy, [405](#)
- CbcStrategyDefault, [407](#)
- CbcTree, [418](#)
- CbcTreeLocal, [423](#)
- CbcTreeVariable, [427](#)
- generateCuts
  - CbcCutGenerator, [83](#)
  - CglTemporary, [432](#)
- generateTuning
  - CbcCutGenerator, [83](#)
- generator
  - CbcCutGenerator, [86](#)
- getAllowableFractionGap
  - CbcModel, [279](#)
- getAllowableGap
  - CbcModel, [279](#)
- getAllowablePercentageGap
  - CbcModel, [279](#)
- getApplicationData
  - CbcModel, [298](#)
- getBestCriterion
  - CbcBranchDecision, [35](#)
  - CbcBranchDefaultDecision, [37](#)
  - CbcBranchDynamicDecision, [39](#)
- getBestPossible
  - CbcCompareDefault, [67](#)
- getBestPossibleObjValue
  - CbcModel, [290](#)
- getBestPossibleObjective
  - CbcTree, [419](#)
- getCbcColLower
  - CbcModel, [288](#)
- getCbcColSolution
  - CbcModel, [288](#)
- getCbcColUpper
  - CbcModel, [288](#)
- getCbcReducedCost
  - CbcModel, [288](#)
- getCbcRowActivity
  - CbcModel, [288](#)
- getCbcRowLower
  - CbcModel, [288](#)
- getCbcRowPrice
  - CbcModel, [288](#)
- getCbcRowUpper
  - CbcModel, [288](#)
- getClique
  - CbcGenCtlBlk, [130](#)
- getCoefficients
  - OsiBiLinear, [449](#)
- getColLower
  - CbcModel, [286](#)
  - OsiCbcSolverInterface, [463](#)
- getColName
  - OsiCbcSolverInterface, [466](#)
- getColNames
  - OsiCbcSolverInterface, [466](#)
- getColSolution
  - CbcModel, [290](#)
  - OsiCbcSolverInterface, [465](#)
- getColUpper
  - CbcModel, [286](#)
  - OsiCbcSolverInterface, [463](#)
- getCombine
  - CbcGenCtlBlk, [132](#)
- getContinuousInfeasibilities
  - CbcModel, [292](#)
- getContinuousObjective
  - CbcModel, [292](#)
- getCurrentMinimizationObjValue
  - CbcModel, [290](#)
- getCurrentNumberNeeded
  - CbcHeuristicDW, [184](#)
- getCurrentNumberNodes
  - CbcHeuristicDW, [184](#)
- getCurrentObjValue
  - CbcModel, [290](#)
- getCurrentPassNumber
  - CbcModel, [280](#)
- getCurrentSeconds
  - CbcModel, [278](#)
- getCutDepth
  - CbcGenCtlBlk, [130](#)
- getCutoff
  - CbcCompareDefault, [67](#)
  - CbcModel, [277](#)
  - OsiCbcSolverInterface, [471](#)
- getCutoffIncrement
  - CbcModel, [280](#)
- getDbIParam
  - CbcModel, [277](#)
  - OsiCbcSolverInterface, [462](#)
- getDualRays
  - OsiCbcSolverInterface, [465](#)
- getEmptyBasis
  - CbcModel, [306](#)
- getEmptyWarmStart
  - OsiCbcSolverInterface, [463](#)
- getEventHandler
  - CbcModel, [298](#)
- getExtraNodeCount
  - CbcModel, [284](#)
- getFPump

- CbcGenCtlBlk, 131
- getFlow
  - CbcGenCtlBlk, 130
- getGomory
  - CbcGenCtlBlk, 131
- getGreedyCover
  - CbcGenCtlBlk, 132
- getGreedyEquality
  - CbcGenCtlBlk, 132
- getHeuristicFractionGap
  - CbcModel, 279
- getHeuristicGap
  - CbcModel, 279
- getHintParam
  - OsiCbcSolverInterface, 462
- getIPPAAction
  - CbcGenCtlBlk, 130
- getInfeasibilityWeight
  - CbcModel, 278
- getInfinity
  - CbcModel, 288
  - OsiCbcSolverInterface, 465
- getIntParam
  - CbcModel, 277
  - OsiCbcSolverInterface, 462
- getIntegerInformation
  - CbcModel.hpp, 532
- getIntegerTolerance
  - CbcModel, 278
- getIterationCount
  - CbcModel, 284
  - OsiCbcSolverInterface, 465
- getKnapsack
  - CbcGenCtlBlk, 131
- getMIPStart
  - CbcModel, 310
- getMatrixByCol
  - CbcModel, 287
  - OsiCbcSolverInterface, 465
- getMatrixByRow
  - CbcModel, 287
  - OsiCbcSolverInterface, 464
- getMaximumBranching
  - CbcTree, 420
- getMaximumCutPasses
  - CbcModel, 280
- getMaximumCutPassesAtRoot
  - CbcModel, 280
- getMaximumNodes
  - CbcModel, 277
  - OsiCbcSolverInterface, 471
- getMaximumSeconds
  - CbcModel, 278
  - OsiCbcSolverInterface, 472
- getMaximumSolutions
  - CbcModel, 277
  - OsiCbcSolverInterface, 472
- getMinimizationObjValue
  - CbcModel, 290
- getMinimumDrop
  - CbcModel, 280
- getMir
  - CbcGenCtlBlk, 131
- getModel
  - CbcEventHandler, 102
- getModelPtr
  - OsiCbcSolverInterface, 471
- getMovement
  - OsiBiLinear, 449
- getMultipleRootTries
  - CbcModel, 300
- getNested
  - CbcStrategy, 404
- getNodeCount
  - CbcModel, 284
  - OsiCbcSolverInterface, 472
- getNodeCount2
  - CbcModel, 305
- getNumCols
  - CbcModel, 285
  - OsiCbcSolverInterface, 463
- getNumElements
  - CbcModel, 285
  - OsiCbcSolverInterface, 463
- getNumRows
  - CbcModel, 285
  - OsiCbcSolverInterface, 463
- getNumberBranching
  - CbcTree, 420
- getNumberHeuristicSolutions
  - CbcModel, 292
- getNumberNeeded
  - CbcHeuristicDW, 183
- getNumberNodes
  - CbcHeuristicDW, 184
- getNumberThreads
  - CbcModel, 304
- getObjCoefficients
  - CbcModel, 287
  - OsiCbcSolverInterface, 464
- getObjName
  - OsiCbcSolverInterface, 466
- getObjSense
  - CbcModel, 287
  - OsiCbcSolverInterface, 464
- getObjValue
  - CbcModel, 290
  - OsiCbcSolverInterface, 465



- getPreferredWay
  - CbcModel, [281](#)
- getPrimalRays
  - OsiCbcSolverInterface, [465](#)
- getPrintingMode
  - CbcModel, [278](#)
- getProbing
  - CbcGenCtIBlk, [130](#)
- getPseudoShadow
  - OsiBiLinear, [449](#)
- getRandomSeed
  - CbcModel, [300](#)
- getRealSolverPtr
  - OsiCbcSolverInterface, [471](#)
- getRedSplit
  - CbcGenCtIBlk, [131](#)
- getReducedCost
  - CbcModel, [290](#)
  - OsiCbcSolverInterface, [465](#)
- getRightHandSide
  - CbcModel, [286](#)
  - OsiCbcSolverInterface, [464](#)
- getRounding
  - CbcGenCtIBlk, [132](#)
- getRowActivity
  - CbcModel, [290](#)
  - OsiCbcSolverInterface, [465](#)
- getRowLower
  - CbcModel, [286](#)
  - OsiCbcSolverInterface, [464](#)
- getRowName
  - OsiCbcSolverInterface, [466](#)
- getRowNames
  - OsiCbcSolverInterface, [466](#)
- getRowPrice
  - CbcModel, [290](#)
  - OsiCbcSolverInterface, [465](#)
- getRowRange
  - CbcModel, [286](#)
  - OsiCbcSolverInterface, [464](#)
- getRowSense
  - CbcModel, [286](#)
  - OsiCbcSolverInterface, [464](#)
- getRowUpper
  - CbcModel, [287](#)
  - OsiCbcSolverInterface, [464](#)
- getSeed
  - CbcHeuristic, [151](#)
- getSolutionCount
  - CbcModel, [291](#)
- getSolverObjValue
  - CbcModel, [291](#)
- getState
  - CbcNode, [317](#)
- getStopNumberIterations
  - CbcModel, [293](#)
- getStrParam
  - OsiCbcSolverInterface, [462](#)
- getThreadMode
  - CbcModel, [304](#)
- getTreeLocal
  - CbcGenCtIBlk, [132](#)
- getTwomir
  - CbcGenCtIBlk, [131](#)
- getWarmStart
  - OsiCbcSolverInterface, [463](#)
- getWeight
  - CbcCompareDefault, [67](#)
- globalCuts
  - CbcCutGenerator, [89](#)
  - CbcModel, [308](#)
- globalCutsAtRoot
  - CbcCutGenerator, [89](#)
- goToDantzig
  - CbcModel, [302](#)
- goodModel\_
  - CbcGenCtIBlk, [136](#)
- gradient
  - ClpAmplObjective, [435](#)
  - ClpConstraintAmpl, [438](#)
- guessedObjectiveValue
  - CbcNode, [316](#)
- gutsOfConstructor
  - CbcHeuristicGreedyCover, [203](#)
  - CbcHeuristicGreedyEquality, [206](#)
  - CbcHeuristicGreedySOS, [209](#)
- gutsOfCopy
  - CbcModel, [303](#)
  - OsiSolverLink, [498](#)
- gutsOfDestructor
  - CbcModel, [303](#)
  - OsiSolverLink, [498](#)
- gutsOfDestructor2
  - CbcModel, [303](#)
- gutsOfFollowOn
  - CbcFollowOn, [117](#)
- H
- HELP
  - CbcGenParam, [141](#)
- HEURISTICSTRATEGY
  - CbcGenParam, [141](#)
- haveAnswer\_
  - CbcGenCtIBlk::babState\_struct, [27](#)
- haveMultiThreadSupport
  - CbcModel, [304](#)
- heuristic
  - CbcModel, [297](#)

- heuristicPass
  - CbcEventHandler, [101](#)
- heuristicSolution
  - CbcEventHandler, [101](#)
- heuristic\_
  - CbcHeuristicJustOne, [212](#)
- heuristicCallBack
  - CbcHeuristicDW, [181](#)
- heuristicModel
  - CbcModel, [293](#)
- heuristicName
  - CbcHeuristic, [151](#)
- heuristicName\_
  - CbcHeuristic, [153](#)
- heuristicSolution
  - OsiSolverLink, [495](#)
- hotstartPriorities
  - CbcModel, [307](#)
- hotstartSolution
  - CbcModel, [307](#)
- howOften
  - CbcCutGenerator, [83](#)
- howOften\_
  - CbcHeuristic, [153](#)
  - CbcHeuristicDINS, [160](#)
  - CbcHeuristicDW, [188](#)
  - CbcHeuristicRINS, [232](#)
  - CbcHeuristicVND, [235](#)
- howOftenGlobalScan
  - CbcModel, [283](#)
- howOftenInSub
  - CbcCutGenerator, [84](#)
- howOftenShallow\_
  - CbcHeuristic, [154](#)
- I
- IDIOT
  - CbcOsiParam, [343](#)
- IMPORT
  - CbcGenParam, [141](#)
- INCREMENT
  - CbcCbcParam, [52](#)
- INFEASIBILITYWEIGHT
  - CbcCbcParam, [52](#)
- INTEGERTOLERANCE
  - CbcCbcParam, [52](#)
- INTPRINT
  - CbcGenParam, [141](#)
- IPPEqual
  - CbcGenCtlBlk, [126](#)
- IPPEqualAll
  - CbcGenCtlBlk, [127](#)
- IPPOff
  - CbcGenCtlBlk, [126](#)
- IPPOn
  - CbcGenCtlBlk, [126](#)
- IPPSOS
  - CbcGenCtlBlk, [126](#)
- IPPSave
  - CbcGenCtlBlk, [126](#)
- IPPStrategy
  - CbcGenCtlBlk, [127](#)
- IPPTrySOS
  - CbcGenCtlBlk, [127](#)
- INFEAS
  - CbcSimpleIntegerDynamicPseudoCost.hpp, [539](#)
- IPPControl
  - CbcGenCtlBlk, [126](#)
- id
  - CbcObject, [337](#)
- id\_
  - CbcObject, [338](#)
  - CbcStatistics, [400](#)
- importData
  - CbcUser, [430](#)
- improvement
  - OsiBiLinearEquality, [454](#)
- inaccuracy
  - CbcCutGenerator, [84](#)
- increaseSpace
  - CbcTree, [420](#)
- increment
  - CbcCountRowCut, [75](#)
  - CbcNodeInfo, [322](#)
- increment\_
  - CbcHeuristicProximity, [225](#)
- incrementCuts
  - CbcNodeInfo, [324](#)
- incrementExtra
  - CbcModel, [309](#)
- incrementIterationCount
  - CbcModel, [284](#)
- incrementNodeCount
  - CbcModel, [284](#)
- incrementNumberBeforeTrust
  - CbcSimpleIntegerDynamicPseudoCost, [377](#)
- incrementNumberColumnCuts
  - CbcCutGenerator, [87](#)
- incrementNumberCutsActive
  - CbcCutGenerator, [87](#)
- incrementNumberCutsInTotal
  - CbcCutGenerator, [86](#)
- incrementNumberElementsInTotal
  - CbcCutGenerator, [86](#)
- incrementNumberPointingToThis
  - CbcNodeInfo, [323](#)
- incrementNumberSolutionsFound
  - CbcHeuristic, [150](#)

- incrementNumberTimesDown
  - CbcSimpleIntegerDynamicPseudoCost, [376](#)
- incrementNumberTimesDownInfeasible
  - CbcSimpleIntegerDynamicPseudoCost, [376](#)
- incrementNumberTimesEntered
  - CbcCutGenerator, [86](#)
- incrementNumberTimesUp
  - CbcSimpleIntegerDynamicPseudoCost, [376](#)
- incrementNumberTimesUpInfeasible
  - CbcSimpleIntegerDynamicPseudoCost, [377](#)
- incrementParentCuts
  - CbcNodeInfo, [324](#)
- incrementStrongInfo
  - CbcModel, [309](#)
- incrementSubTreeStopped
  - CbcModel, [294](#)
- incrementTimeInCutGenerator
  - CbcCutGenerator, [85](#)
- incrementUsed
  - CbcModel, [289](#)
- index
  - CoinHashLink, [439](#)
- indexNumber
  - CbcParam, [353](#)
- indices\_
  - CbcFathomDynamicProgramming, [109](#)
- ineffectualCuts
  - CbcCutGenerator, [88](#)
- infeasibility
  - CbcBranchAllDifferent, [29](#)
  - CbcBranchCut, [31](#)
  - CbcBranchToFixLots, [49](#)
  - CbcCliques, [56](#)
  - CbcFollowOn, [117](#)
  - CbcGeneral, [138](#)
  - CbcIdiotBranch, [238](#)
  - CbcLotsize, [249](#)
  - CbcNWay, [328](#)
  - CbcObject, [335](#)
  - CbcSimpleInteger, [364](#)
  - CbcSimpleIntegerDynamicPseudoCost, [372](#)
  - CbcSimpleIntegerPseudoCost, [383](#)
  - CbcSOS, [392](#)
  - OsiBiLinear, [445](#)
  - OsiLink, [477](#)
  - OsiOldLink, [482](#)
  - OsiSimpleFixedInteger, [488](#)
  - OsiUsesBiLinear, [502](#)
- info\_
  - OsiSolverLink, [499](#)
- initialLower
  - CbcHeuristicDW, [184](#)
- initialSolve
  - CbcModel, [273](#)
- OsiCbcSolverInterface, [462](#)
- OsiSolverLinearizedQuadratic, [490](#)
- OsiSolverLink, [495](#)
- initialUpper
  - CbcHeuristicDW, [185](#)
- initialWeight
  - CbcHeuristicFPump, [197](#)
- initialWeight\_
  - CbcHeuristicFPump, [199](#)
- initialize
  - CbcBranchDecision, [34](#)
  - CbcBranchDefaultDecision, [37](#)
  - CbcBranchDynamicDecision, [39](#)
- initializeData
  - CbcHeuristicDive, [165](#)
  - CbcHeuristicDivePseudoCost, [174](#)
- initializeForBranching
  - CbcIdiotBranch, [238](#)
  - CbcObject, [338](#)
- initializeInfo
  - CbcNode, [315](#)
  - CbcNodeInfo, [322](#)
- inputSolution\_
  - CbcHeuristic, [155](#)
- intArray\_
  - CbcHeuristicDW, [186](#)
- intArrays
  - CbcHeuristicDW, [185](#)
- intDecrease\_
  - CbcObjectUpdateData, [340](#)
- intParameter
  - CbcParam, [351](#)
- intValue
  - CbcParam, [352](#)
  - CbcSolver, [387](#)
- integerPresolve
  - CbcModel, [275](#)
- integerPresolveThisModel
  - CbcModel, [275](#)
- integerPriority
  - OsiSolverLink, [497](#)
- integerPriority\_
  - OsiSolverLink, [500](#)
- integerType
  - CbcModel, [285](#), [286](#)
- integerVariable
  - CbcModel, [285](#)
- intsInBlock
  - CbcHeuristicDW, [185](#)
- intsInBlock\_
  - CbcHeuristicDW, [188](#)
- isAbandoned
  - CbcModel, [283](#)
  - OsiCbcSolverInterface, [462](#)

- isBinary
  - CbcModel, [287](#)
- isContinuous
  - CbcModel, [287](#)
  - OsiCbcSolverInterface, [464](#)
- isContinuousUnbounded
  - CbcModel, [283](#)
- isDualObjectiveLimitReached
  - OsiCbcSolverInterface, [463](#)
- isFreeBinary
  - CbcModel, [287](#)
- isInitialSolveAbandoned
  - CbcModel, [285](#)
- isInitialSolveProvenDualInfeasible
  - CbcModel, [285](#)
- isInitialSolveProvenOptimal
  - CbcModel, [285](#)
- isInitialSolveProvenPrimalInfeasible
  - CbcModel, [285](#)
- isInteger
  - CbcModel, [287](#)
- isIntegerNonBinary
  - CbcModel, [287](#)
- isIterationLimitReached
  - OsiCbcSolverInterface, [463](#)
- isLocked
  - CbcModel, [304](#)
- isNodeLimitReached
  - CbcModel, [283](#)
  - OsiCbcSolverInterface, [472](#)
- isPrimalObjectiveLimitReached
  - OsiCbcSolverInterface, [462](#)
- isProvenDualInfeasible
  - CbcModel, [283](#)
  - OsiCbcSolverInterface, [462](#)
- isProvenInfeasible
  - CbcModel, [283](#)
- isProvenOptimal
  - CbcModel, [283](#)
  - OsiCbcSolverInterface, [462](#)
- isProvenPrimalInfeasible
  - OsiCbcSolverInterface, [462](#)
- isSecondsLimitReached
  - CbcModel, [284](#)
- isSolutionLimitReached
  - CbcModel, [284](#)
  - OsiCbcSolverInterface, [472](#)
- iterationRatio
  - CbcHeuristicFPump, [197](#)
- iterationRatio\_
  - CbcHeuristicFPump, [200](#)
- K
- KEEPNAMES
  - CbcOsiParam, [343](#)
- KKT
  - CbcOsiParam, [343](#)
- KNAPSACKCUTS
  - CbcGenParam, [141](#)
- k\_
  - CbcHeuristicVND, [236](#)
- keepContinuous\_
  - CbcHeuristicDW, [189](#)
- killSolution
  - CbcEventHandler, [101](#)
- kmax\_
  - CbcHeuristicVND, [236](#)
- L
- LOCALTREE
  - CbcGenParam, [141](#)
- LOGLEVEL
  - CbcCbcParam, [52](#)
  - CbcGenParam, [141](#)
- large\_
  - CbcHeuristicNaive, [217](#)
- largeValue
  - CbcHeuristicNaive, [217](#)
- last\_
  - CbcGenCtlBlk::cbcParamsInfo\_struct, [353](#)
  - CbcGenCtlBlk::genParamsInfo\_struct, [441](#)
  - CbcGenCtlBlk::osiParamsInfo\_struct, [486](#)
- lastDepth
  - CbcTree, [420](#)
- lastDepth\_
  - CbcTree, [421](#)
- lastDownCost\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- lastDownDecrease\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- lastHeuristic
  - CbcModel, [297](#)
- lastMpsIn\_
  - CbcGenCtlBlk, [134](#)
- lastNode\_
  - CbcHeuristicRINS, [232](#)
  - CbcHeuristicVND, [235](#)
- lastObjective
  - CbcTree, [420](#)
- lastObjective\_
  - CbcHeuristicDW, [186](#)
  - CbcTree, [421](#)
- lastRunDeep\_
  - CbcHeuristic, [154](#)
- lastSolnOut\_
  - CbcGenCtlBlk, [135](#)
- lastUnsatisfied
  - CbcTree, [420](#)

- lastUnsatisfied\_
  - CbcTree, [421](#)
- lastUpCost\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- lastUpDecrease\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- linearObjective
  - ClpAmplObjective, [436](#)
- linearizedBAB
  - OsiSolverLink, [495](#)
- load
  - OsiSolverLink, [495](#)
- loadCbcParamObj
  - CbcCbcParamUtils, [20](#)
- loadGenParamObj
  - CbcGenParamUtils, [21](#)
- loadOsiParamObj
  - CbcOsiParamUtils, [22](#)
- loadProblem
  - OsiCbcSolverInterface, [469](#), [470](#)
- localSpace\_
  - CbcHeuristicDINS, [161](#)
- lockThread
  - CbcModel, [304](#)
- logLevel
  - ampl\_info, [26](#)
  - CbcGenCtlBlk, [134](#)
  - CbcModel, [299](#)
- LongCliqueBranchObj
  - CbcBranchingObject.hpp, [512](#)
- lookup\_
  - CbcFathomDynamicProgramming, [109](#)
- LotsizeBranchObj
  - CbcBranchingObject.hpp, [512](#)
- lower
  - CbcFullNodeInfo, [120](#)
- lower\_
  - CbcFullNodeInfo, [121](#)
- M
- MAXHOTITS
  - CbcOsiParam, [343](#)
- MAXIMIZE
  - CbcCbcParam, [52](#)
- MAXITERATION
  - CbcOsiParam, [343](#)
- MAXNODES
  - CbcCbcParam, [52](#)
- MESSAGES
  - CbcGenParam, [141](#)
- MINIMIZE
  - CbcCbcParam, [52](#)
- MIPLIB
  - CbcGenParam, [141](#)
- MIPOPTIONS
  - CbcCbcParam, [52](#)
- MIXEDCUTS
  - CbcGenParam, [141](#)
- MOREMIPOPTIONS
  - CbcCbcParam, [52](#)
- MINIMUM\_MOVEMENT
  - CbcSimpleIntegerDynamicPseudoCost.hpp, [539](#)
- MOD\_SHADOW
  - CbcSimpleIntegerDynamicPseudoCost.hpp, [539](#)
- majorStatus\_
  - CbcGenCtlBlk::babState\_struct, [27](#)
- makeGlobalCut
  - CbcModel, [274](#)
- makeGlobalCuts
  - CbcModel, [274](#)
- makePartialCut
  - CbcModel, [274](#)
- mark
  - CbcNodeInfo, [325](#)
- mark\_
  - CbcBranchToFixLots, [49](#)
- markHotStart
  - OsiCbcSolverInterface, [463](#)
- markNonlinear
  - ClpAmplObjective, [436](#)
  - ClpConstraintAmpl, [438](#)
- markNonzero
  - ClpConstraintAmpl, [438](#)
- marked
  - CbcNodeInfo, [325](#)
- masterThread
  - CbcModel, [304](#)
- matchName
  - CbcParam, [351](#)
- matches
  - CbcParam, [352](#)
- matrix\_
  - CbcFollowOn, [117](#)
  - CbcHeuristicDive, [165](#)
  - CbcHeuristicGreedyCover, [203](#)
  - CbcHeuristicGreedyEquality, [206](#)
  - CbcHeuristicGreedySOS, [209](#)
  - CbcHeuristicLocal, [214](#)
  - CbcRounding, [359](#)
  - OsiSolverLink, [498](#)
- matrixByRow\_
  - CbcBranchToFixLots, [49](#)
  - CbcFollowOn, [117](#)
  - CbcHeuristicDive, [165](#)
  - CbcRounding, [359](#)
- maxDiversification
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [428](#)

- maxIterations\_
  - CbcHeuristicDive, 166
- maxSimplexIterations
  - CbcHeuristicDive, 164
- maxSimplexIterations\_
  - CbcHeuristicDive, 166
- maxSimplexIterationsAtRoot\_
  - CbcHeuristicDive, 166
- maxTime\_
  - CbcHeuristicDive, 166
- maximumBranching\_
  - CbcTree, 421
- maximumDW\_
  - CbcHeuristicDW, 189
- maximumKeepSolutions\_
  - CbcHeuristicDINS, 160
- maximumNodeNumber
  - CbcTree, 419
- maximumNodeNumber\_
  - CbcTree, 421
- maximumNumberIterations
  - CbcModel, 308
- maximumPasses
  - CbcHeuristicFPump, 198
- maximumPasses\_
  - CbcHeuristicFPump, 200
- maximumRetries
  - CbcHeuristicFPump, 198
- maximumRetries\_
  - CbcHeuristicFPump, 200
- maximumRows
  - CbcModel, 293
- maximumSavedSolutions
  - CbcModel, 291
- maximumSecondsReached
  - CbcModel, 278
- maximumSize
  - CbcFathomDynamicProgramming, 107
- maximumSizeAllowed\_
  - CbcFathomDynamicProgramming, 109
- maximumTime
  - CbcHeuristicFPump, 196
- maximumTime\_
  - CbcHeuristicFPump, 199
- maximumTries
  - CbcCutGenerator, 84
- members
  - CbcClique, 57
  - CbcNWay, 329
  - CbcSOS, 393
- members\_
  - CbcClique, 57
  - CbcNWay, 329
- mergeModels
  - CbcModel, 305
- message
  - CbcGenCtlBlk, 133
- messageHandler
  - CbcGenCtlBlk, 134
  - CbcModel, 299
- messages
  - CbcModel, 299
- messagesPointer
  - CbcModel, 299
- method
  - CbcSimpleIntegerDynamicPseudoCost, 377
  - CbcSimpleIntegerPseudoCost, 383
- method\_
  - CbcSimpleIntegerDynamicPseudoCost, 380
  - CbcSimpleIntegerPseudoCost, 384
- minDistance
  - CbcHeuristicNode, 218
- minDistanceIsSmall
  - CbcHeuristicNode, 218
- minDistanceToRun\_
  - CbcHeuristic, 154
- minorStatus\_
  - CbcGenCtlBlk::babState\_struct, 27
- model
  - CbcBranchingObject, 45
  - CbcObject, 338
  - CbcSolver, 388
- model\_
  - CbcBranchDecision, 35
  - CbcBranchingObject, 46
  - CbcEventHandler, 102
  - CbcFathom, 104
  - CbcGenCtlBlk, 136
  - CbcHeuristic, 152
  - CbcObject, 338
- modelOwnsSolver
  - CbcModel, 302
- modelPtr\_
  - OsiCbcSolverInterface, 473
- modelSequence
  - CbcLotsize, 250
- modifiableBranchingObject
  - CbcNode, 317
- modifiableObject
  - CbcModel, 276
- modify
  - CbcCutModifier, 91
  - CbcCutSubsetModifier, 93
- moreSpecialOptions
  - CbcModel, 301
- moreSpecialOptions2
  - CbcModel, 301
- moveInfo

- CbcModel, [303](#)
- moveToModel
  - CbcModel, [305](#)
- multiplier\_
  - OsiBiLinear, [451](#)
- mustCallAgain
  - CbcCutGenerator, [87](#)
- mutableLower
  - CbcFullNodeInfo, [120](#)
- mutableMembers
  - CbcSOS, [394](#)
- mutableOwner
  - CbcNodeInfo, [324](#)
- mutableStrongInfo
  - CbcModel, [309](#)
- mutableUpper
  - CbcFullNodeInfo, [121](#)
- mutableWeights
  - CbcSOS, [394](#)
- N
- NETLIB\_BARRIER
  - CbcOsiParam, [343](#)
- NETLIB\_DUAL
  - CbcOsiParam, [343](#)
- NETLIB\_PRIMAL
  - CbcOsiParam, [343](#)
- NETWORK
  - CbcOsiParam, [343](#)
- NUMBERANALYZE
  - CbcCbcParam, [52](#)
- NUMBERBEFORE
  - CbcCbcParam, [52](#)
- NUMBERMINI
  - CbcCbcParam, [52](#)
- NWayBranchObj
  - CbcBranchingObject.hpp, [512](#)
- nDifferent\_
  - CbcHeuristicVND, [236](#)
- nNeeded\_
  - CbcHeuristicDW, [190](#)
- nNeededBase\_
  - CbcHeuristicDW, [190](#)
- nNodes\_
  - CbcHeuristicDW, [190](#)
- nNodesBase\_
  - CbcHeuristicDW, [190](#)
- name
  - CbcParam, [350](#)
  - CbcUser, [431](#)
- needsOptimalBasis
  - CbcCutGenerator, [87](#)
- newBound\_
  - CbcFixVariable, [115](#)
- CbcTree, [421](#)
- newBounds
  - CbcPartialNodeInfo, [356](#)
  - CbcTree, [420](#)
  - OsiBiLinear, [448](#)
- newBounds\_
  - CbcPartialNodeInfo, [356](#)
- newGrid
  - OsiBiLinearEquality, [454](#)
- newLanguage
  - CbcModel, [299](#)
  - OsiCbcSolverInterface, [471](#)
- newSolution
  - CbcCompareBase, [63](#)
  - CbcCompareDefault, [67](#)
- newXValues
  - ClpAmplObjective, [436](#)
  - ClpConstraintAmpl, [438](#)
- next
  - CoinHashLink, [439](#)
- noAction
  - CbcEventHandler, [101](#)
- node
  - CbcEventHandler, [101](#)
  - CbcHeuristicNodeList, [219](#)
  - CbcStatistics, [399](#)
- nodeComparison
  - CbcModel, [294](#)
- nodeInfo
  - CbcNode, [315](#)
- nodeLimit
  - CbcTreeLocal, [425](#)
  - CbcTreeVariable, [428](#)
- nodeNumber
  - CbcNode, [317](#)
  - CbcNodeInfo, [324](#)
- nodeNumber\_
  - CbcNodeInfo, [326](#)
- nodePointer
  - CbcTree, [419](#)
- nodes\_
  - CbcTree, [421](#)
- nonLinear
  - ampl\_info, [26](#)
- nonlinearSLP
  - OsiSolverLink, [495](#)
- normal
  - CbcCutGenerator, [85](#)
- normalSolver
  - CbcModel, [300](#)
- normalizeProbabilities
  - CbcHeuristicJustOne, [212](#)
- notPreferredNewFeasible
  - CbcBranchCut, [32](#)

- CbcLotsize, 250
  - CbcObject, 336
- nullNodeInfo
  - CbcNode, 315
- nullOwner
  - CbcNodeInfo, 324
- nullParent
  - CbcNodeInfo, 323
- numBeforeTrust\_
  - CbcGenCtlBlk::chooseStrongCtl\_struct, 433
- numCols\_
  - CbcGenCtlBlk::debugSolInfo\_struct, 440
- numCouldRun
  - CbcHeuristic, 152
- numCouldRun\_
  - CbcHeuristic, 154
- numIntInfeasDown
  - CbcStrongInfo, 413
- numIntInfeasUp
  - CbcStrongInfo, 413
- numInvocationsInDeep\_
  - CbcHeuristic, 154
- numInvocationsInShallow\_
  - CbcHeuristic, 154
- numItersDown
  - CbcStrongInfo, 414
- numItersUp
  - CbcStrongInfo, 413
- numObjInfeasDown
  - CbcStrongInfo, 413
- numObjInfeasUp
  - CbcStrongInfo, 413
- numRuns
  - CbcHeuristic, 152
- numRuns\_
  - CbcHeuristic, 154
- numStrong\_
  - CbcGenCtlBlk::chooseStrongCtl\_struct, 433
- numberActive\_
  - CbcFathomDynamicProgramming, 109
- numberActiveCutsAtRoot
  - CbcCutGenerator, 89
- numberAnalyzeIterations
  - CbcModel, 282
- numberArguments
  - ampl\_info, 24
- numberBadPasses\_
  - CbcHeuristicDW, 190
- numberBeforeTrust
  - CbcModel, 281
  - CbcSimpleIntegerDynamicPseudoCost, 377
- numberBeforeTrust\_
  - CbcSimpleIntegerDynamicPseudoCost, 380
  - CbcStrategyDefault, 408
- CbcStrategyDefaultSubTree, 410
- numberBiLinear\_
  - OsiUsesBiLinear, 503
- numberBinary
  - ampl\_info, 24
- numberBits\_
  - CbcFathomDynamicProgramming, 109
- numberBlocks
  - CbcHeuristicDW, 182
- numberBlocks\_
  - CbcHeuristicDW, 189
- numberBranches
  - CbcNode, 316
  - CbcNWayBranchingObject, 331
- numberBranchesLeft
  - CbcNodeInfo, 322
- numberBranchesLeft\_
  - CbcNodeInfo, 326
- numberBranching\_
  - CbcTree, 421
- numberChangedBounds
  - CbcPartialNodeInfo, 356
- numberChangedBounds\_
  - CbcPartialNodeInfo, 356
- numberClean\_
  - CbcBranchToFixLots, 50
- numberCoefficients
  - ClpConstraintAmpl, 438
- numberColumnCuts
  - CbcCutGenerator, 86
- numberColumns
  - ampl\_info, 23
- numberColumnsDW
  - CbcHeuristicDW, 182, 183
- numberColumnsDW\_
  - CbcHeuristicDW, 187
- numberCutGenerators
  - CbcModel, 296
  - CbcSolver, 389
- numberCuts
  - CbcNodeInfo, 324
  - CbcRowCuts, 360
- numberCuts\_
  - CbcNodeInfo, 326
- numberCutsActive
  - CbcCutGenerator, 87
- numberCutsAtRoot
  - CbcCutGenerator, 88
- numberCutsInTotal
  - CbcCutGenerator, 86
- numberDW\_
  - CbcHeuristicDW, 189
- numberDWTimes
  - CbcHeuristicDW, 183



- numberDWTimes\_
  - CbcHeuristicDW, [189](#)
- numberElements
  - ampl\_info, [24](#)
- numberElementsInTotal
  - CbcCutGenerator, [86](#)
- numberExtralutations
  - CbcModel, [309](#)
- numberExtraRows\_
  - OsiBiLinear, [451](#)
- numberFix\_
  - OsiSolverLink, [500](#)
- numberGlobalViolations
  - CbcModel, [293](#)
- numberHeuristics
  - CbcModel, [297](#)
- numberHeuristics\_
  - CbcHeuristicJustOne, [212](#)
- numberInSet\_
  - CbcBranchAllDifferent, [29](#)
- numberIntegers
  - ampl\_info, [24](#)
  - CbcModel, [285](#)
- numberIntegers\_
  - CbcFullNodeInfo, [121](#)
  - CbcHeuristicDINS, [161](#)
- numberIterations
  - CbcStatistics, [399](#)
- numberIterations\_
  - CbcStatistics, [400](#)
- numberKeptSolutions\_
  - CbcHeuristicDINS, [160](#)
- numberLinks
  - OsiLink, [477](#)
  - OsiOldLink, [482](#)
- numberMasterColumns\_
  - CbcHeuristicDW, [189](#)
- numberMasterRows\_
  - CbcHeuristicDW, [189](#)
- numberMembers
  - CbcClique, [56](#)
  - CbcNWay, [329](#)
  - CbcSOS, [393](#)
- numberMembers\_
  - CbcClique, [57](#)
  - CbcNWay, [329](#)
- numberNodes
  - CbcHeuristic, [149](#)
- numberNodes\_
  - CbcHeuristic, [153](#)
- numberNodesDone\_
  - CbcHeuristic, [155](#)
- numberNonLinearRows\_
  - OsiSolverLink, [498](#)
- numberNonOne\_
  - CbcFathomDynamicProgramming, [110](#)
- numberNonSOSMembers
  - CbcClique, [56](#)
- numberNonSOSMembers\_
  - CbcClique, [57](#)
- numberObjects
  - CbcModel, [276](#)
- numberObjectsToUse
  - OsiChooseStrongSubset, [475](#)
- numberObjectsToUse\_
  - OsiChooseStrongSubset, [475](#)
- numberPasses\_
  - CbcHeuristicDW, [188](#)
- numberPenalties
  - CbcModel, [282](#)
- numberPointingToThis
  - CbcCountRowCut, [76](#)
  - CbcNodeInfo, [323](#)
- numberPointingToThis\_
  - CbcNodeInfo, [325](#)
- numberPoints
  - OsiBiLinearEquality, [454](#)
- numberRanges
  - CbcLotsize, [251](#)
- numberRows
  - ampl\_info, [23](#)
- numberRows\_
  - CbcNodeInfo, [326](#)
- numberRowsAtContinuous
  - CbcModel, [285](#)
- numberSavedSolutions
  - CbcModel, [291](#)
- numberShortCutsAtRoot
  - CbcCutGenerator, [89](#)
- numberSolutions\_
  - CbcCompareDefault, [68](#)
  - CbcHeuristicCrossover, [157](#)
  - CbcHeuristicDINS, [160](#)
  - CbcHeuristicLocal, [214](#)
  - CbcHeuristicProximity, [225](#)
  - CbcHeuristicRINS, [232](#)
  - CbcHeuristicVND, [235](#)
- numberSolutionsFound
  - CbcHeuristic, [150](#)
- numberSolutionsFound\_
  - CbcHeuristic, [155](#)
- numberSos
  - ampl\_info, [24](#)
- numberStates\_
  - CbcFixVariable, [115](#)
- numberStoppedSubTrees
  - CbcModel, [294](#)
- numberStrong

- CbcModel, [281](#)
- numberStrong\_
  - CbcStrategyDefault, [408](#)
  - CbcStrategyDefaultSubTree, [410](#)
- numberStrongIterations
  - CbcModel, [308](#)
- numberSuccesses\_
  - CbcHeuristicDINS, [160](#)
  - CbcHeuristicRINS, [232](#)
  - CbcHeuristicVND, [235](#)
- numberTimes
  - CbcHeuristicGreedyCover, [203](#)
  - CbcHeuristicGreedyEquality, [206](#)
  - CbcHeuristicGreedySOS, [209](#)
- numberTimes\_
  - CbcHeuristicGreedyCover, [203](#)
  - CbcHeuristicGreedyEquality, [206](#)
  - CbcHeuristicGreedySOS, [209](#)
- numberTimesDown
  - CbcSimpleIntegerDynamicPseudoCost, [375](#)
  - CbcSOS, [393](#)
- numberTimesDown\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- numberTimesDownInfeasible
  - CbcSimpleIntegerDynamicPseudoCost, [376](#)
- numberTimesDownInfeasible\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- numberTimesDownLocalFixed\_
  - CbcSimpleIntegerDynamicPseudoCost, [380](#)
- numberTimesDownTotalFixed\_
  - CbcSimpleIntegerDynamicPseudoCost, [380](#)
- numberTimesEntered
  - CbcCutGenerator, [86](#)
- numberTimesProbingTotal\_
  - CbcSimpleIntegerDynamicPseudoCost, [380](#)
- numberTimesUp
  - CbcSimpleIntegerDynamicPseudoCost, [376](#)
  - CbcSOS, [394](#)
- numberTimesUp\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- numberTimesUpInfeasible
  - CbcSimpleIntegerDynamicPseudoCost, [376](#)
- numberTimesUpInfeasible\_
  - CbcSimpleIntegerDynamicPseudoCost, [380](#)
- numberTimesUpLocalFixed\_
  - CbcSimpleIntegerDynamicPseudoCost, [380](#)
- numberTimesUpTotalFixed\_
  - CbcSimpleIntegerDynamicPseudoCost, [380](#)
- numberTries\_
  - CbcHeuristicDINS, [160](#)
  - CbcHeuristicRENS, [229](#)
  - CbcHeuristicRINS, [232](#)
  - CbcHeuristicVND, [235](#)
- numberUnsatisfied
  - CbcNode, [316](#)
- numberUserFunctions
  - CbcSolver, [388](#)
- numberVariables\_
  - OsiSolverLink, [499](#)
- O
- OBJSCALE
  - CbcOsiParam, [343](#)
- ODDHOLECUTS
  - CbcGenParam, [141](#)
- OUTDUPROWS
  - CbcGenParam, [141](#)
- OUTPUTFORMAT
  - CbcGenParam, [141](#)
- obj
  - CbcCbcParam, [53](#)
  - CbcGenParam, [143](#)
  - CbcOsiParam, [345](#)
- objValue
  - ampl\_info, [24](#)
- object
  - CbcBranchingObject, [45](#)
  - CbcDynamicPseudoCostBranchingObject, [98](#)
  - CbcModel, [276](#)
- object\_
  - CbcBranchDecision, [35](#)
  - CbcDynamicPseudoCostBranchingObject, [99](#)
  - CbcObjectUpdateData, [340](#)
- objectNumber
  - CbcStrongInfo, [414](#)
- objectNumber\_
  - CbcObjectUpdateData, [340](#)
- objective
  - ampl\_info, [24](#)
- objectiveDW
  - CbcHeuristicDW, [183](#)
- objectiveDW\_
  - CbcHeuristicDW, [187](#)
- objectiveRow\_
  - OsiSolverLink, [499](#)
- objectiveValue
  - CbcHeuristicDW, [185](#)
  - CbcNode, [316](#)
  - ClpAmplObjective, [435](#)
- objectiveValueWhen
  - CbcHeuristicDW, [182](#)
- objectiveVariable
  - OsiSolverLink, [497](#)
- objectiveVariable\_
  - OsiSolverLink, [499](#)
- objects
  - CbcModel, [276](#)
- objects\_

- OsiUsesBiLinear, [503](#)
- offset
  - ampl\_info, [24](#)
- onTree
  - CbcNode, [317](#)
- OneGeneralBranchingObj
  - CbcBranchingObject.hpp, [512](#)
- operator()
  - CbcCompare, [61](#)
  - CbcCompareBase, [64](#)
- operator=
  - CbcBranchAllDifferent, [29](#)
  - CbcBranchCut, [31](#)
  - CbcBranchingObject, [43](#)
  - CbcBranchToFixLots, [48](#)
  - CbcCbcParam, [53](#)
  - CbcClique, [56](#)
  - CbcCliqueBranchingObject, [59](#)
  - CbcCompareBase, [63](#)
  - CbcCompareDefault, [66](#)
  - CbcCompareDepth, [70](#)
  - CbcCompareEstimate, [71](#)
  - CbcCompareObjective, [72](#)
  - CbcConsequence, [73](#)
  - CbcCutBranchingObject, [78](#)
  - CbcCutGenerator, [83](#)
  - CbcCutModifier, [91](#)
  - CbcCutSubsetModifier, [92](#)
  - CbcDummyBranchingObject, [94](#)
  - CbcDynamicPseudoCostBranchingObject, [97](#)
  - CbcEventHandler, [102](#)
  - CbcFeasibilityBase, [111](#)
  - CbcFixingBranchingObject, [112](#)
  - CbcFixVariable, [114](#)
  - CbcFollowOn, [117](#)
  - CbcGeneral, [138](#)
  - CbcGenParam, [143](#)
  - CbcHeuristic, [148](#)
  - CbcHeuristicCrossover, [156](#)
  - CbcHeuristicDINS, [159](#)
  - CbcHeuristicDive, [163](#)
  - CbcHeuristicDiveCoefficient, [167](#)
  - CbcHeuristicDiveFractional, [169](#)
  - CbcHeuristicDiveGuided, [171](#)
  - CbcHeuristicDiveLineSearch, [172](#)
  - CbcHeuristicDivePseudoCost, [174](#)
  - CbcHeuristicDiveVectorLength, [176](#)
  - CbcHeuristicDW, [181](#)
  - CbcHeuristicFPump, [195](#)
  - CbcHeuristicGreedyCover, [202](#)
  - CbcHeuristicGreedyEquality, [205](#)
  - CbcHeuristicGreedySOS, [208](#)
  - CbcHeuristicJustOne, [211](#)
  - CbcHeuristicLocal, [213](#)
  - CbcHeuristicNaive, [216](#)
  - CbcHeuristicNodeList, [218](#)
  - CbcHeuristicPartial, [220](#)
  - CbcHeuristicPivotAndFix, [222](#)
  - CbcHeuristicProximity, [224](#)
  - CbcHeuristicRandRound, [226](#)
  - CbcHeuristicRENS, [228](#)
  - CbcHeuristicRINS, [231](#)
  - CbcHeuristicVND, [234](#)
  - CbcIdiotBranch, [238](#)
  - CbcIntegerBranchingObject, [241](#)
  - CbcIntegerPseudoCostBranchingObject, [244](#)
  - CbcLongCliqueBranchingObject, [247](#)
  - CbcLotsize, [249](#)
  - CbcLotsizeBranchingObject, [253](#)
  - CbcModel, [302](#)
  - CbcNode, [313](#)
  - CbcNWay, [328](#)
  - CbcNWayBranchingObject, [331](#)
  - CbcObject, [335](#)
  - CbcObjectUpdateData, [340](#)
  - CbcOsiParam, [345](#)
  - CbcOsiSolver, [347](#)
  - CbcParam, [350](#)
  - CbcRounding, [358](#)
  - CbcRowCuts, [360](#)
  - CbcSerendipity, [361](#)
  - CbcSimpleInteger, [364](#)
  - CbcSimpleIntegerDynamicPseudoCost, [372](#)
  - CbcSimpleIntegerPseudoCost, [382](#)
  - CbcSolver, [387](#)
  - CbcSOS, [392](#)
  - CbcSOSBranchingObject, [396](#)
  - CbcStatistics, [398](#)
  - CbcStopNow, [402](#)
  - CbcTree, [418](#)
  - CbcTreeLocal, [423](#)
  - CbcTreeVariable, [426](#)
  - CbcUser, [431](#)
  - CglTemporary, [433](#)
  - ClpAmplObjective, [436](#)
  - ClpConstraintAmpl, [438](#)
  - OsiBiLinear, [445](#)
  - OsiBiLinearBranchingObject, [452](#)
  - OsiBiLinearEquality, [454](#)
  - OsiCbcSolverInterface, [473](#)
  - OsiChooseStrongSubset, [475](#)
  - OsiLink, [477](#)
  - OsiLinkBranchingObject, [478](#)
  - OsiLinkedBound, [480](#)
  - OsiOldLink, [482](#)
  - OsiOldLinkBranchingObject, [484](#)
  - OsiOneLink, [485](#)
  - OsiSimpleFixedInteger, [488](#)

- OsiSolverLinearizedQuadratic, 490
- OsiSolverLink, 495
- OsiUsesBiLinear, 502
- optionalObject
  - CbcObject, 337
- originalCbcObject\_
  - CbcBranchingObject, 46
- originalCoinModel
  - CbcSolver, 388
- originalColumns
  - CbcModel, 283
- originalLower\_
  - CbcSimpleInteger, 366
- originalLowerBound
  - CbcLotsize, 250
  - CbcSimpleInteger, 365
- originalModel
  - CbcModel, 275
- originalNumberRows\_
  - CbcHeuristicGreedyCover, 203
  - CbcHeuristicGreedyEquality, 206
  - CbcHeuristicGreedySOS, 209
- originalObjective\_
  - CbcObjectUpdateData, 341
- originalRhs\_
  - CbcHeuristicGreedySOS, 209
- originalRowCopy
  - OsiSolverLink, 496
- originalRowCopy\_
  - OsiSolverLink, 498
- originalSolver
  - CbcSolver, 388
- originalUpper\_
  - CbcSimpleInteger, 366
- originalUpperBound
  - CbcLotsize, 251
  - CbcSimpleInteger, 365
- OsiBiLinear, 441
  - ~OsiBiLinear, 445
  - addExtraRow, 449
  - boundBranch, 446
  - boundType, 448
  - boundType\_, 450
  - branchingStrategy, 448
  - branchingStrategy\_, 450
  - canDoHeuristics, 446
  - checkInfeasibility, 446
  - chosen\_, 451
  - clone, 445
  - coefficient, 446
  - coefficient\_, 449
  - computeLambdas, 449
  - convexity\_, 451
  - createBranch, 446
  - extraRow\_, 451
  - feasibleRegion, 445
  - firstLambda, 447
  - firstLambda\_, 450
  - getCoefficients, 449
  - getMovement, 449
  - getPseudoShadow, 449
  - infeasibility, 445
  - multiplier\_, 451
  - newBounds, 448
  - numberExtraRows\_, 451
  - operator=, 445
  - OsiBiLinear, 445
  - OsiBiLinear, 445
  - resetSequenceEtc, 446
  - setBoundType, 448
  - setBranchingStrategy, 448
  - setCoefficient, 447
  - setMeshSizes, 448
  - setXMeshSize, 448
  - setXOtherSatisfied, 447
  - setXSatisfied, 447
  - setXYSatisfied, 448
  - setYMeshSize, 448
  - setYOtherSatisfied, 447
  - setYSatisfied, 447
  - updateCoefficients, 448
  - xColumn, 446
  - xColumn\_, 450
  - xMeshSize, 447
  - xMeshSize\_, 449
  - xOtherSatisfied, 447
  - xOtherSatisfied\_, 450
  - xRow, 446
  - xRow\_, 451
  - xSatisfied, 447
  - xSatisfied\_, 449
  - xyBranchValue\_, 450
  - xyCoefficient, 449
  - xyRow, 446
  - xyRow\_, 451
  - xySatisfied, 448
  - xySatisfied\_, 450
  - yColumn, 446
  - yColumn\_, 450
  - yMeshSize, 448
  - yMeshSize\_, 449
  - yOtherSatisfied, 447
  - yOtherSatisfied\_, 450
  - yRow, 446
  - yRow\_, 451
  - ySatisfied, 447
  - ySatisfied\_, 450
- OsiBiLinearBranchingObject, 452

- ~OsiBiLinearBranchingObject, [452](#)
- boundBranch, [453](#)
- branch, [453](#)
- clone, [452](#)
- operator=, [452](#)
- OsiBiLinearBranchingObject, [452](#)
- OsiBiLinearBranchingObject, [452](#)
- print, [453](#)
- OsiBiLinearEquality, [453](#)
  - ~OsiBiLinearEquality, [454](#)
  - clone, [454](#)
  - improvement, [454](#)
  - newGrid, [454](#)
  - numberPoints, [454](#)
  - operator=, [454](#)
  - OsiBiLinearEquality, [454](#)
  - OsiBiLinearEquality, [454](#)
  - setNumberPoints, [454](#)
- OsiCbcHasNDEBUG
  - OsiCbcSolverInterface.hpp, [546](#)
- OsiCbcInfinity
  - OsiCbcSolverInterface.hpp, [546](#)
- OsiCbcSolverInterface, [455](#)
  - ~OsiCbcSolverInterface, [461](#)
  - addCol, [469](#)
  - addCols, [469](#)
  - addRow, [469](#)
  - addRows, [469](#)
  - applyColCut, [473](#)
  - applyRowCut, [473](#)
  - applyRowCuts, [469](#)
  - assignProblem, [470](#)
  - branchAndBound, [462](#)
  - clone, [472](#)
  - deleteColNames, [467](#)
  - deleteCols, [469](#)
  - deleteRowNames, [466](#)
  - deleteRows, [469](#)
  - dfltRowColName, [466](#)
  - getColLower, [463](#)
  - getColName, [466](#)
  - getColNames, [466](#)
  - getColSolution, [465](#)
  - getColUpper, [463](#)
  - getCutoff, [471](#)
  - getDbfParam, [462](#)
  - getDualRays, [465](#)
  - getEmptyWarmStart, [463](#)
  - getHintParam, [462](#)
  - getInfinity, [465](#)
  - getIntParam, [462](#)
  - getIterationCount, [465](#)
  - getMatrixByCol, [465](#)
  - getMatrixByRow, [464](#)
  - getMaximumNodes, [471](#)
  - getMaximumSeconds, [472](#)
  - getMaximumSolutions, [472](#)
  - getModelPtr, [471](#)
  - getNodeCount, [472](#)
  - getNumCols, [463](#)
  - getNumElements, [463](#)
  - getNumRows, [463](#)
  - getObjCoefficients, [464](#)
  - getObjName, [466](#)
  - getObjSense, [464](#)
  - getObjValue, [465](#)
  - getPrimalRays, [465](#)
  - getRealSolverPtr, [471](#)
  - getReducedCost, [465](#)
  - getRightHandSide, [464](#)
  - getRowActivity, [465](#)
  - getRowLower, [464](#)
  - getRowName, [466](#)
  - getRowNames, [466](#)
  - getRowPrice, [465](#)
  - getRowRange, [464](#)
  - getRowSense, [464](#)
  - getRowUpper, [464](#)
  - getStrParam, [462](#)
  - getWarmStart, [463](#)
  - initialSolve, [462](#)
  - isAbandoned, [462](#)
  - isContinuous, [464](#)
  - isDualObjectiveLimitReached, [463](#)
  - isIterationLimitReached, [463](#)
  - isNodeLimitReached, [472](#)
  - isPrimalObjectiveLimitReached, [462](#)
  - isProvenDualInfeasible, [462](#)
  - isProvenOptimal, [462](#)
  - isProvenPrimalInfeasible, [462](#)
  - isSolutionLimitReached, [472](#)
  - loadProblem, [469](#), [470](#)
  - markHotStart, [463](#)
  - modelPtr\_, [473](#)
  - newLanguage, [471](#)
  - operator=, [473](#)
  - OsiCbcSolverInterface, [461](#)
  - OsiCbcSolverInterfaceUnitTest, [473](#)
  - OsiCbcSolverInterface, [461](#)
  - passInMessageHandler, [472](#)
  - readMps, [470](#)
  - resolve, [462](#)
  - setColBounds, [467](#)
  - setColLower, [467](#)
  - setColName, [466](#)
  - setColNames, [466](#)
  - setColSetBounds, [467](#)
  - setColSolution, [468](#)

- setColUpper, [467](#)
- setContinuous, [468](#)
- setCutoff, [471](#)
- setDbiParam, [462](#)
- setHintParam, [462](#)
- setIntParam, [462](#)
- setInteger, [468](#)
- setLanguage, [471](#)
- setMaximumNodes, [471](#)
- setMaximumSeconds, [472](#)
- setMaximumSolutions, [472](#)
- setObjCoeff, [467](#)
- setObjName, [466](#)
- setObjSense, [468](#)
- setRowBounds, [467](#)
- setRowLower, [467](#)
- setRowName, [466](#)
- setRowNames, [466](#)
- setRowPrice, [469](#)
- setRowSetBounds, [468](#)
- setRowSetTypes, [468](#)
- setRowType, [467](#)
- setRowUpper, [467](#)
- setStrParam, [462](#)
- setWarmStart, [463](#)
- solveFromHotStart, [463](#)
- status, [472](#)
- unmarkHotStart, [463](#)
- writeMps, [471](#)
- writeMpsNative, [471](#)
- OsiCbcSolverInterface.hpp
  - OsiCbcHasNDEBUG, [546](#)
  - OsiCbcInfinity, [546](#)
  - OsiCbcSolverInterfaceUnitTest, [546](#)
- OsiCbcSolverInterfaceUnitTest
  - OsiCbcSolverInterface, [473](#)
  - OsiCbcSolverInterface.hpp, [546](#)
- OsiChooseStrongSubset, [473](#)
  - ~OsiChooseStrongSubset, [474](#)
  - chooseVariable, [475](#)
  - clone, [475](#)
  - numberObjectsToUse, [475](#)
  - numberObjectsToUse\_, [475](#)
  - operator=, [475](#)
  - OsiChooseStrongSubset, [474](#)
  - OsiChooseStrongSubset, [474](#)
  - setNumberObjectsToUse, [475](#)
  - setupList, [475](#)
- OsiLink, [476](#)
  - ~OsiLink, [477](#)
  - boundBranch, [477](#)
  - canDoHeuristics, [477](#)
  - clone, [477](#)
  - createBranch, [477](#)
  - feasibleRegion, [477](#)
  - infeasibility, [477](#)
  - numberLinks, [477](#)
  - operator=, [477](#)
  - OsiLink, [477](#)
  - OsiLink, [477](#)
  - resetSequenceEtc, [477](#)
- OsiLinkBranchingObject, [478](#)
  - ~OsiLinkBranchingObject, [478](#)
  - branch, [479](#)
  - clone, [479](#)
  - operator=, [478](#)
  - OsiLinkBranchingObject, [478](#)
  - OsiLinkBranchingObject, [478](#)
  - print, [479](#)
- OsiLinkedBound, [479](#)
  - ~OsiLinkedBound, [480](#)
  - addBoundModifier, [480](#)
  - operator=, [480](#)
  - OsiLinkedBound, [480](#)
  - OsiLinkedBound, [480](#)
  - updateBounds, [480](#)
  - variable, [480](#)
- osiObject
  - CbcSimpleInteger, [364](#)
  - CbcSOS, [393](#)
- OsiOldLink, [480](#)
  - ~OsiOldLink, [482](#)
  - boundBranch, [483](#)
  - canDoHeuristics, [482](#)
  - clone, [482](#)
  - createBranch, [482](#)
  - feasibleRegion, [482](#)
  - infeasibility, [482](#)
  - numberLinks, [482](#)
  - operator=, [482](#)
  - OsiOldLink, [481](#), [482](#)
  - OsiOldLink, [481](#), [482](#)
  - resetSequenceEtc, [482](#)
- OsiOldLinkBranchingObject, [483](#)
  - ~OsiOldLinkBranchingObject, [484](#)
  - branch, [484](#)
  - clone, [484](#)
  - operator=, [484](#)
  - OsiOldLinkBranchingObject, [484](#)
  - OsiOldLinkBranchingObject, [484](#)
  - print, [484](#)
- OsiOneLink, [484](#)
  - ~OsiOneLink, [485](#)
  - function\_, [485](#)
  - operator=, [485](#)
  - OsiOneLink, [485](#)
  - OsiOneLink, [485](#)
  - xColumn\_, [485](#)

- xRow\_, 485
- xyRow, 485
- osiParams\_
  - CbcGenCtlBlk, 135
- OsiSimpleFixedInteger, 486
  - ~OsiSimpleFixedInteger, 487
  - clone, 488
  - createBranch, 488
  - infeasibility, 488
  - operator=, 488
  - OsiSimpleFixedInteger, 487
  - OsiSimpleFixedInteger, 487
- OsiSolverLinearizedQuadratic, 488
  - ~OsiSolverLinearizedQuadratic, 489
  - bestObjectiveValue, 490
  - bestObjectiveValue\_, 490
  - bestSolution, 490
  - bestSolution\_, 491
  - clone, 490
  - initialSolve, 490
  - operator=, 490
  - OsiSolverLinearizedQuadratic, 489
  - OsiSolverLinearizedQuadratic, 489
  - quadraticModel, 490
  - quadraticModel\_, 490
  - setSpecialOptions3, 490
  - specialOptions3, 490
  - specialOptions3\_, 491
- OsiSolverLink, 491
  - ~OsiSolverLink, 495
  - addBoundModifier, 495
  - addTighterConstraints, 496
  - analyzeObjects, 496
  - bestObjectiveValue, 496
  - bestObjectiveValue\_, 500
  - bestSolution, 496
  - bestSolution\_, 500
  - biLinearPriority, 497
  - biLinearPriority\_, 500
  - cleanMatrix, 496
  - clone, 495
  - coinModel, 498
  - coinModel\_, 499
  - convex\_, 499
  - defaultBound, 497
  - defaultBound\_, 500
  - defaultMeshSize, 497
  - defaultMeshSize\_, 500
  - doAOCuts, 495
  - fathom, 495
  - fixVariables\_, 500
  - gutsOfCopy, 498
  - gutsOfDestructor, 498
  - heuristicSolution, 495
  - info\_, 499
  - initialSolve, 495
  - integerPriority, 497
  - integerPriority\_, 500
  - linearizedBAB, 495
  - load, 495
  - matrix\_, 498
  - nonlinearSLP, 495
  - numberFix\_, 500
  - numberNonLinearRows\_, 498
  - numberVariables\_, 499
  - objectiveRow\_, 499
  - objectiveVariable, 497
  - objectiveVariable\_, 499
  - operator=, 495
  - originalRowCopy, 496
  - originalRowCopy\_, 498
  - OsiSolverLink, 494
  - OsiSolverLink, 494
  - quadraticModel, 497
  - quadraticModel\_, 498
  - quadraticRow, 497
  - resolve, 495
  - rowNonLinear\_, 499
  - sayConvex, 496
  - setBestObjectiveValue, 496
  - setBestSolution, 496
  - setBiLinearPriorities, 498
  - setBiLinearPriority, 497
  - setBranchingStrategyOnVariables, 498
  - setDefaultBound, 497
  - setDefaultMeshSize, 497
  - setFixedPriority, 498
  - setIntegerPriority, 497
  - setMeshSizes, 498
  - setSpecialOptions2, 496
  - specialOptions2, 496
  - specialOptions2\_, 499
  - startNonLinear\_, 499
  - updateCoefficients, 496
  - whichNonLinear\_, 499
- OsiUsesBiLinear, 500
  - ~OsiUsesBiLinear, 502
  - addBiLinearObjects, 503
  - clone, 502
  - createBranch, 502
  - feasibleRegion, 503
  - infeasibility, 502
  - numberBiLinear\_, 503
  - objects\_, 503
  - operator=, 502
  - OsiUsesBiLinear, 502
  - OsiUsesBiLinear, 502
  - type\_, 503



ownObjects  
     CbcModel, [302](#)  
 owner  
     CbcNodeInfo, [324](#)  
 owner\_  
     CbcNodeInfo, [325](#)  
  
 P  
 PERTURBATION  
     CbcOsiParam, [343](#)  
 PERTVALUE  
     CbcOsiParam, [344](#)  
 PFI  
     CbcOsiParam, [344](#)  
 PLUSMINUS  
     CbcOsiParam, [344](#)  
 PREPROCESS  
     CbcGenParam, [141](#)  
 PRESOLVE  
     CbcOsiParam, [344](#)  
 PRESOLVEOPTIONS  
     CbcOsiParam, [344](#)  
 PRESOLVEPASS  
     CbcOsiParam, [344](#)  
 PRIMALPIVOT  
     CbcOsiParam, [344](#)  
 PRIMALSIMPLEX  
     CbcOsiParam, [344](#)  
 PRIMALTOLERANCE  
     CbcOsiParam, [344](#)  
 PRINTMASK  
     CbcGenParam, [141](#)  
 PRINTOPTIONS  
     CbcGenParam, [141](#)  
 PRINTVERSION  
     CbcGenParam, [141](#)  
 PRIORITYIN  
     CbcGenParam, [141](#)  
 PROBINGCUTS  
     CbcGenParam, [141](#)  
 parallelMode  
     CbcModel, [304](#)  
 paramCode  
     CbcCbcParam, [53](#)  
     CbcGenParam, [143](#)  
     CbcOsiParam, [345](#)  
 paramVec\_  
     CbcGenCtlBlk, [135](#)  
 parameterOption  
     CbcParam, [351](#)  
 paramsProcessed\_  
     CbcGenCtlBlk, [135](#)  
 parent  
     CbcNodeInfo, [323](#)  
  
 parent\_  
     CbcNodeInfo, [325](#)  
 parentBranch  
     CbcNodeInfo, [325](#)  
 parentBranch\_  
     CbcNodeInfo, [325](#)  
 parentId\_  
     CbcStatistics, [400](#)  
 parentModel  
     CbcModel, [297](#)  
 parentModel\_  
     CbcStrategyDefaultSubTree, [410](#)  
 parentNode  
     CbcStatistics, [399](#)  
 partialNodeInfo  
     CbcStrategy, [405](#)  
 pass  
     CbcHeuristicDW, [185](#)  
 pass\_  
     CbcHeuristicDW, [189](#)  
 passInContinuousSolution  
     CbcHeuristicDW, [182](#)  
 passInEventHandler  
     CbcModel, [298](#)  
 passInMessageHandler  
     CbcGenCtlBlk, [133](#)  
     CbcModel, [298](#)  
     OsiCbcSolverInterface, [472](#)  
 passInPriorities  
     CbcModel, [298](#)  
 passInSolution  
     CbcHeuristicDW, [182](#)  
     CbcTreeLocal, [424](#)  
     CbcTreeVariable, [427](#)  
 passInSolverCharacteristics  
     CbcModel, [298](#)  
 passInSubTreeModel  
     CbcModel, [294](#)  
 passInTreeHandler  
     CbcModel, [294](#)  
 penaltyScaleFactor  
     CbcModel, [282](#)  
 percentageToFix\_  
     CbcHeuristicDive, [166](#)  
 phase  
     CbcHeuristicDW, [185](#)  
     CbcModel, [292](#)  
 phase\_  
     CbcHeuristicDW, [189](#)  
 pop  
     CbcTree, [418](#)  
     CbcTreeLocal, [423](#)  
     CbcTreeVariable, [427](#)  
 position



- CbcObject, 337
- position\_
  - CbcObject, 338
- possible
  - CbcFathom, 104
- possible\_
  - CbcFathom, 105
- possibleBranch
  - CbcStrongInfo, 413
- preProcessPasses
  - CbcStrategyDefault, 407
- preProcessPasses\_
  - CbcStrategyDefault, 408
- preProcessState
  - CbcStrategy, 404
- preProcessState\_
  - CbcStrategy, 405
- preferredNewFeasible
  - CbcBranchCut, 32
  - CbcLotsize, 250
  - CbcObject, 336
- preferredWay
  - CbcObject, 338
- preferredWay\_
  - CbcObject, 338
  - CbcSimpleInteger, 366
- previousBounds
  - CbcModel, 306
- previousBranch
  - CbcBranchingObject, 44
  - CbcSOSBranchingObject, 396
- primalSolution
  - ampl\_info, 25
- primalSolution\_
  - CbcSolverUsefulData, 390
- print
  - CbcBranchingObject, 45
  - CbcCliqueBranchingObject, 60
  - CbcCutBranchingObject, 78
  - CbcDummyBranchingObject, 95
  - CbcFixingBranchingObject, 112
  - CbcIntegerBranchingObject, 241
  - CbcLongCliqueBranchingObject, 247
  - CbcLotsizeBranchingObject, 253
  - CbcNode, 318
  - CbcNWayBranchingObject, 331
  - CbcSimpleIntegerDynamicPseudoCost, 378
  - CbcSOSBranchingObject, 396
  - CbcStatistics, 398
  - OsiBiLinearBranchingObject, 453
  - OsiLinkBranchingObject, 479
  - OsiOldLinkBranchingObject, 484
- printBaBStatus
  - CbcGenCtlBlk, 133
- printDistanceToNodes
  - CbcHeuristic, 152
- printFrequency
  - CbcModel, 283
- printLevel\_
  - CbcStrategyDefault, 408
  - CbcStrategyDefaultSubTree, 410
- printLongHelp
  - CbcParam, 353
- printMask\_
  - CbcGenCtlBlk, 135
- printMode\_
  - CbcGenCtlBlk, 135
- printOpt\_
  - CbcGenCtlBlk, 134
- printOptions
  - CbcParam, 352
- printString
  - CbcParam, 353
- priorities
  - ampl\_info, 25
- priorities\_
  - CbcSolverUsefulData, 390
- priority
  - CbcModel, 298
- priorityAction\_
  - CbcGenCtlBlk, 137
- probabilities\_
  - CbcHeuristicJustOne, 212
- probingInfo
  - CbcModel, 308
- problemFeasibility
  - CbcModel, 294
- problemStatus
  - ampl\_info, 24
- problemType
  - CbcModel, 282
- process
  - CbcStrategy, 404
- process\_
  - CbcStrategy, 405
- pseudoDown
  - ampl\_info, 25
- pseudoDown\_
  - CbcSolverUsefulData, 390
- pseudoRedCost
  - PseudoReducedCost, 503
- PseudoReducedCost, 503
  - pseudoRedCost, 503
  - var, 503
- pseudoShadow
  - CbcModel, 307
- pseudoUp
  - ampl\_info, 25

- pseudoUp\_
  - CbcSolverUsefulData, 390
- push
  - CbcTree, 418
  - CbcTreeLocal, 423
  - CbcTreeVariable, 427
- pushCbcCbcDbI
  - CbcCbcParamUtils, 21
- pushCbcCbcInt
  - CbcCbcParamUtils, 21
- pushCbcGenCutParam
  - CbcGenParamUtils, 22
- pushCbcGenDbIParam
  - CbcGenParamUtils, 22
- pushCbcGenIntParam
  - CbcGenParamUtils, 22
- pushCbcGenKwdParam
  - CbcGenParamUtils, 22
- pushCbcGenStrParam
  - CbcGenParamUtils, 22
- pushCbcOsiDbI
  - CbcOsiParamUtils, 22
- pushCbcOsiHint
  - CbcOsiParamUtils, 22
- pushCbcOsiInt
  - CbcOsiParamUtils, 22
- pushCbcOsiKwd
  - CbcOsiParamUtils, 22
- pushCbcOsiLogLevel
  - CbcOsiParamUtils, 22
- Q
  - quadraticModel
    - OsiSolverLinearizedQuadratic, 490
    - OsiSolverLink, 497
  - quadraticModel\_
    - OsiSolverLinearizedQuadratic, 490
    - OsiSolverLink, 498
  - quadraticRow
    - OsiSolverLink, 497
- R
  - REALLY\_SCALE
    - CbcOsiParam, 344
  - REDSPLITCUTS
    - CbcGenParam, 141
  - RESTORE
    - CbcOsiParam, 344
  - REVERSE
    - CbcOsiParam, 344
  - RHSSCALE
    - CbcOsiParam, 344
  - ROUNDING
    - CbcGenParam, 141
  - random\_
    - CbcHeuristicCrossover, 157
    - CbcHeuristicDW, 187
  - randomNumberGenerator
    - CbcModel, 308
  - randomNumberGenerator\_
    - CbcHeuristic, 153
    - CbcIdiotBranch, 238
  - range
    - CbcTreeLocal, 424
    - CbcTreeVariable, 427
  - rangeType
    - CbcLotsize, 251
  - readAmpl
    - Cbc\_ampl.h, 504
  - readMIPStart
    - CbcMipStartIO.hpp, 531
  - readMps
    - OsiCbcSolverInterface, 470
  - readSolution
    - CbcGenParamUtils, 21
  - reallyScale
    - ClpAmplObjective, 436
    - ClpConstraintAmpl, 438
  - realpop
    - CbcTree, 419
  - realpush
    - CbcTree, 419
  - rebuild
    - CbcTree, 418
  - redoSequenceEtc
    - CbcBranchToFixLots, 49
    - CbcCliques, 57
    - CbcGeneral, 138
    - CbcNWay, 329
    - CbcObject, 338
    - CbcSOS, 393
  - redoWalkBack
    - CbcModel, 310
  - reducedCostFix
    - CbcHeuristicDive, 165
    - CbcModel, 305
  - reducedCostMultiplier
    - CbcHeuristicFPump, 199
  - reducedCostMultiplier\_
    - CbcHeuristicFPump, 200
  - reducedGradient
    - ClpAmplObjective, 435
  - referenceSolver
    - CbcModel, 303
  - refine
    - CbcTreeLocal, 425
    - CbcTreeVariable, 428
  - refreshModel
    - CbcCutGenerator, 83

- relativeIncrement
  - CbcHeuristicFPump, 196
- relativeIncrement\_
  - CbcHeuristicFPump, 199
- rensType\_
  - CbcHeuristicRENS, 229
- reserveCurrentSolution
  - CbcModel, 290
- resetBounds
  - CbcBranchCut, 32
  - CbcLotsize, 250
  - CbcObject, 336
  - CbcSimpleInteger, 365
- resetModel
  - CbcFathom, 104
  - CbcFathomDynamicProgramming, 107
  - CbcHeuristic, 148
  - CbcHeuristicCrossover, 156
  - CbcHeuristicDINS, 159
  - CbcHeuristicDive, 163
  - CbcHeuristicDW, 181
  - CbcHeuristicDynamic3, 192
  - CbcHeuristicFPump, 195
  - CbcHeuristicGreedyCover, 202
  - CbcHeuristicGreedyEquality, 205
  - CbcHeuristicGreedySOS, 209
  - CbcHeuristicJustOne, 211
  - CbcHeuristicLocal, 214
  - CbcHeuristicNaive, 216
  - CbcHeuristicPartial, 220
  - CbcHeuristicPivotAndFix, 222
  - CbcHeuristicProximity, 224
  - CbcHeuristicRandRound, 227
  - CbcHeuristicRENS, 228
  - CbcHeuristicRINS, 231
  - CbcHeuristicVND, 234
  - CbcModel, 303
  - CbcRounding, 358
  - CbcSerendipity, 362
- resetNodeNumbers
  - CbcTree, 419
- resetNumberBranchesLeft
  - CbcBranchingObject, 44
- resetSequenceEtc
  - CbcSimpleInteger, 365
  - OsiBiLinear, 446
  - OsiLink, 477
  - OsiOldLink, 482
- resetToReferenceSolver
  - CbcModel, 303
- resize
  - ClpAmplObjective, 435
  - ClpConstraintAmpl, 438
- resolve
  - CbcModel, 274, 305
  - OsiCbcSolverInterface, 462
  - OsiSolverLink, 495
- resolveAfterTakeOffCuts
  - CbcModel, 293
- restart
  - CbcEventHandler, 101
- restartRoot
  - CbcEventHandler, 101
- reverseCut
  - CbcTreeLocal, 424
  - CbcTreeVariable, 427
- rhs\_
  - CbcFathomDynamicProgramming, 109
  - CbcFollowOn, 117
- rootObjectiveAfterCuts
  - CbcModel, 292
- roundExpensive\_
  - CbcHeuristicFPump, 200
- rowCutPtr
  - CbcRowCuts, 360
- rowLower
  - ampl\_info, 24
- rowNonLinear\_
  - OsiSolverLink, 499
- rowStatus
  - ampl\_info, 25
- rowUpper
  - ampl\_info, 24
- rows
  - ampl\_info, 25
- rowsInBlock\_
  - CbcHeuristicDW, 188
- runNodes\_
  - CbcHeuristic, 154
- S
- SAVE
  - CbcOsiParam, 344
- SCALING
  - CbcOsiParam, 344
- SHOWUNIMP
  - CbcGenParam, 142
- SLPVALUE
  - CbcOsiParam, 344
- SOLUTION
  - CbcGenParam, 142
- SOLVECONTINUOUS
  - CbcGenParam, 142
- SOLVER
  - CbcGenParam, 142
- SOLVERLOGLEVEL
  - CbcOsiParam, 344
- SOS

- CbcGenParam, [142](#)
- SPARSEFACTOR
  - CbcOsiParam, [344](#)
- SPECIALOPTIONS
  - CbcOsiParam, [344](#)
- SPRINT
  - CbcOsiParam, [344](#)
- STDIN
  - CbcGenParam, [142](#)
- STRENGTHEN
  - CbcGenParam, [142](#)
- STRONGBRANCHING
  - CbcCbcParam, [52](#)
- SCANCUTS
  - CbcCutGenerator.hpp, [516](#)
- SCANCUTS\_PROBING
  - CbcCutGenerator.hpp, [516](#)
- same
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
- saveBestSolution
  - CbcModel, [305](#)
- saveBranchingObject
  - CbcBranchDecision, [34](#)
  - CbcBranchDynamicDecision, [40](#)
- saveExtraSolution
  - CbcModel, [305](#)
- saveLower\_
  - CbcHeuristicDW, [187](#)
- saveModel
  - CbcModel, [275](#)
- saveReferenceSolver
  - CbcModel, [303](#)
- saveSolution
  - CbcGenParamUtils, [21](#)
- saveUpper\_
  - CbcHeuristicDW, [187](#)
- saveWeight\_
  - CbcCompareDefault, [68](#)
- savedRandomNumberGenerator\_
  - CbcIdiotBranch, [238](#)
- savedSolution
  - CbcModel, [291](#)
- savedSolutionObjective
  - CbcModel, [292](#)
- sayConvex
  - OsiSolverLink, [496](#)
- sayEventHappened
  - CbcModel, [300](#)
- sayInfeasible
  - CbcStatistics, [398](#)
- sayThreaded
  - CbcCompareBase, [64](#)
- scaleBackStatistics
  - CbcCutGenerator, [89](#)
- searchStrategy
  - CbcModel, [295](#)
- secondaryStatus
  - CbcModel, [284](#)
- seed\_
  - CbcRounding, [359](#)
- selectBinaryVariables
  - CbcHeuristicDive, [164](#)
- selectVariableToBranch
  - CbcHeuristicDive, [165](#)
  - CbcHeuristicDiveCoefficient, [168](#)
  - CbcHeuristicDiveFractional, [169](#)
  - CbcHeuristicDiveGuided, [171](#)
  - CbcHeuristicDiveLineSearch, [172](#)
  - CbcHeuristicDivePseudoCost, [174](#)
  - CbcHeuristicDiveVectorLength, [176](#)
  - CbcHeuristicJustOne, [211](#)
- sequence\_
  - CbcStatistics, [400](#)
- setAbsoluteIncrement
  - CbcHeuristicFPump, [196](#)
- setAccumulate
  - CbcHeuristicFPump, [198](#)
- setAction
  - CbcEventHandler, [102](#)
- setActive
  - CbcNode, [317](#)
- setAlgorithm
  - CbcFathomDynamicProgramming, [108](#)
  - CbcHeuristicGreedyCover, [203](#)
  - CbcHeuristicGreedyEquality, [206](#)
  - CbcHeuristicGreedySOS, [209](#)
- setAllowableFractionGap
  - CbcModel, [279](#)
- setAllowableGap
  - CbcModel, [278](#)
- setAllowablePercentageGap
  - CbcModel, [279](#)
- setApplicationData
  - CbcModel, [298](#)
- setArtificialCost
  - CbcHeuristicFPump, [197](#)
- setAtSolution
  - CbcCutGenerator, [85](#)
- setBaBStatus
  - CbcGenCtlBlk, [133](#)
- setBestCriterion
  - CbcBranchDecision, [35](#)
  - CbcBranchDefaultDecision, [37](#)
  - CbcBranchDynamicDecision, [39](#)
- setBestObjectiveValue
  - CbcModel, [289](#)
  - OsiSolverLink, [496](#)
- setBestPossible

- CbcCompareDefault, [67](#)
- setBestSolution
  - CbcModel, [289](#), [291](#)
  - OsiSolverLink, [496](#)
- setBestSolutionBasis
  - CbcModel, [310](#)
- setBiLinearPriorities
  - OsiSolverLink, [498](#)
- setBiLinearPriority
  - OsiSolverLink, [497](#)
- setBoundType
  - OsiBiLinear, [448](#)
- setBranchingMethod
  - CbcModel, [295](#)
- setBranchingObject
  - CbcNode, [317](#)
- setBranchingStrategy
  - OsiBiLinear, [448](#)
- setBranchingStrategyOnVariables
  - OsiSolverLink, [498](#)
- setBreadthDepth
  - CbcCompareDefault, [67](#)
- setBreakEven
  - CbcSimpleInteger, [366](#)
- setByUser\_
  - CbcGenCtlBlk, [136](#)
- setCbcModel
  - CbcOsiSolver, [347](#)
- setCbcModelDefaults
  - CbcCbcParamUtils, [20](#)
- setChangeInGuessed
  - CbcDynamicPseudoCostBranchingObject, [98](#)
  - CbcIntegerPseudoCostBranchingObject, [245](#)
- setChooseMethod
  - CbcBranchDecision, [35](#)
- setCliqueAction
  - CbcGenCtlBlk, [130](#)
- setCoefficient
  - OsiBiLinear, [447](#)
- setColBounds
  - OsiCbcSolverInterface, [467](#)
- setColLower
  - CbcFullNodeInfo, [120](#)
  - OsiCbcSolverInterface, [467](#)
- setColName
  - OsiCbcSolverInterface, [466](#)
- setColNames
  - OsiCbcSolverInterface, [466](#)
- setColSetBounds
  - OsiCbcSolverInterface, [467](#)
- setColSolution
  - OsiCbcSolverInterface, [468](#)
- setColUpper
  - CbcFullNodeInfo, [121](#)
- OsiCbcSolverInterface, [467](#)
- setColumnNumber
  - CbcSimpleInteger, [365](#)
- setCombineAction
  - CbcGenCtlBlk, [132](#)
- setComparison
  - CbcTree, [418](#)
- setConsequence
  - CbcNWay, [328](#)
- setConstraint
  - CbcHeuristicDINS, [160](#)
- setContinuous
  - OsiCbcSolverInterface, [468](#)
- setContinuousInfeasibilities
  - CbcModel, [292](#)
- setContinuousObjective
  - CbcModel, [292](#)
- setContinuousPriority
  - CbcModel, [309](#)
- setCurrentNumberNeeded
  - CbcHeuristicDW, [183](#)
- setCurrentNumberNodes
  - CbcHeuristicDW, [184](#)
- setCurrentOption
  - CbcParam, [352](#)
- setCurrentPassNumber
  - CbcModel, [281](#)
- setCutAndHeuristicOptions
  - CbcModel.hpp, [532](#)
- setCutDepth
  - CbcGenCtlBlk, [130](#)
- setCutModifier
  - CbcModel, [295](#)
- setCutoff
  - CbcCompareDefault, [67](#)
  - CbcModel, [277](#)
  - OsiCbcSolverInterface, [471](#)
- setCutoffAsConstraint
  - CbcModel, [301](#)
- setCutoffIncrement
  - CbcModel, [279](#)
- setDblParam
  - CbcModel, [277](#)
  - OsiCbcSolverInterface, [462](#)
- setDecayFactor
  - CbcHeuristic, [151](#)
- setDefaultBound
  - OsiSolverLink, [497](#)
- setDefaultHandler
  - CbcModel, [299](#)
- setDefaultMeshSize
  - OsiSolverLink, [497](#)
- setDefaultRounding
  - CbcHeuristicFPump, [196](#)

setDepth  
     CbcNode, 316  
 setDfltAction  
     CbcEventHandler, 102  
 setDoubleParameter  
     CbcParam, 350, 351  
 setDoubleValue  
     CbcParam, 352  
     CbcSolver, 388  
 setDownBounds  
     CbcIntegerBranchingObject, 241  
 setDownDynamicPseudoCost  
     CbcSimpleIntegerDynamicPseudoCost, 373  
 setDownInformation  
     CbcSimpleIntegerDynamicPseudoCost, 377  
 setDownPseudoCost  
     CbcSimpleIntegerPseudoCost, 383  
 setDownShadowPrice  
     CbcSimpleIntegerDynamicPseudoCost, 373  
 setFPumpAction  
     CbcGenCtlBlk, 131  
 setFakeCutoff  
     CbcHeuristicFPump, 196  
 setFastNodeDepth  
     CbcModel, 309  
 setFeasibilityPumpOptions  
     CbcHeuristic, 150  
 setFixOnReducedCosts  
     CbcHeuristicFPump, 198  
 setFixPriority  
     CbcHeuristicPartial, 221  
 setFixedPriority  
     OsiSolverLink, 498  
 setFlowAction  
     CbcGenCtlBlk, 130  
 setFraction  
     CbcHeuristicGreedyEquality, 206  
 setFractionSmall  
     CbcHeuristic, 150  
 setGlobalCuts  
     CbcCutGenerator, 89  
 setGlobalCutsAtRoot  
     CbcCutGenerator, 89  
 setGomoryAction  
     CbcGenCtlBlk, 131  
 setGreedyCoverAction  
     CbcGenCtlBlk, 132  
 setGreedyEqualityAction  
     CbcGenCtlBlk, 132  
 setGuessedObjectiveValue  
     CbcNode, 316  
 setHeuristicFractionGap  
     CbcModel, 279  
 setHeuristicGap  
     CbcModel, 279  
 setHeuristicModel  
     CbcModel, 294  
 setHeuristicName  
     CbcHeuristic, 151  
 setHintParam  
     OsiCbcSolverInterface, 462  
 setHotstartSolution  
     CbcModel, 280  
 setHowOften  
     CbcCutGenerator, 83  
     CbcHeuristicDINS, 160  
     CbcHeuristicDW, 184  
     CbcHeuristicRINS, 231  
     CbcHeuristicVND, 235  
 setHowOftenGlobalScan  
     CbcModel, 282  
 setHowOftenShallow  
     CbcHeuristic, 152  
 setIPPAAction  
     CbcGenCtlBlk, 130  
 setId  
     CbcObject, 337  
 setInaccuracy  
     CbcCutGenerator, 84  
 setIncrement  
     CbcHeuristicProximity, 225  
 setIneffectualCuts  
     CbcCutGenerator, 88  
 setInfeasibilityWeight  
     CbcModel, 278  
 setInfo  
     CbcCountRowCut, 76  
 setInfoInChild  
     CbcModel, 305  
 setInitialWeight  
     CbcHeuristicFPump, 197  
 setInputSolution  
     CbcHeuristic, 151  
 setIntParam  
     CbcModel, 277  
     OsiCbcSolverInterface, 462  
 setIntParameter  
     CbcParam, 351  
 setIntValue  
     CbcParam, 352  
     CbcSolver, 388  
 setInteger  
     OsiCbcSolverInterface, 468  
 setIntegerPriority  
     OsiSolverLink, 497  
 setIntegerTolerance  
     CbcModel, 278  
 setIntegerValued

- CbcSOS, [394](#)
- setIterationRatio
  - CbcHeuristicFPump, [197](#)
- setKnapsackAction
  - CbcGenCtlBlk, [131](#)
- setLanguage
  - CbcModel, [299](#)
  - OsiCbcSolverInterface, [471](#)
- setLargeValue
  - CbcHeuristicNaive, [217](#)
- setLastHeuristic
  - CbcModel, [297](#)
- setLastNode
  - CbcHeuristicRINS, [232](#)
- setLogLevel
  - CbcGenCtlBlk, [134](#)
  - CbcModel, [299](#)
- setLonghelp
  - CbcParam, [353](#)
- setMIPStart
  - CbcModel, [310](#)
- setMaxDiversification
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [428](#)
- setMaxIterations
  - CbcHeuristicDive, [164](#)
- setMaxSimplexIterations
  - CbcHeuristicDive, [164](#)
- setMaxSimplexIterationsAtRoot
  - CbcHeuristicDive, [165](#)
- setMaxTime
  - CbcHeuristicDive, [165](#)
- setMaximumBranching
  - CbcTree, [420](#)
- setMaximumCutPasses
  - CbcModel, [280](#)
- setMaximumCutPassesAtRoot
  - CbcModel, [280](#)
- setMaximumKeep
  - CbcHeuristicDINS, [160](#)
- setMaximumNodes
  - CbcModel, [277](#)
  - OsiCbcSolverInterface, [471](#)
- setMaximumNumberIterations
  - CbcModel, [309](#)
- setMaximumPasses
  - CbcHeuristicFPump, [197](#)
- setMaximumRetries
  - CbcHeuristicFPump, [198](#)
- setMaximumSavedSolutions
  - CbcModel, [291](#)
- setMaximumSeconds
  - CbcModel, [278](#)
  - OsiCbcSolverInterface, [472](#)
- setMaximumSize
  - CbcFathomDynamicProgramming, [108](#)
- setMaximumSolutions
  - CbcModel, [277](#)
  - OsiCbcSolverInterface, [472](#)
- setMaximumTime
  - CbcHeuristicFPump, [196](#)
- setMaximumTries
  - CbcCutGenerator, [84](#)
- setMeshSizes
  - OsiBiLinear, [448](#)
  - OsiSolverLink, [498](#)
- setMessages
  - CbcGenCtlBlk, [134](#)
- setMethod
  - CbcSimpleIntegerDynamicPseudoCost, [377](#)
  - CbcSimpleIntegerPseudoCost, [384](#)
- setMinDistanceToRun
  - CbcHeuristic, [152](#)
- setMinimizationObjValue
  - CbcModel, [290](#)
- setMinimumDrop
  - CbcModel, [280](#)
- setMirAction
  - CbcGenCtlBlk, [131](#)
- setModel
  - CbcBranchingObject, [45](#)
  - CbcCutGenerator, [89](#)
  - CbcEventHandler, [102](#)
  - CbcFathom, [104](#)
  - CbcFathomDynamicProgramming, [107](#)
  - CbcHeuristic, [148](#)
  - CbcHeuristicCrossover, [157](#)
  - CbcHeuristicDINS, [159](#)
  - CbcHeuristicDive, [164](#)
  - CbcHeuristicDW, [182](#)
  - CbcHeuristicDynamic3, [191](#)
  - CbcHeuristicFPump, [195](#)
  - CbcHeuristicGreedyCover, [202](#)
  - CbcHeuristicGreedyEquality, [205](#)
  - CbcHeuristicGreedySOS, [208](#)
  - CbcHeuristicJustOne, [211](#)
  - CbcHeuristicLocal, [214](#)
  - CbcHeuristicNaive, [216](#)
  - CbcHeuristicPartial, [220](#)
  - CbcHeuristicPivotAndFix, [222](#)
  - CbcHeuristicProximity, [224](#)
  - CbcHeuristicRandRound, [227](#)
  - CbcHeuristicRENS, [228](#)
  - CbcHeuristicRINS, [231](#)
  - CbcHeuristicVND, [235](#)
  - CbcObject, [337](#)
  - CbcRounding, [358](#)
  - CbcSerendipity, [361](#)



- setModelOnly
  - CbcHeuristic, 150
- setModelOwnsSolver
  - CbcModel, 302
- setModelSequence
  - CbcLotsize, 250
- setMoreSpecialOptions
  - CbcModel, 300
- setMoreSpecialOptions2
  - CbcModel, 301
- setMultipleRootTries
  - CbcModel, 300
- setMustCallAgain
  - CbcCutGenerator, 87
- setNeedsOptimalBasis
  - CbcCutGenerator, 87
- setNested
  - CbcStrategy, 404
- setNextRowCut
  - CbcModel, 308
- setNodeComparison
  - CbcModel, 294
- setNodeLimit
  - CbcTreeLocal, 425
  - CbcTreeVariable, 428
- setNodeNumber
  - CbcNode, 317
  - CbcNodeInfo, 324
- setNormal
  - CbcCutGenerator, 85
- setNumberActiveCutsAtRoot
  - CbcCutGenerator, 89
- setNumberAnalyzeIterations
  - CbcModel, 282
- setNumberBadPasses
  - CbcHeuristicDW, 183
- setNumberBeforeTrust
  - CbcModel, 281
  - CbcSimpleIntegerDynamicPseudoCost, 377
- setNumberBranches
  - CbcBranchingObject, 44
- setNumberBranchesLeft
  - CbcNodeInfo, 322
- setNumberBranching
  - CbcTree, 419
- setNumberColumnCuts
  - CbcCutGenerator, 86
- setNumberCuts
  - CbcNodeInfo, 324
- setNumberCutsActive
  - CbcCutGenerator, 87
- setNumberCutsAtRoot
  - CbcCutGenerator, 88
- setNumberCutsInTotal
  - CbcCutGenerator, 86
- setNumberElementsInTotal
  - CbcCutGenerator, 86
- setNumberHeuristicSolutions
  - CbcModel, 292
- setNumberHeuristics
  - CbcModel, 297
- setNumberMembers
  - CbcSOS, 394
- setNumberNeeded
  - CbcHeuristicDW, 183
- setNumberNodes
  - CbcHeuristic, 149
  - CbcHeuristicDW, 184
- setNumberObjects
  - CbcModel, 276
- setNumberObjectsToUse
  - OsiChooseStrongSubset, 475
- setNumberPasses
  - CbcHeuristicDW, 183
- setNumberPenalties
  - CbcModel, 282
- setNumberPointingToThis
  - CbcNodeInfo, 323
- setNumberPoints
  - OsiBiLinearEquality, 454
- setNumberShortCutsAtRoot
  - CbcCutGenerator, 89
- setNumberSolutions
  - CbcHeuristicCrossover, 157
- setNumberStrong
  - CbcModel, 281
- setNumberStrongIterations
  - CbcModel, 308
- setNumberThreads
  - CbcModel, 304
- setNumberTimes
  - CbcHeuristicGreedyCover, 203
  - CbcHeuristicGreedyEquality, 206
  - CbcHeuristicGreedySOS, 209
- setNumberTimesDown
  - CbcSimpleIntegerDynamicPseudoCost, 376
- setNumberTimesDownInfeasible
  - CbcSimpleIntegerDynamicPseudoCost, 376
- setNumberTimesEntered
  - CbcCutGenerator, 86
- setNumberTimesUp
  - CbcSimpleIntegerDynamicPseudoCost, 376
- setNumberTimesUpInfeasible
  - CbcSimpleIntegerDynamicPseudoCost, 376
- setNumberUnsatisfied
  - CbcNode, 316
- setObj
  - CbcCbcParam, 53



- CbcGenParam, [143](#)
- CbcOsiParam, [345](#)
- setObjCoeff
  - OsiCbcSolverInterface, [467](#)
- setObjName
  - OsiCbcSolverInterface, [466](#)
- setObjSense
  - CbcModel, [292](#)
  - OsiCbcSolverInterface, [468](#)
- setObjValue
  - CbcModel, [291](#)
- setObject
  - CbcDynamicPseudoCostBranchingObject, [98](#)
- setObjectiveValue
  - CbcModel, [307](#)
  - CbcNode, [316](#)
- setOnTree
  - CbcNode, [317](#)
- setOptionalInteger
  - CbcModel, [276](#)
- setOriginalCoinModel
  - CbcSolver, [388](#)
- setOriginalColumns
  - CbcModel, [283](#)
- setOriginalLowerBound
  - CbcSimpleInteger, [365](#)
- setOriginalObject
  - CbcBranchingObject, [45](#)
- setOriginalSolver
  - CbcSolver, [388](#)
- setOriginalUpperBound
  - CbcSimpleInteger, [365](#)
- setOsiSolverInterfaceDefaults
  - CbcOsiParamUtils, [22](#)
- setParamCode
  - CbcCbcParam, [53](#)
  - CbcGenParam, [143](#)
  - CbcOsiParam, [345](#)
- setParentModel
  - CbcModel, [297](#)
- setPenaltyScaleFactor
  - CbcModel, [282](#)
- setPercentageToFix
  - CbcHeuristicDive, [164](#)
- setPointers
  - CbcModel, [305](#)
- setPosition
  - CbcObject, [337](#)
- setPreProcessState
  - CbcStrategy, [404](#)
- setPreferredWay
  - CbcModel, [281](#)
  - CbcObject, [338](#)
- setPrintFrequency
  - CbcModel, [283](#)
- setPrinting
  - CbcSolver, [389](#)
- setPrintingMode
  - CbcModel, [278](#)
- setProbingAction
  - CbcGenCtlBlk, [130](#)
- setProbingInformation
  - CbcSimpleIntegerDynamicPseudoCost, [377](#)
- setProblemFeasibility
  - CbcModel, [294](#)
- setProblemStatus
  - CbcModel, [284](#)
- setProblemType
  - CbcModel, [282](#)
- setProposalActions
  - CbcHeuristicDW, [182](#)
- setRandomSeed
  - CbcModel, [300](#)
- setRange
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [427](#)
- setReadMode
  - CbcSolver, [389](#)
- setRedSplitAction
  - CbcGenCtlBlk, [131](#)
- setReducedCostMultiplier
  - CbcHeuristicFPump, [198](#)
- setRefine
  - CbcTreeLocal, [425](#)
  - CbcTreeVariable, [428](#)
- setRelativeIncrement
  - CbcHeuristicFPump, [196](#)
- setRensType
  - CbcHeuristicRENS, [229](#)
- setResolveAfterTakeOffCuts
  - CbcModel, [293](#)
- setRoundingAction
  - CbcGenCtlBlk, [132](#)
- setRowBounds
  - OsiCbcSolverInterface, [467](#)
- setRowLower
  - OsiCbcSolverInterface, [467](#)
- setRowName
  - OsiCbcSolverInterface, [466](#)
- setRowNames
  - OsiCbcSolverInterface, [466](#)
- setRowPrice
  - OsiCbcSolverInterface, [469](#)
- setRowSetBounds
  - OsiCbcSolverInterface, [468](#)
- setRowSetTypes
  - OsiCbcSolverInterface, [468](#)
- setRowType

- OsiCbcSolverInterface, [467](#)
- setRowUpper
  - OsiCbcSolverInterface, [467](#)
- setSearchStrategy
  - CbcModel, [296](#)
- setSearchType
  - CbcHeuristicLocal, [214](#)
- setSecondaryStatus
  - CbcModel, [285](#)
- setSeed
  - CbcHeuristic, [151](#)
  - CbcRounding, [358](#)
- setShallowDepth
  - CbcHeuristic, [151](#)
- setSolutionCount
  - CbcHeuristicRINS, [232](#)
  - CbcModel, [291](#)
- setSpecialOptions
  - CbcModel, [299](#)
- setSpecialOptions2
  - OsiSolverLink, [496](#)
- setSpecialOptions3
  - OsiSolverLinearizedQuadratic, [490](#)
- setState
  - CbcNode, [317](#)
- setStateOfSearch
  - CbcModel, [295](#)
- setStopNumberIterations
  - CbcModel, [293](#)
- setStoredRowCuts
  - CbcModel, [310](#)
- setStrParam
  - OsiCbcSolverInterface, [462](#)
- setStrategy
  - CbcModel, [297](#)
- setStringValue
  - CbcParam, [352](#)
- setStrongStrategy
  - CbcModel, [296](#)
- setSumDownChange
  - CbcSimpleIntegerDynamicPseudoCost, [374](#)
- setSumDownCost
  - CbcSimpleIntegerDynamicPseudoCost, [374](#)
- setSumDownDecrease
  - CbcSimpleIntegerDynamicPseudoCost, [375](#)
- setSumInfeasibilities
  - CbcNode, [316](#)
- setSumUpChange
  - CbcSimpleIntegerDynamicPseudoCost, [375](#)
- setSumUpCost
  - CbcSimpleIntegerDynamicPseudoCost, [374](#)
- setSumUpDecrease
  - CbcSimpleIntegerDynamicPseudoCost, [375](#)
- setSwitchOffIfLessThan
  - CbcCutGenerator, [87](#)
- setSwitchedOff
  - CbcCutGenerator, [87](#)
- setSwitches
  - CbcHeuristic, [149](#)
- setTarget
  - CbcFathomDynamicProgramming, [108](#)
- setTargetObjective
  - CbcHeuristicDW, [184](#)
- setTemporaryPointer
  - CbcModel, [301](#)
- setTestSolution
  - CbcModel, [289](#)
- setThreadMode
  - CbcModel, [304](#)
- setTimeLimit
  - CbcTreeLocal, [425](#)
  - CbcTreeVariable, [428](#)
- setTiming
  - CbcCutGenerator, [85](#)
- setTreeLocalAction
  - CbcGenCtlBlk, [133](#)
- setTwomirAction
  - CbcGenCtlBlk, [131](#)
- setTypeCuts
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [428](#)
- setTypePresolve
  - CbcModel, [295](#)
- setUpBounds
  - CbcIntegerBranchingObject, [242](#)
- setUpDownSeparator
  - CbcSimpleIntegerDynamicPseudoCost, [374](#)
  - CbcSimpleIntegerPseudoCost, [383](#)
- setUpDynamicPseudoCost
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
- setUpInformation
  - CbcSimpleIntegerDynamicPseudoCost, [377](#)
- setUpPseudoCost
  - CbcSimpleIntegerPseudoCost, [383](#)
- setUpShadowPrice
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
- setUseElapsedTime
  - CbcModel, [301](#)
- setUserCallBack
  - CbcSolver, [387](#)
- setWarmStart
  - OsiCbcSolverInterface, [463](#)
- setWeight
  - CbcCompareDefault, [67](#)
- setWeightFactor
  - CbcHeuristicFPump, [197](#)
- setWhatDepth
  - CbcCutGenerator, [84](#)

- setWhatDepthInSub
  - CbcCutGenerator, [84](#)
- setWhen
  - CbcHeuristic, [149](#)
- setWhenCuts
  - CbcModel, [281](#)
- setWhenInfeasible
  - CbcCutGenerator, [85](#)
- setWhereFrom
  - CbcHeuristic, [151](#)
- setWhetherCallAtEnd
  - CbcCutGenerator, [88](#)
- setWhetherInMustCallAgainMode
  - CbcCutGenerator, [88](#)
- setWhetherToUse
  - CbcCutGenerator, [88](#)
- setXMeshSize
  - OsiBiLinear, [448](#)
- setXOtherSatisfied
  - OsiBiLinear, [447](#)
- setXSatisfied
  - OsiBiLinear, [447](#)
- setXYSatisfied
  - OsiBiLinear, [448](#)
- setYMeshSize
  - OsiBiLinear, [448](#)
- setYOtherSatisfied
  - OsiBiLinear, [447](#)
- setYSatisfied
  - OsiBiLinear, [447](#)
- setupCutGenerators
  - CbcStrategy, [404](#)
  - CbcStrategyDefault, [407](#)
  - CbcStrategyDefaultSubTree, [409](#)
  - CbcStrategyNull, [411](#)
- setupForDiving\_
  - CbcCompareDefault, [69](#)
- setupHeuristics
  - CbcStrategy, [404](#)
  - CbcStrategyDefault, [407](#)
  - CbcStrategyDefaultSubTree, [409](#)
  - CbcStrategyNull, [411](#)
- setupList
  - OsiChooseStrongSubset, [475](#)
- setupOther
  - CbcStrategy, [404](#)
  - CbcStrategyDefault, [407](#)
  - CbcStrategyDefaultSubTree, [410](#)
  - CbcStrategyNull, [412](#)
- setupPreProcessing
  - CbcStrategyDefault, [407](#)
- setupPrinting
  - CbcStrategy, [404](#)
  - CbcStrategyDefault, [407](#)
- CbcStrategyDefaultSubTree, [410](#)
- CbcStrategyNull, [412](#)
- shadowPriceMode\_
  - CbcGenCtlBlk::chooseStrongCtl\_struct, [433](#)
- shallWe
  - CbcBranchToFixLots, [49](#)
- shallowDepth\_
  - CbcHeuristic, [154](#)
- shortHelp
  - CbcParam, [350](#)
- shouldHeurRun
  - CbcHeuristic, [152](#)
  - CbcHeuristicPartial, [221](#)
- shouldHeurRun\_randomChoice
  - CbcHeuristic, [152](#)
- SimpleIntegerBranchObj
  - CbcBranchingObject.hpp, [512](#)
- SimpleIntegerDynamicPseudoCostBranchObj
  - CbcBranchingObject.hpp, [512](#)
- size
  - CbcHeuristicNodeList, [219](#)
  - CbcTree, [418](#)
- size\_
  - CbcFathomDynamicProgramming, [108](#)
- sizeFingerPrint\_
  - CbcHeuristicDW, [189](#)
- sizeRowCuts
  - CbcRowCuts, [360](#)
- slack\_
  - CbcCliques, [58](#)
- smallBranchAndBound
  - CbcEventHandler, [101](#)
- smallBranchAndBound
  - CbcHeuristic, [150](#)
- SoSBranchObj
  - CbcBranchingObject.hpp, [512](#)
- solution
  - CbcEventHandler, [101](#)
  - CbcHeuristic, [148](#)
  - CbcHeuristicCrossover, [157](#)
  - CbcHeuristicDINS, [159](#)
  - CbcHeuristicDive, [164](#)
  - CbcHeuristicDW, [182](#)
  - CbcHeuristicDynamic3, [191](#)
  - CbcHeuristicFPump, [195](#)
  - CbcHeuristicGreedyCover, [202](#)
  - CbcHeuristicGreedyEquality, [205](#)
  - CbcHeuristicGreedySOS, [208](#)
  - CbcHeuristicJustOne, [211](#)
  - CbcHeuristicLocal, [214](#)
  - CbcHeuristicNaive, [216](#)
  - CbcHeuristicPartial, [220](#)
  - CbcHeuristicPivotAndFix, [223](#)
  - CbcHeuristicProximity, [224](#)

- CbcHeuristicRandRound, 227
- CbcHeuristicRENS, 229
- CbcHeuristicRINS, 231
- CbcHeuristicVND, 235
- CbcRounding, 358
- CbcSerendipity, 362
- solution2
  - CbcHeuristic, 148
- solutionFix
  - CbcHeuristicDINS, 160
  - CbcHeuristicLocal, 214
  - CbcHeuristicRINS, 231
  - CbcHeuristicVND, 235
- solve
  - CbcSolver, 387
  - CbcUser, 431
- solveFromHotStart
  - OsiCbcSolverInterface, 463
- solveState\_
  - CbcHeuristicDW, 190
- solver
  - CbcHeuristicDW, 182
  - CbcModel, 302
- solver\_
  - CbcHeuristicDW, 186
- solverBranch
  - CbcObject, 336
  - CbcSimpleInteger, 365
  - CbcSimpleIntegerDynamicPseudoCost, 372
  - CbcSOS, 393
- solverCharacteristics
  - CbcModel, 298
- sosIndices
  - ampl\_info, 26
- sosPriority
  - ampl\_info, 25
- sosPriority\_
  - CbcSolverUsefulData, 390
- sosReference
  - ampl\_info, 26
- sosStart
  - ampl\_info, 25
- sosType
  - ampl\_info, 25
  - CbcSOS, 393
- special
  - ampl\_info, 26
- specialOptions
  - CbcModel, 300
- specialOptions2
  - OsiSolverLink, 496
- specialOptions2\_
  - OsiSolverLink, 499
- specialOptions3
  - OsiSolverLinearizedQuadratic, 490
- specialOptions3\_
  - OsiSolverLinearizedQuadratic, 491
- splitModel
  - CbcModel, 305
- startBit\_
  - CbcFathomDynamicProgramming, 109
- startColumnBlock
  - CbcHeuristicDW, 185
- startColumnBlock\_
  - CbcHeuristicDW, 188
- startDive
  - CbcCompareDefault, 68
- startLower\_
  - CbcFixVariable, 115
- startNodeNumber\_
  - CbcCompareDefault, 68
- startNonLinear\_
  - OsiSolverLink, 499
- startRowBlock\_
  - CbcHeuristicDW, 188
- startSplitModel
  - CbcModel, 305
- startTime
  - CbcSolver, 389
- startTime\_
  - CbcHeuristicFPump, 199
- startUpper\_
  - CbcFixVariable, 115
- startingInfeasibility
  - CbcStatistics, 399
- startingInfeasibility\_
  - CbcStatistics, 400
- startingObjective
  - CbcStatistics, 399
- startingObjective\_
  - CbcStatistics, 399
- starts
  - ampl\_info, 25
- stateOfFixing\_
  - CbcHeuristicRINS, 232
- stateOfSearch
  - CbcModel, 295
- states\_
  - CbcFixVariable, 115
- status
  - CbcModel, 284
  - CbcStrategy, 405
  - OsiCbcSolverInterface, 472
- status\_
  - CbcObjectUpdateData, 340
- stepLength
  - ClpAmplObjective, 435
- stepSize\_

- CbcHeuristicVND, [236](#)
- stop
  - CbcEventHandler, [101](#)
- storedRowCuts
  - CbcModel, [309](#)
- strategy
  - CbcModel, [296](#)
- stringValue
  - CbcParam, [352](#)
- strongInfo
  - CbcModel, [309](#)
- strongStrategy
  - CbcModel, [296](#)
- stuff
  - CbcUser, [430](#)
- subTreeModel
  - CbcModel, [294](#)
- sumChangeObjective
  - CbcModel, [293](#)
- sumDownChange
  - CbcSimpleIntegerDynamicPseudoCost, [374](#)
- sumDownChange\_
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
- sumDownCost
  - CbcSimpleIntegerDynamicPseudoCost, [374](#)
- sumDownCost\_
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
- sumDownDecrease
  - CbcSimpleIntegerDynamicPseudoCost, [375](#)
- sumDownDecrease\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- sumInfeasibilities
  - CbcNode, [316](#)
- sumUpChange
  - CbcSimpleIntegerDynamicPseudoCost, [375](#)
- sumUpChange\_
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
- sumUpCost
  - CbcSimpleIntegerDynamicPseudoCost, [374](#)
- sumUpCost\_
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
- sumUpDecrease
  - CbcSimpleIntegerDynamicPseudoCost, [375](#)
- sumUpDecrease\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- swap\_
  - CbcHeuristicLocal, [214](#)
- swapSolver
  - CbcModel, [302](#)
- switchOffIfLessThan
  - CbcCutGenerator, [87](#)
- switchedOff
  - CbcCutGenerator, [87](#)
- switches
  - CbcCutGenerator, [84](#)
  - CbcHeuristic, [149](#)
- switches\_
  - CbcHeuristic, [153](#)
- synchronizeHandlers
  - CbcModel, [305](#)
- synchronizeModel
  - CbcModel, [276](#)
- synchronizeNumberBeforeTrust
  - CbcModel, [307](#)
- T
- TIGHTEN
  - CbcOsiParam, [344](#)
- TIGHTENFACTOR
  - CbcGenParam, [142](#)
- TIMELIMIT\_BAB
  - CbcCbcParam, [52](#)
- TWOMIRCUTS
  - CbcGenParam, [142](#)
- TYPE2
  - CbcSimpleIntegerDynamicPseudoCost.hpp, [539](#)
- TYPERATIO
  - CbcSimpleIntegerDynamicPseudoCost.hpp, [539](#)
- takeAction
  - CbcEventHandler, [101](#)
- takeOffCuts
  - CbcModel, [306](#)
- target
  - CbcFathomDynamicProgramming, [108](#)
- target\_
  - CbcFathomDynamicProgramming, [109](#)
- targetObjective\_
  - CbcHeuristicDW, [185](#)
- temporaryPointer
  - CbcModel, [301](#)
- test
  - CbcCompareBase, [64](#)
  - CbcCompareDefault, [67](#)
  - CbcCompareDepth, [70](#)
  - CbcCompareEstimate, [71](#)
  - CbcCompareObjective, [72](#)
- test\_
  - CbcCompare, [61](#)
  - CbcCompareBase, [64](#)
- testSolution
  - CbcModel, [289](#)
- threaded\_
  - CbcCompareBase, [64](#)
- threshold\_
  - CbcGenCtlBlk::djFixCtl\_struct, [440](#)
- throwAway
  - CbcNodeInfo, [323](#)
- tighten

- CbcBranchingObject, [44](#)
- CbcIntegerBranchingObject, [241](#)
- tightenVubs
  - CbcModel, [275](#)
- timeInCutGenerator
  - CbcCutGenerator, [85](#)
- timeLimit
  - CbcTreeLocal, [424](#)
  - CbcTreeVariable, [428](#)
- timing
  - CbcCutGenerator, [85](#)
- top
  - CbcTree, [418](#)
  - CbcTreeLocal, [423](#)
  - CbcTreeVariable, [427](#)
- topOfTree
  - CbcModel, [282](#)
- totalTime\_
  - CbcGenCtlBlk, [136](#)
- translateMajor
  - CbcGenCtlBlk, [133](#)
- translateMinor
  - CbcGenCtlBlk, [133](#)
- tree
  - CbcModel, [294](#)
- treeStatus
  - CbcEventHandler, [101](#)
- treeSize\_
  - CbcCompareDefault, [68](#)
- tryColumn
  - CbcFathomDynamicProgramming, [108](#)
- twoWay
  - CbcNWayBranchingObject, [331](#)
- type
  - CbcBranchingObject, [46](#)
  - CbcClique, [57](#)
  - CbcCliqueBranchingObject, [60](#)
  - CbcCutBranchingObject, [78](#)
  - CbcDummyBranchingObject, [95](#)
  - CbcDynamicPseudoCostBranchingObject, [98](#)
  - CbcFixingBranchingObject, [112](#)
  - CbcIntegerBranchingObject, [242](#)
  - CbcIntegerPseudoCostBranchingObject, [245](#)
  - CbcLongCliqueBranchingObject, [247](#)
  - CbcLotsizeBranchingObject, [253](#)
  - CbcNWayBranchingObject, [331](#)
  - CbcParam, [352](#)
  - CbcSOSBranchingObject, [396](#)
- type\_
  - CbcClique, [57](#)
  - CbcFathomDynamicProgramming, [108](#)
  - OsiUsesBiLinear, [503](#)
- typeCuts
  - CbcTreeLocal, [424](#)
- CbcTreeVariable, [428](#)
- typePresolve
  - CbcModel, [294](#)
- U
- UNITTEST
  - CbcGenParam, [142](#)
- USERCBC
  - CbcGenParam, [142](#)
- USESOLUTION
  - CbcGenParam, [142](#)
- unlockThread
  - CbcModel, [304](#)
- unmark
  - CbcNodeInfo, [325](#)
- unmarkHotStart
  - OsiCbcSolverInterface, [463](#)
- unsetParentBasedData
  - CbcNodeInfo, [325](#)
- up\_
  - CbcCutBranchingObject, [79](#)
  - CbcIntegerBranchingObject, [242](#)
  - CbcLotsizeBranchingObject, [254](#)
  - CbcRounding, [359](#)
- upArray\_
  - CbcHeuristicDive, [166](#)
- upBounds
  - CbcIntegerBranchingObject, [241](#)
- upDownSeparator
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
  - CbcSimpleIntegerPseudoCost, [383](#)
- upDownSeparator\_
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
  - CbcSimpleIntegerPseudoCost, [384](#)
- upDynamicPseudoCost
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
- upDynamicPseudoCost\_
  - CbcSimpleIntegerDynamicPseudoCost, [378](#)
- upEstimate
  - CbcSimpleIntegerDynamicPseudoCost, [377](#)
  - CbcSimpleIntegerPseudoCost, [383](#)
- upLocks\_
  - CbcHeuristicDive, [166](#)
- upMovement
  - CbcStrongInfo, [413](#)
- upPseudoCost
  - CbcSimpleIntegerPseudoCost, [383](#)
- upPseudoCost\_
  - CbcSimpleIntegerPseudoCost, [384](#)
- upShadowPrice
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
- upShadowPrice\_
  - CbcSimpleIntegerDynamicPseudoCost, [379](#)
- updateAfter

- CbcSimpleIntegerDynamicPseudoCost, [372](#)
- updateAfterMini
  - CbcSimpleIntegerDynamicPseudoCost, [372](#)
- updateBefore
  - CbcSimpleIntegerDynamicPseudoCost, [372](#)
- updateBounds
  - OsiLinkedBound, [480](#)
- updateCoefficients
  - OsiBiLinear, [448](#)
  - OsiSolverLink, [496](#)
- updateDownDynamicPseudoCost
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
- updateInfeasibility
  - CbcStatistics, [398](#)
- updateInformation
  - CbcBranchDecision, [35](#)
  - CbcBranchDynamicDecision, [40](#)
  - CbcObject, [337](#)
  - CbcSimpleIntegerDynamicPseudoCost, [372](#)
  - CbcSOS, [393](#)
- updateModel
  - CbcSolver, [387](#)
- updateUpDynamicPseudoCost
  - CbcSimpleIntegerDynamicPseudoCost, [373](#)
- upper
  - CbcFullNodeInfo, [120](#)
- upper\_
  - CbcFullNodeInfo, [121](#)
- useElapsedTime
  - CbcModel, [301](#)
- useNumber\_
  - CbcHeuristicCrossover, [157](#)
- used
  - CbcHeuristicLocal, [214](#)
  - CbcHeuristicProximity, [225](#)
  - CbcHeuristicRINS, [231](#)
- used\_
  - CbcHeuristicLocal, [215](#)
  - CbcHeuristicProximity, [225](#)
  - CbcHeuristicRINS, [232](#)
- usedInSolution
  - CbcModel, [289](#)
- usefullInformation
  - CbcModel, [310](#)
- userFunction
  - CbcSolver, [388](#)
- userFunctionArray
  - CbcSolver, [388](#)
- userName\_
  - CbcUser, [431](#)
- V
- VERBOSE
  - CbcGenParam, [142](#)
- validate
  - CbcHeuristic, [149](#)
  - CbcHeuristicDive, [164](#)
  - CbcHeuristicGreedyCover, [202](#)
  - CbcHeuristicGreedyEquality, [205](#)
  - CbcHeuristicGreedySOS, [208](#)
  - CbcHeuristicJustOne, [212](#)
  - CbcHeuristicPartial, [221](#)
  - CbcRounding, [358](#)
- value
  - CbcStatistics, [399](#)
- value\_
  - CbcStatistics, [399](#)
- values\_
  - CbcGenCtIBlk::debugSolInfo\_struct, [440](#)
  - CbcHeuristicDINS, [161](#)
- var
  - PseudoReducedCost, [503](#)
- variable
  - CbcBranchingObject, [45](#)
  - OsiLinkedBound, [480](#)
- variable\_
  - CbcBranchingObject, [46](#)
  - CbcFixVariable, [115](#)
- variables
  - CbcPartialNodeInfo, [355](#)
- variables\_
  - CbcPartialNodeInfo, [356](#)
- vbRowIndex\_
  - CbcHeuristicDive, [166](#)
- verbose\_
  - CbcGenCtIBlk, [135](#)
- version\_
  - CbcGenCtIBlk, [134](#)
- virginCutGenerator
  - CbcModel, [296](#)
- W
- WEIGHT\_AFTER
  - CbcSimpleIntegerDynamicPseudoCost.hpp, [539](#)
- WEIGHT\_BEFORE
  - CbcSimpleIntegerDynamicPseudoCost.hpp, [540](#)
- WEIGHT\_PRODUCT
  - CbcSimpleIntegerDynamicPseudoCost.hpp, [540](#)
- waitingForMiniBranchAndBound
  - CbcModel, [300](#)
- walkback
  - CbcModel, [304](#)
- way
  - CbcBranchingObject, [45](#)
  - CbcNode, [316](#)
  - CbcStatistics, [399](#)
- way\_
  - CbcBranchingObject, [46](#)



- CbcObjectUpdateData, 340
- CbcStatistics, 400
- weight\_
  - CbcCompareDefault, 68
- weightFactor
  - CbcHeuristicFPump, 197
- weightFactor\_
  - CbcHeuristicFPump, 199
- weights
  - CbcSOS, 394
- weights\_
  - CbcHeuristicDW, 187
- whatDepth
  - CbcCutGenerator, 84
- whatDepthInSub
  - CbcCutGenerator, 84
- when
  - CbcHeuristic, 149
- when\_
  - CbcHeuristic, 153
- whenCuts
  - CbcModel, 281
- whenInfeasible
  - CbcCutGenerator, 85
- where\_
  - CbcGenCtIBlk::babState\_struct, 27
- whereFrom
  - CbcHeuristic, 151
- whereFrom\_
  - CbcHeuristic, 153
- whetherCallAtEnd
  - CbcCutGenerator, 88
- whetherInMustCallAgainMode
  - CbcCutGenerator, 88
- whetherToUse
  - CbcCutGenerator, 88
- which\_
  - CbcBranchAllDifferent, 29
- whichColumnBlock
  - CbcHeuristicDW, 184
- whichColumnBlock\_
  - CbcHeuristicDW, 187
- whichCutGenerator
  - CbcCountRowCut, 76
- whichGenerator
  - CbcModel, 274
- whichMethod
  - CbcBranchDecision, 34
  - CbcBranchDynamicDecision, 40
- whichNonLinear\_
  - OsiSolverLink, 499
- whichRowBlock
  - CbcHeuristicDW, 184
- whichRowBlock\_

- CbcHeuristicDW, 187
- workingBasis
  - CbcModel, 293
- writeAmpl
  - Cbc\_ampl.h, 504
- writeMps
  - OsiCbcSolverInterface, 471
- writeMpsNative
  - OsiCbcSolverInterface, 471

## X

- xColumn
  - OsiBiLinear, 446
- xColumn\_
  - OsiBiLinear, 450
  - OsiOneLink, 485
- xMeshSize
  - OsiBiLinear, 447
- xMeshSize\_
  - OsiBiLinear, 449
- xOtherSatisfied
  - OsiBiLinear, 447
- xOtherSatisfied\_
  - OsiBiLinear, 450
- xRow
  - OsiBiLinear, 446
- xRow\_
  - OsiBiLinear, 451
  - OsiOneLink, 485
- xSatisfied
  - OsiBiLinear, 447
- xSatisfied\_
  - OsiBiLinear, 449
- xyBranchValue\_
  - OsiBiLinear, 450
- xyCoefficient
  - OsiBiLinear, 449
- xyRow
  - OsiBiLinear, 446
  - OsiOneLink, 485
- xyRow\_
  - OsiBiLinear, 451
- xySatisfied
  - OsiBiLinear, 448
- xySatisfied\_
  - OsiBiLinear, 450

## Y

- yColumn
  - OsiBiLinear, 446
- yColumn\_
  - OsiBiLinear, 450
- yMeshSize
  - OsiBiLinear, 448
- yMeshSize\_



OsiBiLinear, [449](#)  
yOtherSatisfied  
    OsiBiLinear, [447](#)  
yOtherSatisfied\_  
    OsiBiLinear, [450](#)  
yRow  
    OsiBiLinear, [446](#)  
yRow\_  
    OsiBiLinear, [451](#)  
ySatisfied  
    OsiBiLinear, [447](#)  
ySatisfied\_  
    OsiBiLinear, [450](#)  
  
Z  
zapIntegerInformation  
    CbcModel, [307](#)