

Cgl  
0.59

Generated by Doxygen 1.8.9.1

Thu Oct 8 2015 22:41:49



# Contents

<b>1</b>	<b>Namespace Index</b>	<b>1</b>
1.1	Namespace List . . . . .	1
<b>2</b>	<b>Hierarchical Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>11</b>
3.1	Class List . . . . .	11
<b>4</b>	<b>File Index</b>	<b>15</b>
4.1	File List . . . . .	15
<b>5</b>	<b>Namespace Documentation</b>	<b>17</b>
5.1	LAP Namespace Reference . . . . .	17
5.1.1	Detailed Description . . . . .	18
5.1.2	Enumeration Type Documentation . . . . .	18
5.1.2.1	LAP_messages . . . . .	18
5.1.3	Function Documentation . . . . .	18
5.1.3.1	normCoef . . . . .	18
5.1.3.2	modularizeRow . . . . .	18
<b>6</b>	<b>Class Documentation</b>	<b>19</b>
6.1	auxiliary_graph Struct Reference . . . . .	19
6.2	Cgl012Cut Class Reference . . . . .	19
6.2.1	Detailed Description . . . . .	20
6.3	cgl_arc Struct Reference . . . . .	20
6.3.1	Detailed Description . . . . .	20
6.4	cgl_graph Struct Reference . . . . .	20
6.4.1	Detailed Description . . . . .	20
6.5	cgl_node Struct Reference . . . . .	20

6.5.1	Detailed Description	20
6.6	CglAllDifferent Class Reference	20
6.6.1	Detailed Description	21
6.6.2	Member Function Documentation	22
6.6.2.1	mayGenerateRowCutsInTree	22
6.7	CglBK Class Reference	22
6.7.1	Detailed Description	22
6.8	CglClique Class Reference	23
6.8.1	Detailed Description	25
6.8.2	Constructor & Destructor Documentation	25
6.8.2.1	CglClique	25
6.8.3	Member Function Documentation	25
6.8.3.1	generateCuts	25
6.8.4	Friends And Related Function Documentation	25
6.8.4.1	CglCliqueUnitTest	25
6.8.5	Member Data Documentation	26
6.8.5.1	node_node	26
6.8.5.2	petol	26
6.8.5.3	do_row_clique	26
6.8.5.4	do_star_clique	26
6.8.5.5	scl_candidate_length_threshold	26
6.8.5.6	rcl_candidate_length_threshold	26
6.8.5.7	cl_perm_indices	26
6.8.5.8	cl_indices	27
6.8.5.9	cl_del_indices	27
6.9	CglCutGenerator Class Reference	27
6.9.1	Detailed Description	28
6.9.2	Member Function Documentation	28
6.9.2.1	generateCuts	28
6.9.2.2	generateCpp	28
6.9.2.3	getAggressiveness	29
6.9.2.4	setAggressiveness	29
6.9.2.5	mayGenerateRowCutsInTree	29
6.9.3	Member Data Documentation	29
6.9.3.1	aggressive_	29
6.10	CglDuplicateRow Class Reference	29
6.10.1	Detailed Description	31

6.10.2	Member Function Documentation	31
6.10.2.1	generateCuts	31
6.10.2.2	outDuplicates	32
6.11	CglFakeClique Class Reference	32
6.11.1	Detailed Description	33
6.11.2	Constructor & Destructor Documentation	33
6.11.2.1	CglFakeClique	33
6.11.3	Member Function Documentation	33
6.11.3.1	generateCuts	33
6.12	CglFlowCover Class Reference	34
6.12.1	Detailed Description	35
6.12.2	Member Function Documentation	35
6.12.2.1	flowPreprocess	35
6.12.2.2	generateCuts	35
6.12.3	Friends And Related Function Documentation	35
6.12.3.1	CglFlowCoverUnitTest	35
6.13	CglFlowVUB Class Reference	35
6.13.1	Detailed Description	36
6.13.2	Constructor & Destructor Documentation	36
6.13.2.1	CglFlowVUB	36
6.13.3	Member Data Documentation	36
6.13.3.1	upper_	36
6.14	CglGMI Class Reference	36
6.14.1	Detailed Description	38
6.14.2	Member Function Documentation	38
6.14.2.1	generateCuts	38
6.14.2.2	setTrackRejection	38
6.14.3	Friends And Related Function Documentation	38
6.14.3.1	CglGMIUnitTest	38
6.15	CglGMIParam Class Reference	39
6.15.1	Detailed Description	42
6.15.2	Member Function Documentation	42
6.15.2.1	setInfinity	42
6.15.2.2	setEps	43
6.15.2.3	setEpsCoeff	43
6.15.2.4	setMaxSupport	43
6.15.2.5	setMINVIOL	43

6.15.2.6	setMAX_SUPPORT_REL	43
6.15.2.7	setUSE_INTSLACKS	43
6.15.2.8	setCHECK_DUPLICATES	43
6.15.2.9	setCLEAN_PROC	43
6.15.2.10	getCLEAN_PROC	44
6.15.2.11	setINTEGRAL_SCALE_CONT	44
6.15.2.12	getINTEGRAL_SCALE_CONT	44
6.15.2.13	setENFORCE_SCALING	44
6.15.2.14	getENFORCE_SCALING	44
6.15.3	Member Data Documentation	44
6.15.3.1	AWAY	44
6.15.3.2	EPS_ELIM	44
6.15.3.3	EPS_RELAX_ABS	44
6.15.3.4	EPS_RELAX_REL	45
6.15.3.5	MAXDYN	45
6.15.3.6	MINVIOL	45
6.15.3.7	MAX_SUPPORT_REL	45
6.15.3.8	USE_INTSLACKS	45
6.15.3.9	INTEGRAL_SCALE_CONT	45
6.15.3.10	ENFORCE_SCALING	45
6.16	CglGomory Class Reference	46
6.16.1	Detailed Description	48
6.16.2	Member Function Documentation	48
6.16.2.1	generateCuts	48
6.16.3	Friends And Related Function Documentation	48
6.16.3.1	CglGomoryUnitTest	48
6.17	CglHashLink Struct Reference	48
6.17.1	Detailed Description	48
6.18	CglImplication Class Reference	49
6.18.1	Detailed Description	49
6.19	CglKnapsackCover Class Reference	50
6.19.1	Detailed Description	51
6.19.2	Member Function Documentation	51
6.19.2.1	generateCuts	51
6.19.2.2	createCliques	51
6.19.3	Friends And Related Function Documentation	51
6.19.3.1	CglKnapsackCoverUnitTest	51

6.20 CglLandP Class Reference . . . . .	51
6.20.1 Detailed Description . . . . .	53
6.20.2 Member Enumeration Documentation . . . . .	53
6.20.2.1 SelectionRules . . . . .	53
6.20.2.2 ExtraCutsMode . . . . .	53
6.20.2.3 SeparationSpaces . . . . .	53
6.20.2.4 RhsWeightType . . . . .	53
6.20.3 Member Function Documentation . . . . .	54
6.20.3.1 generateCuts . . . . .	54
6.20.3.2 setLogLevel . . . . .	54
6.21 LAP::CglLandPSimplex Class Reference . . . . .	54
6.21.1 Detailed Description . . . . .	56
6.21.2 Member Function Documentation . . . . .	56
6.21.2.1 generateMig . . . . .	56
6.21.2.2 generateExtraCuts . . . . .	57
6.21.2.3 generateExtraCut . . . . .	57
6.21.2.4 insertAllExtr . . . . .	57
6.21.2.5 fastFindCutImprovingPivotRow . . . . .	57
6.21.2.6 fastFindBestPivotColumn . . . . .	57
6.21.2.7 findBestPivot . . . . .	57
6.21.2.8 computeCglpObjective . . . . .	57
6.21.2.9 strengthenedIntersectionCutCoef . . . . .	58
6.21.2.10 normalizationFactor . . . . .	58
6.21.2.11 scaleCut . . . . .	58
6.21.2.12 createMIG . . . . .	58
6.21.2.13 getStatus . . . . .	58
6.21.2.14 isInteger . . . . .	58
6.21.2.15 computeWeights . . . . .	58
6.21.2.16 normedCoef . . . . .	58
6.21.2.17 printTableau . . . . .	58
6.21.2.18 printEverything . . . . .	59
6.21.2.19 printTableauLateX . . . . .	59
6.21.2.20 printCglpBasis . . . . .	59
6.21.2.21 get_M1_M2_M3 . . . . .	59
6.21.2.22 eliminate_slacks . . . . .	59
6.21.2.23 findCutImprovingPivotRow . . . . .	59
6.21.2.24 findBestPivotColumn . . . . .	59

6.22 CglLiftAndProject Class Reference . . . . .	59
6.22.1 Detailed Description . . . . .	60
6.22.2 Member Function Documentation . . . . .	60
6.22.2.1 generateCuts . . . . .	60
6.22.2.2 setBeta . . . . .	60
6.22.3 Friends And Related Function Documentation . . . . .	61
6.22.3.1 CglLiftAndProjectUnitTest . . . . .	61
6.23 CglMessage Class Reference . . . . .	61
6.23.1 Detailed Description . . . . .	61
6.24 CglMixedIntegerRounding Class Reference . . . . .	61
6.24.1 Detailed Description . . . . .	62
6.24.2 Member Function Documentation . . . . .	63
6.24.2.1 generateCuts . . . . .	63
6.25 CglMixedIntegerRounding2 Class Reference . . . . .	63
6.25.1 Detailed Description . . . . .	64
6.25.2 Member Function Documentation . . . . .	64
6.25.2.1 generateCuts . . . . .	64
6.26 CglMixIntRoundVUB Class Reference . . . . .	64
6.26.1 Detailed Description . . . . .	64
6.27 CglMixIntRoundVUB2 Class Reference . . . . .	65
6.27.1 Detailed Description . . . . .	65
6.28 CglOddHole Class Reference . . . . .	65
6.28.1 Detailed Description . . . . .	66
6.28.2 Member Function Documentation . . . . .	66
6.28.2.1 generateCuts . . . . .	66
6.28.3 Friends And Related Function Documentation . . . . .	67
6.28.3.1 CglOddHoleUnitTest . . . . .	67
6.29 CglParam Class Reference . . . . .	67
6.29.1 Detailed Description . . . . .	68
6.30 CglPreProcess Class Reference . . . . .	68
6.30.1 Detailed Description . . . . .	70
6.30.2 Member Function Documentation . . . . .	71
6.30.2.1 preProcess . . . . .	71
6.30.2.2 preProcessNonDefault . . . . .	71
6.30.2.3 tightenPrimalBounds . . . . .	71
6.30.2.4 someFixed . . . . .	71
6.30.2.5 cliquelt . . . . .	71



6.30.2.6	<a href="#">setCutoff</a>	71
6.30.2.7	<a href="#">originalColumns</a>	72
6.30.2.8	<a href="#">originalRows</a>	72
6.30.2.9	<a href="#">rowTypes</a>	72
6.30.2.10	<a href="#">setApplicationData</a>	72
6.31	<a href="#">CglProbing Class Reference</a>	72
6.31.1	<a href="#">Detailed Description</a>	75
6.31.2	<a href="#">Member Function Documentation</a>	75
6.31.2.1	<a href="#">generateCuts</a>	75
6.31.2.2	<a href="#">snapshot</a>	76
6.31.2.3	<a href="#">createCliques</a>	76
6.31.2.4	<a href="#">mayGenerateRowCutsInTree</a>	76
6.31.3	<a href="#">Friends And Related Function Documentation</a>	77
6.31.3.1	<a href="#">CglProbingUnitTest</a>	77
6.32	<a href="#">CglRedSplit Class Reference</a>	77
6.32.1	<a href="#">Detailed Description</a>	79
6.32.2	<a href="#">Member Function Documentation</a>	79
6.32.2.1	<a href="#">generateCuts</a>	79
6.32.2.2	<a href="#">set_given_optsol</a>	79
6.32.2.3	<a href="#">setMaxTab</a>	79
6.32.3	<a href="#">Friends And Related Function Documentation</a>	80
6.32.3.1	<a href="#">CglRedSplitUnitTest</a>	80
6.33	<a href="#">CglRedSplit2 Class Reference</a>	80
6.33.1	<a href="#">Detailed Description</a>	81
6.33.2	<a href="#">Member Function Documentation</a>	81
6.33.2.1	<a href="#">generateCuts</a>	81
6.33.3	<a href="#">Friends And Related Function Documentation</a>	82
6.33.3.1	<a href="#">CglRedSplit2UnitTest</a>	82
6.34	<a href="#">CglRedSplit2Param Class Reference</a>	82
6.34.1	<a href="#">Detailed Description</a>	86
6.34.2	<a href="#">Member Enumeration Documentation</a>	87
6.34.2.1	<a href="#">RowSelectionStrategy</a>	87
6.34.2.2	<a href="#">ColumnSelectionStrategy</a>	87
6.34.3	<a href="#">Constructor &amp; Destructor Documentation</a>	88
6.34.3.1	<a href="#">CglRedSplit2Param</a>	88
6.34.3.2	<a href="#">CglRedSplit2Param</a>	88
6.34.4	<a href="#">Member Function Documentation</a>	88

6.34.4.1	setMINVIOL	88
6.34.4.2	setMAX_SUPP_ABS	88
6.34.4.3	setMAX_SUPP_REL	88
6.34.4.4	setUSE_INTSLACKS	88
6.34.4.5	addNumRowsReduction	89
6.34.4.6	addNumRowsReductionLAP	89
6.34.4.7	setSkipGomory	89
6.34.5	Member Data Documentation	89
6.34.5.1	EPS_ELIM	89
6.34.5.2	EPS_RELAX_ABS	89
6.34.5.3	EPS_RELAX_REL	89
6.34.5.4	MINVIOL	89
6.34.5.5	normIsZero	90
6.34.5.6	away	90
6.35	CglRedSplitParam Class Reference	90
6.35.1	Detailed Description	92
6.35.2	Member Function Documentation	93
6.35.2.1	setMINVIOL	93
6.35.2.2	setUSE_INTSLACKS	93
6.35.2.3	setUSE_CG2	93
6.35.2.4	setMaxTab	93
6.35.3	Member Data Documentation	93
6.35.3.1	LUB	93
6.35.3.2	EPS_ELIM	94
6.35.3.3	EPS_RELAX_ABS	94
6.35.3.4	EPS_RELAX_REL	94
6.35.3.5	EPS_COEFF_LUB	94
6.35.3.6	MINVIOL	94
6.35.3.7	USE_CG2	94
6.35.3.8	normIsZero	94
6.35.3.9	minReduc	95
6.35.3.10	away	95
6.35.3.11	maxTab	95
6.36	CglResidualCapacity Class Reference	95
6.36.1	Detailed Description	96
6.36.2	Member Function Documentation	96
6.36.2.1	generateCuts	96

6.36.3 Friends And Related Function Documentation . . . . .	97
6.36.3.1 CglResidualCapacityUnitTest . . . . .	97
6.37 CglSimpleRounding Class Reference . . . . .	97
6.37.1 Detailed Description . . . . .	98
6.37.2 Member Function Documentation . . . . .	98
6.37.2.1 generateCuts . . . . .	98
6.37.3 Friends And Related Function Documentation . . . . .	98
6.37.3.1 CglSimpleRoundingUnitTest . . . . .	98
6.38 CglStored Class Reference . . . . .	98
6.38.1 Detailed Description . . . . .	100
6.38.2 Member Function Documentation . . . . .	100
6.38.2.1 generateCuts . . . . .	100
6.39 CglTreeInfo Class Reference . . . . .	100
6.39.1 Detailed Description . . . . .	101
6.39.2 Member Data Documentation . . . . .	102
6.39.2.1 formulation_rows . . . . .	102
6.39.2.2 options . . . . .	102
6.39.2.3 strengthenRow . . . . .	102
6.40 CglTreeProbingInfo Class Reference . . . . .	102
6.40.1 Detailed Description . . . . .	104
6.41 CglTwomir Class Reference . . . . .	104
6.41.1 Detailed Description . . . . .	106
6.41.2 Friends And Related Function Documentation . . . . .	106
6.41.2.1 CglTwomirUnitTest . . . . .	106
6.42 CglUniqueRowCuts Class Reference . . . . .	106
6.42.1 Detailed Description . . . . .	106
6.43 CglZeroHalf Class Reference . . . . .	106
6.43.1 Detailed Description . . . . .	107
6.43.2 Member Function Documentation . . . . .	107
6.43.2.1 generateCuts . . . . .	107
6.43.3 Friends And Related Function Documentation . . . . .	108
6.43.3.1 CglZeroHalfUnitTest . . . . .	108
6.44 CliqueEntry Struct Reference . . . . .	108
6.44.1 Detailed Description . . . . .	108
6.45 CglProbing::CliqueType Struct Reference . . . . .	108
6.45.1 Detailed Description . . . . .	108
6.46 cut Struct Reference . . . . .	108

6.46.1 Detailed Description	108
6.47 cut_list Struct Reference	109
6.47.1 Detailed Description	109
6.48 cutParams Struct Reference	109
6.48.1 Detailed Description	109
6.49 LAP::Cuts Struct Reference	109
6.49.1 Detailed Description	110
6.49.2 Constructor & Destructor Documentation	110
6.49.2.1 ~Cuts	110
6.49.3 Member Function Documentation	110
6.49.3.1 insert	110
6.49.3.2 numberCuts	110
6.49.3.3 resize	110
6.50 cycle Struct Reference	110
6.50.1 Detailed Description	110
6.51 cycle_list Struct Reference	111
6.51.1 Detailed Description	111
6.52 DGG_constraint_t Struct Reference	111
6.52.1 Detailed Description	111
6.53 DGG_data_t Struct Reference	111
6.53.1 Detailed Description	111
6.54 DGG_list_t Struct Reference	111
6.54.1 Detailed Description	111
6.55 disaggregationAction Struct Reference	112
6.55.1 Detailed Description	112
6.56 edge Struct Reference	112
6.56.1 Detailed Description	112
6.57 ilp Struct Reference	112
6.57.1 Detailed Description	112
6.58 info_weak Struct Reference	112
6.58.1 Detailed Description	112
6.59 LAP::LandPMessages Class Reference	113
6.59.1 Detailed Description	113
6.60 LAP::LapMessages Class Reference	113
6.60.1 Detailed Description	113
6.60.2 Constructor & Destructor Documentation	113
6.60.2.1 ~LapMessages	114

6.61	log_var Struct Reference	114
6.61.1	Detailed Description	114
6.62	CglLandP::NoBasisError Class Reference	114
6.62.1	Detailed Description	114
6.63	CglLandP::Parameters Class Reference	114
6.63.1	Detailed Description	116
6.63.2	Member Data Documentation	116
6.63.2.1	pivotLimitInTree	116
6.63.2.2	extraCutsLimit	116
6.63.2.3	timeLimit	116
6.63.2.4	singleCutTimeLimit	116
6.63.2.5	rhsWeight	116
6.63.2.6	countMistakenRc	116
6.63.2.7	perturb	117
6.63.2.8	normalization	117
6.63.2.9	rhsWeightType	117
6.63.2.10	lhs_norm	117
6.63.2.11	generateExtraCuts	117
6.63.2.12	pivotSelection	117
6.64	parity_ilp Struct Reference	117
6.64.1	Detailed Description	117
6.65	pool_cut Struct Reference	118
6.65.1	Detailed Description	118
6.66	pool_cut_list Struct Reference	118
6.66.1	Detailed Description	118
6.67	select_cut Struct Reference	118
6.67.1	Detailed Description	118
6.68	separation_graph Struct Reference	118
6.68.1	Detailed Description	118
6.69	short_path_node Struct Reference	118
6.69.1	Detailed Description	119
6.70	CglLandP::SimplexInterfaceError Class Reference	119
6.70.1	Detailed Description	119
6.71	LAP::TabRow Struct Reference	119
6.71.1	Detailed Description	119
6.71.2	Member Data Documentation	119
6.71.2.1	num	119

6.71.2.2	<a href="#">rhs</a>	120
6.71.2.3	<a href="#">modularized_</a>	120
6.72	<a href="#">LAP::Validator Class Reference</a>	120
6.72.1	<a href="#">Detailed Description</a>	121
6.72.2	<a href="#">Member Enumeration Documentation</a>	121
6.72.2.1	<a href="#">RejectionsReasons</a>	121
<b>Index</b>		<b>123</b>

# Chapter 1

## Namespace Index

### 1.1 Namespace List

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

<a href="#">LAP</a>	Performs one round of Lift & Project using <a href="#">CgLLandPSimplex</a> to build cuts . . . . .	<a href="#">17</a>
---------------------	--	--------------------





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

<code>_EKKfactinfo</code>	<code>[external]</code>	
<code>doubleton_action</code>	<code>::action</code>	<code>[external]</code>
<code>forcing_constraint_action</code>	<code>::action</code>	<code>[external]</code>
<code>tripletion_action</code>	<code>::action</code>	<code>[external]</code>
<code>remove_fixed_action</code>	<code>::action</code>	<code>[external]</code>
<code>std::allocator</code>	<code>&lt; T &gt;</code>	
<code>std::array</code>	<code>&lt; T &gt;</code>	
<code>std::auto_ptr</code>	<code>&lt; T &gt;</code>	
<code>auxiliary_graph</code>		19
<code>std::basic_string</code>	<code>&lt; Char &gt;</code>	
<code>std::string</code>		
<code>std::wstring</code>		
<code>std::basic_string</code>	<code>&lt; char &gt;</code>	
<code>std::basic_string</code>	<code>&lt; wchar_t &gt;</code>	
<code>std::bitset</code>	<code>&lt; Bits &gt;</code>	
<code>BitVector128</code>	<code>[external]</code>	
<code>Cgl012Cut</code>		19
<code>cgl_arc</code>		20
<code>cgl_graph</code>		20
<code>cgl_node</code>		20
<code>CglBK</code>		22
<code>CglCutGenerator</code>		27
<code>CglAllDifferent</code>		20
<code>CglClique</code>		23
<code>CglFakeClique</code>		32
<code>CglDuplicateRow</code>		29
<code>CglFlowCover</code>		34
<code>CglGMI</code>		36
<code>CglGomory</code>		46
<code>CglImplication</code>		49
<code>CglKnapsackCover</code>		50
<code>CglLandP</code>		51
<code>CglLiftAndProject</code>		59

CglMixedIntegerRounding	61
CglMixedIntegerRounding2	63
CglOddHole	65
CglProbing	72
CglRedSplit	77
CglRedSplit2	80
CglResidualCapacity	95
CglSimpleRounding	97
CglStored	98
CglTwomir	104
CglZeroHalf	106
CglFlowVUB	35
CglHashLink	48
LAP::CglLandPSimplex	54
CglMixIntRoundVUB	64
CglMixIntRoundVUB2	65
CglParam	67
CglGMIParam	39
CglLandP::Parameters	114
CglRedSplit2Param	82
CglRedSplitParam	90
CglPreProcess	68
CglTreeInfo	100
CglTreeProbingInfo	102
CglUniqueRowCuts	106
CliqueEntry	108
CglProbing::CliqueType	108
CoinAbsFitEq [external]	
CoinArrayWithLength [external]	
CoinArbitraryArrayWithLength [external]	
CoinBigIndexArrayWithLength [external]	
CoinDoubleArrayWithLength [external]	
CoinFactorizationDoubleArrayWithLength [external]	
CoinFactorizationLongDoubleArrayWithLength [external]	
CoinIntArrayWithLength [external]	
CoinUnsignedIntArrayWithLength [external]	
CoinVoidStarArrayWithLength [external]	
CoinBaseModel [external]	
CoinModel [external]	
CoinStructuredModel [external]	
CoinBuild [external]	
CoinDenseVector< T > [external]	
CoinError [external]	
CglLandP::NoBasisError	114
CglLandP::SimplexInterfaceError	119
CoinExternalVectorFirstGreater_2< class, class, class > [external]	
CoinExternalVectorFirstGreater_3< class, class, class, class > [external]	
CoinExternalVectorFirstLess_2< class, class, class > [external]	
CoinExternalVectorFirstLess_3< class, class, class, class > [external]	
CoinFactorization [external]	
CoinFileIOBase [external]	
CoinFileInput [external]	
CoinFileOutput [external]	

CoinFirstAbsGreater_2< class, class >[external]	
CoinFirstAbsGreater_3< class, class, class >[external]	
CoinFirstAbsLess_2< class, class >[external]	
CoinFirstAbsLess_3< class, class, class >[external]	
CoinFirstGreater_2< class, class >[external]	
CoinFirstGreater_3< class, class, class >[external]	
CoinFirstLess_2< class, class >[external]	
CoinFirstLess_3< class, class, class >[external]	
CoinLpIO::CoinHashLink[external]	
CoinMpsIO::CoinHashLink[external]	
CoinIndexedVector[external]	
CoinPartitionedVector[external]	
LAP::TabRow . . . . .	119
CoinLpIO[external]	
CoinMessageHandler[external]	
CoinMessages[external]	
CglMessage . . . . .	61
CoinMessage[external]	
LAP::LandPMessages . . . . .	113
LAP::LapMessages . . . . .	113
CoinModelHash[external]	
CoinModelHash2[external]	
CoinModelHashLink[external]	
CoinModelInfo2[external]	
CoinModelLink[external]	
CoinModelLinkedList[external]	
CoinModelTriple[external]	
CoinMpsCardReader[external]	
CoinMpsIO[external]	
CoinOneMessage[external]	
CoinOtherFactorization[external]	
CoinDenseFactorization[external]	
CoinOsIFactorization[external]	
CoinSimpFactorization[external]	
CoinPackedMatrix[external]	
CoinPackedVectorBase[external]	
CoinPackedVector[external]	
CoinShallowPackedVector[external]	
CoinPair< S, T >[external]	
CoinParam[external]	
CoinPrePostsolveMatrix[external]	
CoinPostsolveMatrix[external]	
CoinPresolveMatrix[external]	
CoinPresolveAction[external]	
do_tighten_action[external]	
doubleton_action[external]	
drop_empty_cols_action[external]	
drop_empty_rows_action[external]	
drop_zero_coefficients_action[external]	
dupcol_action[external]	
duprow3_action[external]	
duprow_action[external]	
forcing_constraint_action[external]	
gubrow_action[external]	

```

    implied_free_action[external]
    isolated_constraint_action[external]
    make_fixed_action[external]
    remove_dual_action[external]
    remove_fixed_action[external]
    slack_doubleton_action[external]
    slack_singleton_action[external]
    subst_constraint_action[external]
    tripleton_action[external]
    twotwo_action[external]
    useless_constraint_action[external]
CoinPresolveMonitor[external]
CoinRational[external]
CoinRelFltEq[external]
CoinSearchTreeBase[external]
    CoinSearchTree< class >[external]
CoinSearchTreeCompareBest[external]
CoinSearchTreeCompareBreadth[external]
CoinSearchTreeCompareDepth[external]
CoinSearchTreeComparePreferred[external]
CoinSearchTreeManager[external]
CoinSet[external]
    CoinSosSet[external]
CoinSnapshot[external]
CoinThreadRandom[external]
CoinTimer[external]
CoinTreeNode[external]
CoinTreeSiblings[external]
CoinTriple< S, T, U >[external]
CoinWarmStart[external]
    CoinWarmStartBasis[external]
    CoinWarmStartDual[external]
    CoinWarmStartPrimalDual[external]
    CoinWarmStartVector< T >[external]
    CoinWarmStartVector< double >[external]
    CoinWarmStartVector< U >[external]
    CoinWarmStartVectorPair< T, U >[external]
CoinWarmStartDiff[external]
    CoinWarmStartBasisDiff[external]
    CoinWarmStartDualDiff[external]
    CoinWarmStartPrimalDualDiff[external]
    CoinWarmStartVectorDiff< T >[external]
    CoinWarmStartVectorDiff< double >[external]
    CoinWarmStartVectorDiff< U >[external]
    CoinWarmStartVectorPairDiff< T, U >[external]
CoinYacc[external]
std::complex
std::basic_string< Char >::const_iterator
std::string::const_iterator
std::wstring::const_iterator
std::deque< T >::const_iterator
std::list< T >::const_iterator
std::forward_list< T >::const_iterator
std::map< K, T >::const_iterator

```

std::unordered_map< K, T >::const_iterator	
std::multimap< K, T >::const_iterator	
std::unordered_multimap< K, T >::const_iterator	
std::set< K >::const_iterator	
std::unordered_set< K >::const_iterator	
std::multiset< K >::const_iterator	
std::unordered_multiset< K >::const_iterator	
std::vector< T >::const_iterator	
std::string::const_reverse_iterator	
std::basic_string< Char >::const_reverse_iterator	
std::wstring::const_reverse_iterator	
std::deque< T >::const_reverse_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::unordered_multimap< K, T >::const_reverse_iterator	
std::set< K >::const_reverse_iterator	
std::unordered_set< K >::const_reverse_iterator	
std::multiset< K >::const_reverse_iterator	
std::unordered_multiset< K >::const_reverse_iterator	
std::vector< T >::const_reverse_iterator	
cut . . . . .	108
cut_list . . . . .	109
cutParams . . . . .	109
LAP::Cuts . . . . .	109
cycle . . . . .	110
cycle_list . . . . .	111
std::deque< T >	
DGG_constraint_t . . . . .	111
DGG_data_t . . . . .	111
DGG_list_t . . . . .	111
disaggregationAction . . . . .	112
dropped_zero[external]	
edge . . . . .	112
EKKHlink[external]	
std::error_category	
std::error_code	
std::error_condition	
std::exception	
std::bad_alloc	
std::bad_cast	
std::bad_exception	
std::bad_typeid	
std::ios_base::failure	
std::logic_error	
std::domain_error	
std::invalid_argument	
std::length_error	
std::out_of_range	
std::runtime_error	
std::overflow_error	
std::range_error	

```

    std::underflow_error
FactorPointers[external]
std::forward_list< T >
ilp ..... 112
info_weak ..... 112
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_istreamstream< char >
            basic_istreamstream< wchar_t >
            std::basic_ifstream
                std::ifstream
                std::wifstream
            std::basic_iostream
                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_istreamstream
                std::istreamstream
                std::wistreamstream
            std::istream
            std::wistream
        std::basic_ostream
            basic_iostream< char >
            basic_iostream< wchar_t >
            basic_ofstream< char >
            basic_ofstream< wchar_t >
            basic_ostreamstream< char >
            basic_ostreamstream< wchar_t >
            std::basic_iostream
            std::basic_ofstream
                std::ofstream
                std::wofstream
            std::basic_ostreamstream
                std::ostreamstream
                std::wostreamstream
            std::ostream

```

std::wostream	
std::ios	
std::wios	
std::multiset< K >::iterator	
std::unordered_multimap< K, T >::iterator	
std::unordered_map< K, T >::iterator	
std::forward_list< T >::iterator	
std::map< K, T >::iterator	
std::basic_string< Char >::iterator	
std::wstring::iterator	
std::deque< T >::iterator	
std::list< T >::iterator	
std::string::iterator	
std::multimap< K, T >::iterator	
std::unordered_set< K >::iterator	
std::set< K >::iterator	
std::unordered_multiset< K >::iterator	
std::vector< T >::iterator	
std::list< T >	
log_var . . . . .	114
std::map< K, T >	
std::multimap< K, T >	
std::multiset< K >	
parity_ilp . . . . .	117
pool_cut . . . . .	118
pool_cut_list . . . . .	118
presolvehlink[external]	
std::priority_queue< T >	
std::queue< T >	
Coin::ReferencedObject[external]	
std::deque< T >::reverse_iterator	
std::unordered_set< K >::reverse_iterator	
std::unordered_map< K, T >::reverse_iterator	
std::multimap< K, T >::reverse_iterator	
std::string::reverse_iterator	
std::forward_list< T >::reverse_iterator	
std::map< K, T >::reverse_iterator	
std::unordered_multiset< K >::reverse_iterator	
std::multiset< K >::reverse_iterator	
std::wstring::reverse_iterator	
std::list< T >::reverse_iterator	
std::unordered_multimap< K, T >::reverse_iterator	
std::basic_string< Char >::reverse_iterator	
std::vector< T >::reverse_iterator	
std::set< K >::reverse_iterator	
select_cut . . . . .	118
separation_graph . . . . .	118
std::set< K >	
short_path_node . . . . .	118
std::smart_ptr< T >	
Coin::SmartPtr< T >[external]	
std::stack< T >	
symrec[external]	
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 >  
LAP::Validator . . . . . 120  
std::vector< T >  
std::vector< bool >  
std::vector< ColumnSelectionStrategy >  
std::vector< double >  
std::vector< int >  
std::vector< OsiRowCut \* >  
std::vector< RowSelectionStrategy >  
std::weak\_ptr< T >  
K  
S  
T  
U



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">auxiliary_graph</a>	19
<a href="#">Cgl012Cut</a>	
012Cut Generator Class	19
<a href="#">cgl_arc</a>	20
<a href="#">cgl_graph</a>	20
<a href="#">cgl_node</a>	20
<a href="#">CglAllDifferent</a>	
AllDifferent Cut Generator Class This has a number of sets	20
<a href="#">CglBK</a>	
For Bron-Kerbosch	22
<a href="#">CglClique</a>	23
<a href="#">CglCutGenerator</a>	
Cut Generator Base Class	27
<a href="#">CglDuplicateRow</a>	
DuplicateRow Cut Generator Class	29
<a href="#">CglFakeClique</a>	32
<a href="#">CglFlowCover</a>	
Lifed Simple Generalized Flow Cover Cut Generator Class	34
<a href="#">CglFlowVUB</a>	
Variable upper bound class	35
<a href="#">CglGMI</a>	
Gomory cut generator with several cleaning procedures, used to test the numerical safety of the resulting cuts	36
<a href="#">CglGMIParam</a>	
Class collecting parameters for the GMI cut generator	39
<a href="#">CglGomory</a>	
Gomory Cut Generator Class	46
<a href="#">CglHashLink</a>	
Only store unique row cuts	48
<a href="#">CglImplication</a>	
This just uses implication info	49
<a href="#">CglKnapsackCover</a>	
Knapsack Cover Cut Generator Class	50
<a href="#">CglLandP</a>	51

<a href="#">LAP::CglLandPSimplex</a>	54
<a href="#">CglLiftAndProject</a>	
Lift And Project Cut Generator Class	59
<a href="#">CglMessage</a>	
This deals with Cgl messages (as against Osi messages etc)	61
<a href="#">CglMixedIntegerRounding</a>	
Mixed Integer Rounding Cut Generator Class	61
<a href="#">CglMixedIntegerRounding2</a>	
Mixed Integer Rounding Cut Generator Class	63
<a href="#">CglMixIntRoundVUB</a>	64
<a href="#">CglMixIntRoundVUB2</a>	65
<a href="#">CglOddHole</a>	
Odd Hole Cut Generator Class	65
<a href="#">CglParam</a>	
Class collecting parameters for all cut generators	67
<a href="#">CglPreProcess</a>	
Class for preProcessing and postProcessing	68
<a href="#">CglProbing</a>	
Probing Cut Generator Class	72
<a href="#">CglRedSplit</a>	
Gomory Reduce-and-Split Cut Generator Class; See method <a href="#">generateCuts()</a>	77
<a href="#">CglRedSplit2</a>	
Reduce-and-Split Cut Generator Class; See method <a href="#">generateCuts()</a>	80
<a href="#">CglRedSplit2Param</a>	
Class collecting parameters the Reduced-and-split cut generator	82
<a href="#">CglRedSplitParam</a>	
Class collecting parameters the Reduced-and-split cut generator	90
<a href="#">CglResidualCapacity</a>	
Residual Capacity Inequalities Cut Generator Class	95
<a href="#">CglSimpleRounding</a>	
Simple Rounding Cut Generator Class	97
<a href="#">CglStored</a>	
Stored Cut Generator Class	98
<a href="#">CglTreeInfo</a>	
Information about where the cut generator is invoked from	100
<a href="#">CglTreeProbingInfo</a>	102
<a href="#">CglTwomir</a>	
Twostep MIR Cut Generator Class	104
<a href="#">CglUniqueRowCuts</a>	106
<a href="#">CglZeroHalf</a>	
Zero Half Cut Generator Class	106
<a href="#">CliqueEntry</a>	
Derived class to pick up probing info	108
<a href="#">CglProbing::CliqueType</a>	
Clique type	108
<a href="#">cut</a>	108
<a href="#">cut_list</a>	109
<a href="#">cutParams</a>	109
<a href="#">LAP::Cuts</a>	
To store extra cuts generated by columns from which they origin	109
<a href="#">cycle</a>	110
<a href="#">cycle_list</a>	111
<a href="#">DGG_constraint_t</a>	111
<a href="#">DGG_data_t</a>	111

DGG_list_t	111
disaggregationAction	
Only useful type of disaggregation is most normal For now just done for 0-1 variables Can be used	
for building cliques	112
edge	112
ilp	112
info_weak	112
LAP::LandPMessages	
Message handler for lift-and-project simplex	113
LAP::LapMessages	
Output messages for Cgl	113
log_var	114
CglLandP::NoBasisError	114
CglLandP::Parameters	
Class storing parameters	114
parity_ilp	117
pool_cut	118
pool_cut_list	118
select_cut	118
separation_graph	118
short_path_node	118
CglLandP::SimplexInterfaceError	119
LAP::TabRow	119
LAP::Validator	
Class to validate or reject a cut	120



## Chapter 4

# File Index

### 4.1 File List

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

Cgl012cut.hpp	??
CglAllDifferent.hpp	??
CglClique.hpp	??
CglConfig.h	??
CglCutGenerator.hpp	??
CglDuplicateRow.hpp	??
CglFlowCover.hpp	??
CglGMI.hpp	??
CglGMIParam.hpp	??
CglGomory.hpp	??
CglKnapsackCover.hpp	??
CglLandP.hpp	??
CglLandPMessages.hpp	??
CglLandPSimplex.hpp	??
CglLandPTabRow.hpp	??
CglLandPUtils.hpp	??
CglLandPValidator.hpp	??
CglLiftAndProject.hpp	??
CglMessage.hpp	??
CglMixedIntegerRounding.hpp	??
CglMixedIntegerRounding2.hpp	??
CglOddHole.hpp	??
CglParam.hpp	??
CglPreProcess.hpp	??
CglProbing.hpp	??
CglRedSplit.hpp	??
CglRedSplit2.hpp	??
CglRedSplit2Param.hpp	??
CglRedSplitParam.hpp	??
CglResidualCapacity.hpp	??
CglSimpleRounding.hpp	??
CglStored.hpp	??
CglTreeInfo.hpp	??
CglTwomir.hpp	??

CglZeroHalf.hpp	..	??
config_cgl_default.h	..	??
config_default.h	..	??

## Chapter 5

# Namespace Documentation

### 5.1 LAP Namespace Reference

Performs one round of Lift & Project using [CglLandPSimplex](#) to build cuts.

#### Classes

- class [CglLandPSimplex](#)
- struct [Cuts](#)  
*To store extra cuts generated by columns from which they origin.*
- class [LandPMessages](#)  
*Message handler for lift-and-project simplex.*
- class [LapMessages](#)  
*Output messages for Cgl.*
- struct [TabRow](#)
- class [Validator](#)  
*Class to validate or reject a cut.*

#### Enumerations

- enum [LAP\\_messages](#)  
*Types of messages for lift-and-project simplex.*

#### Functions

- double [normCoef](#) ([TabRow](#) &row, int ncols, const int \*nonBasics)  
*Compute  $\sum_{j=1}^n a_{ij} \mid \{1 - a_{i0}\}$  for row passed as argument.*
- void [scale](#) (OsiRowCut &cut)  
*scale the cut passed as argument*
- void [scale](#) (OsiRowCut &cut, double norma)  
*scale the cut passed as argument using provided normalization factor*
- void [modularizeRow](#) ([TabRow](#) &row, const bool \*integerVar)  
*Modularize row.*

- double [intersectionCutCoef](#) (double alpha\_i, double beta)  
*return the coefficients of the intersection cut*
- double [modularizedCoef](#) (double alpha, double beta)  
*compute the modularized row coefficient for an integer variable*
- bool [int\\_val](#) (double value, double tol)  
*Says is value is integer.*

### 5.1.1 Detailed Description

Performs one round of Lift & Project using [CglLandPSimplex](#) to build cuts.  
constants describing rejection codes

### 5.1.2 Enumeration Type Documentation

#### 5.1.2.1 enum LAP::LAP\_messages

Types of messages for lift-and-project simplex.  
Definition at line 22 of file CglLandPMessages.hpp.

### 5.1.3 Function Documentation

#### 5.1.3.1 double LAP::normCoef ( TabRow & row, int ncols, const int \* nonBasics )

Compute  $\sum_{j=1}^n |a_{ij}| \{1 - a_{i0}\}$  for row passed as argument.

#### 5.1.3.2 void LAP::modularizeRow ( TabRow & row, const bool \* integerVar )

Modularize row.



## Chapter 6

# Class Documentation

### 6.1 auxiliary\_graph Struct Reference

Collaboration diagram for auxiliary\_graph:

### 6.2 Cgl012Cut Class Reference

012Cut Generator Class

```
#include <Cgl012cut.hpp>
```

#### Public Member Functions

##### Constructors and destructors

- [Cgl012Cut \(\)](#)  
*Default constructor.*
- [Cgl012Cut \(const Cgl012Cut &\)](#)  
*Copy constructor.*
- [Cgl012Cut & operator= \(const Cgl012Cut &rhs\)](#)  
*Assignment operator.*
- virtual [~Cgl012Cut \(\)](#)  
*Destructor.*

#### Generate Cuts

- int **sep\_012\_cut** (int mr, int mc, int mnz, int \*mtbeg, int \*mtcnt, int \*mtind, int \*mtval, int \*vlb, int \*vub, int \*mrhs, char \*msense, const double \*xstar, bool aggressive, int \*cnum, int \*cnzcnt, int \*\*cbeg, int \*\*ccnt, int \*\*cind, int \*\*cval, int \*\*crhs, char \*\*csense)
- void **ilp\_load** (int mr, int mc, int mnz, int \*mtbeg, int \*mtcnt, int \*mtind, int \*mtval, int \*vlb, int \*vub, int \*mrhs, char \*msense)
- void **free\_ilp** ()
- void **alloc\_parity\_ilp** (int mr, int mc, int mnz)
- void **free\_parity\_ilp** ()
- void **initialize\_log\_var** ()
- void **free\_log\_var** ()

### 6.2.1 Detailed Description

012Cut Generator Class

This class is to make Cgl01cut thread safe etc

Definition at line 207 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.3 cgl\_arc Struct Reference

### 6.3.1 Detailed Description

Definition at line 35 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.4 cgl\_graph Struct Reference

Collaboration diagram for cgl\_graph:

### 6.4.1 Detailed Description

Definition at line 49 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.5 cgl\_node Struct Reference

Collaboration diagram for cgl\_node:

### 6.5.1 Detailed Description

Definition at line 41 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.6 CglAllDifferent Class Reference

AllDifferent Cut Generator Class This has a number of sets.

```
#include <CglAllDifferent.hpp>
```

Inheritance diagram for CglAllDifferent:

Collaboration diagram for CglAllDifferent:

## Public Member Functions

### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*This fixes (or reduces bounds) on sets of all different variables.*

### Constructors and destructors

- [CglAllDifferent](#) ()  
*Default constructor.*
- [CglAllDifferent](#) (int numberSets, const int \*starts, const int \*which)  
*Useful constructot.*
- [CglAllDifferent](#) (const [CglAllDifferent](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglAllDifferent](#) & [operator=](#) (const [CglAllDifferent](#) &rhs)  
*Assignment operator.*
- virtual [~CglAllDifferent](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [refreshSolver](#) (OsiSolverInterface \*solver)  
*This can be used to refresh any inforamtion.*
- virtual bool [mayGenerateRowCutsInTree](#) () const  
*Returns true if may generate Row cuts in tree (rather than root node).*

### Sets and Gets

- void [setLogLevel](#) (int value)  
*Set log level.*
- int [getLogLevel](#) () const  
*Get log level.*
- void [setMaxLook](#) (int value)  
*Set Maximum number of sets to look at at once.*
- int [getMaxLook](#) () const  
*Get Maximum number of sets to look at at once.*

## Additional Inherited Members

### 6.6.1 Detailed Description

AllDifferent Cut Generator Class This has a number of sets.

All the members in each set are general integer variables which have to be different from all others in the set.

At present this only generates column cuts

At present it is very primitive compared to proper CSP implementations

Definition at line 20 of file CglAllDifferent.hpp.

## 6.6.2 Member Function Documentation

6.6.2.1 `virtual bool CglAllDifferent::mayGenerateRowCutsInTree ( ) const [inline],[virtual]`

Returns true if may generate Row cuts in tree (rather than root node).

Used so know if matrix will change in tree. Really meant so column cut generators can still be active without worrying code. Default is true

Reimplemented from [CglCutGenerator](#).

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

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

- `CglAllDifferent.hpp`

## 6.7 CglBK Class Reference

For Bron-Kerbosch.

```
#include <CglPreProcess.hpp>
```

### Public Member Functions

#### Main methods

- void [bronKerbosch](#) ()  
*For recursive Bron-Kerbosch.*
- `OsiSolverInterface * newSolver (const OsiSolverInterface &model)`  
*Creates strengthened smaller model.*

#### Constructors and destructors etc

- [CglBK](#) ()  
*Default constructor.*
- [CglBK](#) (const OsiSolverInterface &model, const char \*rowType, int numberElements)  
*Useful constructor.*
- [CglBK](#) (const [CglBK](#) &rhs)  
*Copy constructor.*
- [CglBK](#) & [operator=](#) (const [CglBK](#) &rhs)  
*Assignment operator.*
- [~CglBK](#) ()  
*Destructor.*

### 6.7.1 Detailed Description

For Bron-Kerbosch.

Definition at line 366 of file `CglPreProcess.hpp`.

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

- `CglPreProcess.hpp`

## 6.8 CglClique Class Reference

Inheritance diagram for CglClique:

Collaboration diagram for CglClique:

### Public Member Functions

- [CglClique](#) (const [CglClique](#) &rhs)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglClique](#) & [operator=](#) (const [CglClique](#) &rhs)  
*Assignment operator.*
- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate cuts for the model data contained in si.*

### Protected Attributes

- const int \* [cl\\_perm\\_indices](#)  
*variables/arrays that are used across many methods*
- int [cl\\_perm\\_length](#)  
*The length of cl\_perm\_indices.*
- int \* [cl\\_indices](#)  
*List of indices that should be considered for extending the ones listed in cl\_perm\_indices.*
- int [cl\\_length](#)  
*The length of cl\_indices.*
- int \* [cl\\_del\\_indices](#)  
*An array of nodes discarded from the candidate list.*
- int [cl\\_del\\_length](#)  
*The length of cl\_del\_indices.*

### Friends

- void [CglCliqueUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
*A function that tests the methods in the [CglClique](#) class.*

### Constructors and destructors

- enum [scl\\_next\\_node\\_method](#)  
*possible choices for selecting the next node in the star clique search*
- struct [frac\\_graph](#)
- bool [setPacking\\_](#)  
*An indicator showing whether the whole matrix in the solverinterface is a set packing problem or not.*
- bool [justOriginalRows\\_](#)  
*True if just look at original rows.*
- int [sp\\_numrows](#)

*pieces of the set packing part of the solverinterface*

- int \* **sp\_orig\_row\_ind**
- int **sp\_numcols**
- int \* **sp\_orig\_col\_ind**
- double \* **sp\_colsol**
- int \* **sp\_col\_start**
- int \* **sp\_col\_ind**
- int \* **sp\_row\_start**
- int \* **sp\_row\_ind**
- frac\_graph **fgraph**

*the intersection graph corresponding to the set packing problem*

- bool \* **node\_node**

*the node-node incidence matrix of the intersection graph.*

- double **petol**

*The primal tolerance in the solverinterface.*

- bool **do\_row\_clique**

*data for the star clique algorithm*

- bool **do\_star\_clique**

*whether to do the star clique algorithm or not.*

- scl\_next\_node\_method scl\_next\_node\_rule

*How the next node to be added to the star clique should be selected.*

- int **scl\_candidate\_length\_threshold**

*In the star clique method the maximal length of the candidate list (those nodes that are in a star, i.e., connected to the center of the star) to allow complete enumeration of maximal cliques.*

- bool **scl\_report\_result**

*whether to give a detailed statistics on the star clique method*

- int **rcl\_candidate\_length\_threshold**

*In the row clique method the maximal length of the candidate list (those nodes that can extend the row clique, i.e., connected to all nodes in the row clique) to allow complete enumeration of maximal cliques.*

- bool **rcl\_report\_result**

*whether to give a detailed statistics on the row clique method*

- CglClique (bool setPacking=false, bool justOriginalRows=false)

*Default constructor.*

- virtual ~CglClique ()

*Destructor.*

- virtual std::string **generateCpp** (FILE \*fp)

*Create C++ lines to get to current state.*

- void **considerRows** (const int numRows, const int \*rowInd)
- void **setStarCliqueNextNodeMethod** (scl\_next\_node\_method method)
- void **setStarCliqueCandidateLengthThreshold** (int maxlen)
- void **setRowCliqueCandidateLengthThreshold** (int maxlen)
- void **setStarCliqueReport** (bool yesno=true)
- void **setRowCliqueReport** (bool yesno=true)
- void **setDoStarClique** (bool yesno=true)
- void **setDoRowClique** (bool yesno=true)
- void **setMinViolation** (double minviol)
- double **getMinViolation** () const

## Additional Inherited Members

### 6.8.1 Detailed Description

Definition at line 14 of file CglClique.hpp.

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 CglClique::CglClique ( bool *setPacking* = false, bool *justOriginalRows* = false )

Default constructor.

If the *setPacking* argument is set to true then [CglClique](#) will assume that the problem in the solverinterface passed to the [generateCuts\(\)](#) method describes a set packing problem, i.e.,

- all variables are binary
- the matrix is a 0-1 matrix
- all constraints are '= 1' or '<= 1'

Otherwise the user can use the [considerRows\(\)](#) method to set the list of clique rows, that is,

- all coeffs corresponding to binary variables at fractional level is 1
- all other coeffs are non-negative
- the constraint is '= 1' or '<= 1'.

If the user does not set the list of clique rows then [CglClique](#) will start the [generateCuts\(\)](#) methods by scanning the matrix for them. Also *justOriginalRows* can be set to true to limit clique creation

### 6.8.3 Member Function Documentation

#### 6.8.3.1 virtual void CglClique::generateCuts ( const OsiSolverInterface & *si*, OsiCuts & *cs*, const CglTreeInfo *info* = CglTreeInfo() ) [virtual]

Generate cuts for the model data contained in *si*.

The generated cuts are inserted into and returned in the collection of cuts *cs*.

Implements [CglCutGenerator](#).

Reimplemented in [CglFakeClique](#).

### 6.8.4 Friends And Related Function Documentation

#### 6.8.4.1 void CglCliqueUnitTest ( const OsiSolverInterface \* *siP*, const std::string *mpdDir* ) [friend]

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

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

### 6.8.5 Member Data Documentation

#### 6.8.5.1 `bool* CgIClique::node_node` [protected]

the node-node incidence matrix of the intersection graph.

Definition at line 156 of file CgIClique.hpp.

#### 6.8.5.2 `double CgIClique::petol` [protected]

The primal tolerance in the solverinterface.

Definition at line 159 of file CgIClique.hpp.

#### 6.8.5.3 `bool CgIClique::do_row_clique` [protected]

data for the star clique algorithm

Parameters whether to do the row clique algorithm or not.

Definition at line 166 of file CgIClique.hpp.

#### 6.8.5.4 `bool CgIClique::do_star_clique` [protected]

whether to do the star clique algorithm or not.

Definition at line 168 of file CgIClique.hpp.

#### 6.8.5.5 `int CgIClique::scl_candidate_length_threshold` [protected]

In the star clique method the maximal length of the candidate list (those nodes that are in a star, i.e., connected to the center of the star) to allow complete enumeration of maximal cliques.

Otherwise a greedy algorithm is used.

Definition at line 176 of file CgIClique.hpp.

#### 6.8.5.6 `int CgIClique::rcl_candidate_length_threshold` [protected]

In the row clique method the maximal length of the candidate list (those nodes that can extend the row clique, i.e., connected to all nodes in the row clique) to allow complete enumeration of maximal cliques.

Otherwise a greedy algorithm is used.

Definition at line 184 of file CgIClique.hpp.

#### 6.8.5.7 `const int* CgIClique::cl_perm_indices` [protected]

variables/arrays that are used across many methods

List of indices that must be in the to be created clique. This is just a pointer, it is never new'd and therefore does not need to be delete[]'d either.

Definition at line 194 of file CgIClique.hpp.



6.8.5.8 `int* CglClique::cl_indices` [protected]

List of indices that should be considered for extending the ones listed in `cl_perm_indices`.

Definition at line 200 of file `CglClique.hpp`.

6.8.5.9 `int* CglClique::cl_del_indices` [protected]

An array of nodes discarded from the candidate list.

These are rechecked when a maximal clique is found just to make sure that the clique is really maximal.

Definition at line 207 of file `CglClique.hpp`.

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

- `CglClique.hpp`

## 6.9 CglCutGenerator Class Reference

Cut Generator Base Class.

```
#include <CglCutGenerator.hpp>
```

Inheritance diagram for `CglCutGenerator`:

### Public Member Functions

#### Generate Cuts

- virtual void `generateCuts` (const `OsiSolverInterface` &si, `OsiCuts` &cs, const `CglTreeInfo` info=`CglTreeInfo()`)=0  
*Generate cuts for the model data contained in si.*

#### Constructors and destructors

- `CglCutGenerator` ()  
*Default constructor.*
- `CglCutGenerator` (const `CglCutGenerator` &)  
*Copy constructor.*
- virtual `CglCutGenerator * clone` () const =0  
*Clone.*
- `CglCutGenerator & operator=` (const `CglCutGenerator` &rhs)  
*Assignment operator.*
- virtual `~CglCutGenerator` ()  
*Destructor.*
- virtual std::string `generateCpp` (FILE \*)  
*Create C++ lines to set the generator in the current state.*
- virtual void `refreshSolver` (`OsiSolverInterface` \*)  
*This can be used to refresh any information.*

#### Gets and Sets

- int `getAggressiveness` () const

- *Get Aggressiveness - 0 = neutral, 100 is normal root node.*
- void [setAggressiveness](#) (int value)
  - *Set Aggressiveness - 0 = neutral, 100 is normal root node.*
- void [setGlobalCuts](#) (bool trueOrFalse)
  - *Set whether can do global cuts.*
- bool [canDoGlobalCuts](#) () const
  - *Say whether can do global cuts.*
- virtual bool [mayGenerateRowCutsInTree](#) () const
  - *Returns true if may generate Row cuts in tree (rather than root node).*
- virtual bool [needsOptimalBasis](#) () const
  - *Return true if needs optimal basis to do cuts.*
- virtual int [maxLengthOfCutInTree](#) () const
  - *Return maximum length of cut in tree.*

## Public Attributes

- int [aggressive\\_](#)
  - *Aggressiveness - 0 = neutral, 100 is normal root node.*
- bool [canDoGlobalCuts\\_](#)
  - *True if can do global cuts i.e. no general integers.*

## 6.9.1 Detailed Description

Cut Generator Base Class.

This is an abstract base class for generating cuts. A specific cut generator will inherit from this class.

Definition at line 23 of file CglCutGenerator.hpp.

## 6.9.2 Member Function Documentation

**6.9.2.1** virtual void CglCutGenerator::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [pure virtual]

Generate cuts for the model data contained in si.

The generated cuts are inserted into and returned in the collection of cuts cs.

Implemented in [CglImplication](#), [CglFakeClique](#), [CglLandP](#), [CglFlowCover](#), [CglMixedIntegerRounding2](#), [CglMixedIntegerRounding](#), [CglTwomir](#), [CglProbing](#), [CglResidualCapacity](#), [CglRedSplit2](#), [CglGMI](#), [CglRedSplit](#), [CglOddHole](#), [CglDuplicateRow](#), [CglSimpleRounding](#), [CglZeroHalf](#), [CglGomory](#), [CglStored](#), [CglClique](#), [CglAllDifferent](#), [CglKnapsackCover](#), and [CglLiftAndProject](#).

**6.9.2.2** virtual std::string CglCutGenerator::generateCpp ( FILE \* ) [inline],[virtual]

Create C++ lines to set the generator in the current state.

The output must be parsed by the calling code, as each line starts with a key indicating the following:

- 0: must be kept (for #includes etc)
- 3: Set to changed (not default) values
- 4: Set to default values (redundant)

Keys 1, 2, 5, 6, 7, 8 are defined, but not applicable to cut generators.

Reimplemented in [CglImplication](#), [CglProbing](#), [CglFlowCover](#), [CglRedSplit](#), [CglTwomir](#), [CglMixedIntegerRounding2](#), [CglMixedIntegerRounding](#), [CglGMI](#), [CglGomory](#), [CglDuplicateRow](#), [CglZeroHalf](#), [CglLiftAndProject](#), [CglSimpleRounding](#), [CglAllDifferent](#), [CglClique](#), and [CglKnapsackCover](#).

Definition at line 65 of file `CglCutGenerator.hpp`.

**6.9.2.3** `int CglCutGenerator::getAggressiveness ( ) const` `[inline]`

Get Aggressiveness - 0 = neutral, 100 is normal root node.

Really just a hint to cut generator

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

**6.9.2.4** `void CglCutGenerator::setAggressiveness ( int value )` `[inline]`

Set Aggressiveness - 0 = neutral, 100 is normal root node.

Really just a hint to cut generator

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

**6.9.2.5** `virtual bool CglCutGenerator::mayGenerateRowCutsInTree ( ) const` `[virtual]`

Returns true if may generate Row cuts in tree (rather than root node).

Used so know if matrix will change in tree. Really meant so column cut generators can still be active without worrying code. Default is true

Reimplemented in [CglProbing](#), and [CglAllDifferent](#).

## 6.9.3 Member Data Documentation

**6.9.3.1** `int CglCutGenerator::aggressive_`

Aggressiveness - 0 = neutral, 100 is normal root node.

Really just a hint to cut generator

Definition at line 116 of file `CglCutGenerator.hpp`.

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

- `CglCutGenerator.hpp`

## 6.10 CglDuplicateRow Class Reference

DuplicateRow Cut Generator Class.

```
#include <CglDuplicateRow.hpp>
```

Inheritance diagram for `CglDuplicateRow`:

Collaboration diagram for `CglDuplicateRow`:

## Public Member Functions

### Get information on size of problem

- const int \* [duplicate](#) () const  
*Get duplicate row list, -1 means still in, -2 means out (all fixed),  $k \geq$  means same as row  $k$ .*
- int [sizeDynamic](#) () const  
*Size of dynamic program.*
- int [numberOriginalRows](#) () const  
*Number of rows in original problem.*
- int [logLevel](#) () const  
*logLevel*
- void [setLogLevel](#) (int value)

### We only check for duplicates amongst rows with effective rhs $\leq$ this

- int [maximumRhs](#) () const  
*Get.*
- void [setMaximumRhs](#) (int value)  
*Set.*

### We only check for dominated amongst groups of columns whose size $\leq$ this

- int [maximumDominated](#) () const  
*Get.*
- void [setMaximumDominated](#) (int value)  
*Set.*

### gets and sets

- int [mode](#) () const  
*Get mode.*
- void [setMode](#) (int value)  
*Set mode.*

### Constructors and destructors

- [CglDuplicateRow](#) ()  
*Default constructor.*
- [CglDuplicateRow](#) (OsiSolverInterface \*solver)  
*Useful constructor.*
- [CglDuplicateRow](#) (const [CglDuplicateRow](#) &rhs)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglDuplicateRow](#) & [operator=](#) (const [CglDuplicateRow](#) &rhs)  
*Assignment operator.*
- virtual [~CglDuplicateRow](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [refreshSolver](#) (OsiSolverInterface \*solver)  
*This can be used to refresh any information.*

## Protected Attributes

### Protected member data

- **CoinPackedMatrix** [matrix\\_](#)  
*Matrix.*
- **CoinPackedMatrix** [matrixByRow\\_](#)  
*Matrix by row.*
- int \* [rhs\\_](#)  
*Possible rhs (if 0 then not possible)*
- int \* [duplicate\\_](#)  
*Marks duplicate rows.*
- int \* [lower\\_](#)  
*To allow for  $\leq$  rows.*
- **CglStored** \* [storedCuts\\_](#)  
*Stored cuts if we found dominance cuts.*
- int [maximumDominated\\_](#)  
*Check dominated columns if less than this number of candidates.*
- int [maximumRhs\\_](#)  
*Check duplicates if effective rhs  $\leq$  this.*
- int [sizeDynamic\\_](#)  
*Size of dynamic program.*
- int [mode\\_](#)  
*1 do rows, 2 do columns, 3 do both*
- int [logLevel\\_](#)  
*Controls print out.*

## Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Fix variables and find duplicate/dominated rows for the model of the solver interface, si.*
- **CglStored** \* [outDuplicates](#) (OsiSolverInterface \*solver)  
*Fix variables and find duplicate/dominated rows for the model of the solver interface, si.*

## Additional Inherited Members

### 6.10.1 Detailed Description

DuplicateRow Cut Generator Class.

Definition at line 15 of file CglDuplicateRow.hpp.

### 6.10.2 Member Function Documentation

**6.10.2.1** virtual void CglDuplicateRow::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]

Fix variables and find duplicate/dominated rows for the model of the solver interface, si.

This is a very simple minded idea but I (JJF) am using it in a project so thought I might as well add it. It should really be called before first solve and I may modify CBC to allow for that.

This is designed for problems with few rows and many integer variables where the rhs are  $\leq$  or  $=$  and all coefficients and rhs are small integers.

If effective rhs is K then we can fix all variables with coefficients  $> K$  to their lower bounds (effective rhs just means original with variables with nonzero lower bounds subtracted out).

If one row is a subset of another and the effective rhs are same we can fix some variables and then the two rows are identical.

The generator marks identical rows so can be taken out in solve

Implements [CglCutGenerator](#).

#### 6.10.2.2 CglStored\* CglDuplicateRow::outDuplicates ( OsiSolverInterface \* solver )

Fix variables and find duplicate/dominated rows for the model of the solver interface, si.

This is a very simple minded idea but I (JJF) am using it in a project so thought I might as well add it. It should really be called before first solve and I may modify CBC to allow for that.

This is designed for problems with few rows and many integer variables where the rhs are  $\leq$  or  $=$  and all coefficients and rhs are small integers.

If effective rhs is K then we can fix all variables with coefficients  $> K$  to their lower bounds (effective rhs just means original with variables with nonzero lower bounds subtracted out).

If one row is a subset of another and the effective rhs are same we can fix some variables and then the two rows are identical.

This version does deletions and fixings and may return stored cuts for dominated columns

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

- CglDuplicateRow.hpp

## 6.11 CglFakeClique Class Reference

Inheritance diagram for CglFakeClique:

Collaboration diagram for CglFakeClique:

### Public Member Functions

- [CglFakeClique](#) (const [CglFakeClique](#) &rhs)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* clone () const  
*Clone.*
- [CglFakeClique](#) & operator= (const [CglFakeClique](#) &rhs)  
*Assignment operator.*
- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate cuts for the model data contained in si.*

### Constructors and destructors

- OsiSolverInterface \* [fakeSolver\\_](#)

- fake solver to use*
- [CglProbing](#) \* [probing\\_](#)  
*Probing object.*
- [CglFakeClique](#) (OsiSolverInterface \*solver=NULL, bool setPacking=false)  
*Default constructor.*
- virtual [~CglFakeClique](#) ()  
*Destructor.*
- void [assignSolver](#) (OsiSolverInterface \*fakeSolver)  
*Assign solver (generator takes over ownership)*

## Additional Inherited Members

### 6.11.1 Detailed Description

Definition at line 262 of file CglClique.hpp.

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 CglFakeClique::CglFakeClique ( OsiSolverInterface \* solver = NULL, bool setPacking = false )

Default constructor.

If the setPacking argument is set to true then [CglFakeClique](#) will assume that the problem in the solverinterface passed to the [generateCuts\(\)](#) method describes a set packing problem, i.e.,

- all variables are binary
- the matrix is a 0-1 matrix
- all constraints are '= 1' or '<= 1'

Otherwise the user can use the [considerRows\(\)](#) method to set the list of clique rows, that is,

- all coeffs corresponding to binary variables at fractional level is 1
- all other coeffs are non-negative
- the constraint is '= 1' or '<= 1'.

If the user does not set the list of clique rows then [CglFakeClique](#) will start the [generateCuts\(\)](#) methods by scanning the matrix for them.

### 6.11.3 Member Function Documentation

#### 6.11.3.1 virtual void CglFakeClique::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]

Generate cuts for the model data contained in si.

The generated cuts are inserted into and returned in the collection of cuts cs.

Reimplemented from [CglClique](#).

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

- CglClique.hpp

## 6.12 CglFlowCover Class Reference

Lifed Simple Generalized Flow Cover Cut Generator Class.

```
#include <CglFlowCover.hpp>
```

Inheritance diagram for CglFlowCover:

Collaboration diagram for CglFlowCover:

### Public Member Functions

- void [flowPreprocess](#) (const OsiSolverInterface &si)

*Do the following tasks:*

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())

*Generate Lifed Simple Generalized flow cover cuts for the model data contained in si.*

**Functions to query and set maximum number of cuts can be generated.**

- int **getMaxNumCuts** () const
- void **setMaxNumCuts** (int mc)

### Constructors and destructors

- [CglFlowCover](#) ()  
*Default constructor.*
- [CglFlowCover](#) (const [CglFlowCover](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglFlowCover](#) & [operator=](#) (const [CglFlowCover](#) &rhs)  
*Assignment operator.*
- virtual [~CglFlowCover](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

### Static Public Member Functions

**Functions to query and set the number of cuts have been generated.**

- static int **getNumFlowCuts** ()
- static void **setNumFlowCuts** (int fc)
- static void **incNumFlowCuts** (int fc=1)



## Friends

- void [CglFlowCoverUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
A function that tests the methods in the [CglFlowCover](#) class.

## Additional Inherited Members

### 6.12.1 Detailed Description

Lifed Simple Generalized Flow Cover Cut Generator Class.

Definition at line 148 of file CglFlowCover.hpp.

### 6.12.2 Member Function Documentation

#### 6.12.2.1 void CglFlowCover::flowPreprocess ( const OsiSolverInterface & si )

Do the following tasks:

- classify row types
- indentify vubs and vlbs

This function is called by `generateCuts(const OsiSolverInterface & si, OsiCuts & cs).`

#### 6.12.2.2 virtual void CglFlowCover::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]

Generate Lifed Simple Generalized flow cover cuts for the model data contained in si.

The generated cuts are inserted into and returned in the collection of cuts cs.

Implements [CglCutGenerator](#).

### 6.12.3 Friends And Related Function Documentation

#### 6.12.3.1 void CglFlowCoverUnitTest ( const OsiSolverInterface \* siP, const std::string mpdDir ) [friend]

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

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

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

- CglFlowCover.hpp

## 6.13 CglFlowVUB Class Reference

Variable upper bound class.

```
#include <CglFlowCover.hpp>
```

## Public Member Functions

- [CglFlowVUB](#) ()

*The Value of the associated upper bound.*

### Query and set functions for associated 0-1 variable index

*and value.*

- int **getVar** () const
- double **getVal** () const
- void **setVar** (const int v)
- void **setVal** (const double v)

## Protected Attributes

- double [upper\\_](#)

*The index of the associated 0-1 variable.*

### 6.13.1 Detailed Description

Variable upper bound class.

Definition at line 102 of file CglFlowCover.hpp.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 `CglFlowVUB::CglFlowVUB ( )` `[inline]`

The Value of the associated upper bound.

Definition at line 109 of file CglFlowCover.hpp.

### 6.13.3 Member Data Documentation

#### 6.13.3.1 `double CglFlowVUB::upper_` `[protected]`

The index of the associated 0-1 variable.

Definition at line 106 of file CglFlowCover.hpp.

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

- CglFlowCover.hpp

## 6.14 CglGMI Class Reference

Gomory cut generator with several cleaning procedures, used to test the numerical safety of the resulting cuts.

```
#include <CglGMI.hpp>
```

Inheritance diagram for CglGMI:

Collaboration diagram for CglGMI:

## Public Types

- enum [RejectionType](#)

*Public enum: all possible reasons for cut rejection.*

## Public Member Functions

### generateCuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate Gomory Mixed-Integer cuts for the model of the solver interface si.*
- virtual bool [needsOptimalBasis](#) () const  
*Return true if needs optimal basis to do cuts (will return true)*

### Common Methods

- bool **areEqual** (double x, double y, double epsAbs=1e-12, double epsRel=1e-12)
- bool **isZero** (double x, double epsZero=1e-20)
- bool **isIntegerValue** (double x, double intEpsAbs=1e-9, double intEpsRel=1e-15)

### Public Methods

- void **setParam** (const [CglGMIParam](#) &source)
- [CglGMIParam](#) **getParam** () const
- [CglGMIParam](#) & **getParam** ()
- void **computeIsInteger** ()
- void [printOptTab](#) (OsiSolverInterface \*solver) const  
*Print the current simplex tableau.*
- void [setTrackRejection](#) (bool value)  
*Set/get tracking of the rejection of cutting planes.*
- bool **getTrackRejection** ()
- int [getNumberRejectedCuts](#) ([RejectionType](#) reason)  
*Get number of cuts rejected for given reason; see above.*
- void [resetRejectionCounters](#) ()  
*Reset counters for cut rejection tracking; see above.*
- int [getNumberGeneratedCuts](#) ()  
*Get total number of generated cuts since last [resetRejectionCounters](#)()*

### Constructors and destructors

- [CglGMI](#) ()  
*Default constructor.*
- [CglGMI](#) (const [CglGMIParam](#) &param)  
*Constructor with specified parameters.*
- [CglGMI](#) (const [CglGMI](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglGMI](#) & **operator=** (const [CglGMI](#) &rhs)  
*Assignment operator.*
- virtual ~[CglGMI](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

## Friends

- void [CglGMIUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)

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

## Additional Inherited Members

### 6.14.1 Detailed Description

Gomory cut generator with several cleaning procedures, used to test the numerical safety of the resulting cuts.

Definition at line 37 of file CglGMI.hpp.

### 6.14.2 Member Function Documentation

6.14.2.1 **virtual void CglGMI::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]**

Generate Gomory Mixed-Integer cuts for the model of the solver interface si.

Insert the generated cuts into OsiCuts cs.

Warning: This generator currently works only with the Lp solvers Clp or Cplex9.0 or higher. It requires access to the optimal tableau and optimal basis inverse and makes assumptions on the way slack variables are added by the solver. The Osi implementations for Clp and Cplex verify these assumptions.

When calling the generator, the solver interface si must contain an optimized problem and information related to the optimal basis must be available through the OsiSolverInterface methods (si->optimalBasisIsAvailable() must return 'true'). It is also essential that the integrality of structural variable i can be obtained using si->isInteger(i).

Implements [CglCutGenerator](#).

6.14.2.2 **void CglGMI::setTrackRejection ( bool value )**

Set/get tracking of the rejection of cutting planes.

Note that all rejection related functions will not do anything unless the generator is compiled with the define GMI\_TRACK\_REJECTION

### 6.14.3 Friends And Related Function Documentation

6.14.3.1 **void CglGMIUnitTest ( const OsiSolverInterface \* siP, const std::string mpdDir ) [friend]**

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

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

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

- CglGMI.hpp

## 6.15 CglGMIPParam Class Reference

Class collecting parameters for the GMI cut generator.

```
#include <CglGMIPParam.hpp>
```

Inheritance diagram for CglGMIPParam:

Collaboration diagram for CglGMIPParam:

### Public Types

#### Enumerations

- enum **CleaningProcedure**

### Public Member Functions

#### Set/get methods

- void **setInfinity** (double value)  
*Aliases for parameter get/set method in the base class [CglParam](#).*
- double **getInfinity** () const
- void **setEps** (double value)  
*Epsilon for comparing numbers.*
- double **getEps** () const
- void **setEpsCoeff** (double value)  
*Epsilon for zeroing out coefficients.*
- double **getEpsCoeff** () const
- void **setMaxSupport** (int value)  
*Maximum support of the cutting planes.*
- int **getMaxSupport** () const
- void **setMaxSupportAbs** (int value)  
*Alias for consistency with our naming scheme.*
- int **getMaxSupportAbs** () const
- int **getMAX\_SUPPORT\_ABS** () const
- virtual void **setAway** (double value)  
*Set AWAY, the minimum distance from being integer used for selecting rows for cut generation; all rows whose pivot variable should be integer but is more than away from integrality will be selected; Default: 0.005.*
- double **getAway** () const  
*Get value of away.*
- void **setAWAY** (double value)  
*Aliases.*
- double **getAWAY** () const
- virtual void **setEPS\_ELIM** (double value)  
*Set the value of EPS\_ELIM, epsilon for values of coefficients when eliminating slack variables; Default: 0.*
- double **getEPS\_ELIM** () const  
*Get the value of EPS\_ELIM.*
- void **setEpsElim** (double value)  
*Aliases.*
- double **getEpsElim** () const
- virtual void **setEPS\_RELAX\_ABS** (double value)  
*Set EPS\_RELAX\_ABS.*
- double **getEPS\_RELAX\_ABS** () const

- Get value of EPS\_RELAX\_ABS.*
  - void [setEpsRelaxAbs](#) (double value)
  - Aliases.*
  - double **getEpsRelaxAbs** () const
  - virtual void [setEPS\\_RELAX\\_REL](#) (double value)
  - Set EPS\_RELAX\_REL.*
  - double [getEPS\\_RELAX\\_REL](#) () const
  - Get value of EPS\_RELAX\_REL.*
  - void [setEpsRelaxRel](#) (double value)
  - Aliases.*
  - double **getEpsRelaxRel** () const
  - virtual void **setMAXDYN** (double value)
  - double [getMAXDYN](#) () const
  - Get the value of MAXDYN.*
  - void [setMaxDyn](#) (double value)
  - Aliases.*
  - double **getMaxDyn** () const
  - virtual void [setMINVIOL](#) (double value)
  - Set the value of MINVIOL, the minimum violation for the current basic solution in a generated cut.*
  - double [getMINVIOL](#) () const
  - Get the value of MINVIOL.*
  - void [setMinViol](#) (double value)
  - Aliases.*
  - double **getMinViol** () const
  - virtual void [setMAX\\_SUPPORT\\_REL](#) (double value)
  - Set the value of MAX\_SUPPORT\_REL, the factor contributing to the maximum support relative to the number of columns.*
  - double [getMAX\\_SUPPORT\\_REL](#) () const
  - Get the value of MINVIOL.*
  - void [setMaxSupportRel](#) (double value)
  - Aliases.*
  - double **getMaxSupportRel** () const
  - virtual void [setUSE\\_INTSLACKS](#) (bool value)
  - Set the value of USE\_INTSLACKS.*
  - bool [getUSE\\_INTSLACKS](#) () const
  - Get the value of USE\_INTSLACKS.*
  - void [setUseIntSlacks](#) (bool value)
  - Aliases.*
  - int **getUseIntSlacks** () const
  - virtual void [setCHECK\\_DUPLICATES](#) (bool value)
  - Set the value of CHECK\_DUPLICATES.*
  - bool [getCHECK\\_DUPLICATES](#) () const
  - Get the value of CHECK\_DUPLICATES.*
  - void [setCheckDuplicates](#) (bool value)
  - Aliases.*
  - bool **getCheckDuplicates** () const
  - virtual void [setCLEAN\\_PROC](#) (CleaningProcedure value)
  - Set the value of CLEAN\_PROC.*
  - CleaningProcedure [getCLEAN\\_PROC](#) () const
  - Get the value of CLEAN\_PROC.*
  - void [setCleanProc](#) (CleaningProcedure value)
  - Aliases.*
  - CleaningProcedure **getCleaningProcedure** () const
  - virtual void [setINTEGRAL\\_SCALE\\_CONT](#) (bool value)

- Set the value of `INTEGRAL_SCALE_CONT`.  
• bool `getINTEGRAL_SCALE_CONT` () const  
Get the value of `INTEGRAL_SCALE_CONT`.
- void `setIntegralScaleCont` (bool value)  
Aliases.
- bool `getIntegralScaleCont` () const
- virtual void `setENFORCE_SCALING` (bool value)  
Set the value of `ENFORCE_SCALING`.
- bool `getENFORCE_SCALING` () const  
Get the value of `ENFORCE_SCALING`.
- void `setEnforceScaling` (bool value)  
Aliases.
- bool `getEnforcescalings` () const

### Constructors and destructors

- `CglGMIPParam` (double eps=1e-12, double away=0.005, double eps\_coeff=1e-11, double eps\_elim=0, double eps\_relax\_abs=1e-11, double eps\_relax\_rel=1e-13, double max\_dyn=1e6, double min\_viol=1e-4, int max\_↵ supp\_abs=1000, double max\_supp\_rel=0.1, CleaningProcedure clean\_proc=CP\_CGLLANDP1, bool use\_↵ int\_slacks=false, bool check\_duplicates=false, bool integral\_scale\_cont=false, bool enforce\_scaling=true)  
Default constructor.
- `CglGMIPParam` (`CglParam` &source, double away=0.005, double eps\_elim=1e-12, double eps\_relax\_abs=1e-11, double eps\_relax\_rel=1e-13, double max\_dyn=1e6, double min\_viol=1e-4, double max\_supp\_rel=0.↵ 1, CleaningProcedure clean\_proc=CP\_CGLLANDP1, bool use\_int\_slacks=false, bool check\_duplicates=false, bool integral\_scale\_cont=false, bool enforce\_scaling=true)  
Constructor from `CglParam`.
- `CglGMIPParam` (const `CglGMIPParam` &source)  
Copy constructor.
- virtual `CglGMIPParam` \* `clone` () const  
Clone.
- virtual `CglGMIPParam` & `operator=` (const `CglGMIPParam` &rhs)  
Assignment operator.
- virtual `~CglGMIPParam` ()  
Destructor.

### Protected Attributes

#### Parameters

- double `AWAY`  
Use row only if pivot variable should be integer but is more than `AWAY` from being integer.
- double `EPS_ELIM`  
Epsilon for value of coefficients when eliminating slack variables.
- double `EPS_RELAX_ABS`  
Value added to the right hand side of each generated cut to relax it.
- double `EPS_RELAX_REL`  
For a generated cut with right hand side `rhs_val`, `EPS_RELAX_EPS * fabs(rhs_val)` is used to relax the constraint.
- double `MAXDYN`  
Maximum ratio between largest and smallest non zero coefficients in a cut.
- double `MINVIOL`  
Minimum violation for the current basic solution in a generated cut.
- double `MAX_SUPPORT_REL`  
Maximum support relative to number of columns.

- CleaningProcedure [CLEAN\\_PROC](#)  
*Which cleaning procedure should be used?*
- bool [USE\\_INTSLACKS](#)  
*Use integer slacks to generate cuts if `USE_INTSLACKS` = 1.*
- bool [CHECK\\_DUPLICATES](#)  
*Check for duplicates when adding the cut to the collection?*
- bool [INTEGRAL\\_SCALE\\_CONT](#)  
*Should we try to rescale cut coefficients on continuous variables so that they become integral, or do we only rescale coefficients on integral variables? Used only by cleaning procedure that try the integral scaling.*
- bool [ENFORCE\\_SCALING](#)  
*Should we discard badly scaled cuts (according to the scaling procedure in use)? If false, `CglGMI::scaleCut` always returns true, even though it still scales cuts whenever possible, but not cut is rejected for scaling.*

### 6.15.1 Detailed Description

Class collecting parameters for the GMI cut generator.

Parameters of the generator are listed below. Modifying the default values for parameters other than the last four might result in invalid cuts.

- MAXDYN: Maximum ratio between largest and smallest non zero coefficients in a cut. See method `setMaxDYN()`.
- EPS\_ELIM: Precision for deciding if a coefficient is zero when eliminating slack variables. See method `setEPS←_ELIM()`.
- MINVIOL: Minimum violation for the current basic solution in a generated cut. See method `setMINVIOL()`.
- USE\_INTSLACKS: Use integer slacks to generate cuts. (not implemented yet, will be in the future). See method `setUSE_INTSLACKS()`.
- AWAY: Look only at basic integer variables whose current value is at least this value away from being integer. See method `setAway()`.
- CHECK\_DUPLICATES: Should we check for duplicates when adding a cut to the collection? Can be slow. Default 0 - do not check, add cuts anyway.
- CLEAN\_PROC: Cleaning procedure that should be used. Look below at the enumeration `CleaningProcedure` for possible values.
- INTEGRAL\_SCALE\_CONT: If we try to scale cut coefficients so that they become integral, do we also scale on continuous variables? Default 0 - do not scale continuous vars. Used only if `CLEAN_PROC` does integral scaling.
- ENFORCE\_SCALING: Discard badly scaled cuts, or keep them (unscaled). Default 1 - yes.

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

### 6.15.2 Member Function Documentation

#### 6.15.2.1 `void CglGMIParam::setInfinity ( double value )` `[inline]`

Aliases for parameter get/set method in the base class [CglParam](#).

Value for Infinity. Default: `DBL_MAX`

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



#### 6.15.2.2 void CglGMIPParam::setEps ( double *value* ) [inline]

Epsilon for comparing numbers.

Default: 1.0e-6

Definition at line 84 of file CglGMIPParam.hpp.

#### 6.15.2.3 void CglGMIPParam::setEpsCoeff ( double *value* ) [inline]

Epsilon for zeroing out coefficients.

Default: 1.0e-5

Definition at line 88 of file CglGMIPParam.hpp.

#### 6.15.2.4 void CglGMIPParam::setMaxSupport ( int *value* ) [inline]

Maximum support of the cutting planes.

Default: INT\_MAX

Definition at line 92 of file CglGMIPParam.hpp.

#### 6.15.2.5 virtual void CglGMIPParam::setMINVIOL ( double *value* ) [virtual]

Set the value of MINVIOL, the minimum violation for the current basic solution in a generated cut.

Default: 1e-7

#### 6.15.2.6 virtual void CglGMIPParam::setMax\_SUPPORT\_REL ( double *value* ) [virtual]

Set the value of MAX\_SUPPORT\_REL, the factor contributing to the maximum support relative to the number of columns.

Maximum allowed support is: MAX\_SUPPORT\_ABS + MAX\_SUPPORT\_REL\*ncols. Default: 0.1

#### 6.15.2.7 virtual void CglGMIPParam::setUSE\_INTSLACKS ( bool *value* ) [virtual]

Set the value of USE\_INTSLACKS.

Default: 0

#### 6.15.2.8 virtual void CglGMIPParam::setCHECK\_DUPLICATES ( bool *value* ) [virtual]

Set the value of CHECK\_DUPLICATES.

Default: 0

#### 6.15.2.9 virtual void CglGMIPParam::setCLEAN\_PROC ( CleaningProcedure *value* ) [virtual]

Set the value of CLEAN\_PROC.

Default: CP\_CGLLANDP1

#### 6.15.2.10 `CleaningProcedure CglGMIParam::getCLEAN_PROC ( ) const` `[inline]`

Get the value of CLEAN\_PROC.

Definition at line 184 of file CglGMIParam.hpp.

#### 6.15.2.11 `virtual void CglGMIParam::setINTEGRAL_SCALE_CONT ( bool value )` `[virtual]`

Set the value of INTEGRAL\_SCALE\_CONT.

Default: 0

#### 6.15.2.12 `bool CglGMIParam::getINTEGRAL_SCALE_CONT ( ) const` `[inline]`

Get the value of INTEGRAL\_SCALE\_CONT.

Definition at line 192 of file CglGMIParam.hpp.

#### 6.15.2.13 `virtual void CglGMIParam::setENFORCE_SCALING ( bool value )` `[virtual]`

Set the value of ENFORCE\_SCALING.

Default: 1

#### 6.15.2.14 `bool CglGMIParam::getENFORCE_SCALING ( ) const` `[inline]`

Get the value of ENFORCE\_SCALING.

Definition at line 200 of file CglGMIParam.hpp.

### 6.15.3 Member Data Documentation

#### 6.15.3.1 `double CglGMIParam::AWAY` `[protected]`

Use row only if pivot variable should be integer but is more than AWAY from being integer.

Definition at line 261 of file CglGMIParam.hpp.

#### 6.15.3.2 `double CglGMIParam::EPS_ELIM` `[protected]`

Epsilon for value of coefficients when eliminating slack variables.

Default: 0.

Definition at line 265 of file CglGMIParam.hpp.

#### 6.15.3.3 `double CglGMIParam::EPS_RELAX_ABS` `[protected]`

Value added to the right hand side of each generated cut to relax it.

Default: 1e-11

Definition at line 269 of file CglGMIParam.hpp.

**6.15.3.4** `double CglGMIPParam::EPS_RELAX_REL` `[protected]`

For a generated cut with right hand side `rhs_val`, `EPS_RELAX_EPS * fabs(rhs_val)` is used to relax the constraint.

Default: 1.e-13

Definition at line 274 of file `CglGMIPParam.hpp`.

**6.15.3.5** `double CglGMIPParam::MAXDYN` `[protected]`

Maximum ratio between largest and smallest non zero coefficients in a cut.

Default: 1e6.

Definition at line 278 of file `CglGMIPParam.hpp`.

**6.15.3.6** `double CglGMIPParam::MINVIOL` `[protected]`

Minimum violation for the current basic solution in a generated cut.

Default: 1e-4.

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

**6.15.3.7** `double CglGMIPParam::MAX_SUPPORT_REL` `[protected]`

Maximum support relative to number of columns.

Must be between 0 and 1. Default: 0.

Definition at line 286 of file `CglGMIPParam.hpp`.

**6.15.3.8** `bool CglGMIPParam::USE_INTSLACKS` `[protected]`

Use integer slacks to generate cuts if `USE_INTSLACKS = 1`.

Default: 0.

Definition at line 292 of file `CglGMIPParam.hpp`.

**6.15.3.9** `bool CglGMIPParam::INTEGRAL_SCALE_CONT` `[protected]`

Should we try to rescale cut coefficients on continuous variables so that they become integral, or do we only rescale coefficients on integral variables? Used only by cleaning procedure that try the integral scaling.

Definition at line 301 of file `CglGMIPParam.hpp`.

**6.15.3.10** `bool CglGMIPParam::ENFORCE_SCALING` `[protected]`

Should we discard badly scaled cuts (according to the scaling procedure in use)? If false, `CglGMI::scaleCut` always returns true, even though it still scales cuts whenever possible, but not cut is rejected for scaling.

Default true. Used only by cleaning procedure that try to scale.

Definition at line 308 of file `CglGMIPParam.hpp`.

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

- CglGMIParam.hpp

## 6.16 CglGomory Class Reference

Gomory Cut Generator Class.

```
#include <CglGomory.hpp>
```

Inheritance diagram for CglGomory:

Collaboration diagram for CglGomory:

### Public Member Functions

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate Mixed Integer Gomory cuts for the model of the solver interface, si.*
- int [generateCuts](#) (const OsiRowCutDebugger \*debugger, OsiCuts &cs, const **CoinPackedMatrix** &columnCopy, const **CoinPackedMatrix** &rowCopy, const double \*colsol, const double \*colLower, const double \*colUpper, const double \*rowLower, const double \*rowUpper, const char \*intVar, const **CoinWarmStartBasis** \*warm, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generates cuts given matrix and solution etc, returns number of cuts generated.*
- int [generateCuts](#) (const OsiRowCutDebugger \*debugger, OsiCuts &cs, const **CoinPackedMatrix** &columnCopy, const double \*colsol, const double \*colLower, const double \*colUpper, const double \*rowLower, const double \*rowUpper, const char \*intVar, const **CoinWarmStartBasis** \*warm, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generates cuts given matrix and solution etc, returns number of cuts generated (no row copy passed in)*
- virtual bool [needsOptimalBasis](#) () const  
*Return true if needs optimal basis to do cuts (will return true)*

#### Change way Gomory works

- void [passInOriginalSolver](#) (OsiSolverInterface \*solver)  
*Pass in a copy of original solver (clone it)*
- OsiSolverInterface \* [originalSolver](#) () const  
*Returns original solver.*
- void [setGomoryType](#) (int type)  
*Set type - 0 normal, 1 add original matrix one, 2 replace.*
- int [gomoryType](#) () const  
*Return type.*

#### Change limit on how many variables in cut (default 50)

- void [setLimit](#) (int limit)  
*Set.*
- int [getLimit](#) () const  
*Get.*
- void [setLimitAtRoot](#) (int limit)  
*Set at root (if < normal then use normal)*
- int [getLimitAtRoot](#) () const  
*Get at root.*

- virtual int [maxLengthOfCutInTree](#) () const  
*Return maximum length of cut in tree.*

#### Change criterion on which variables to look at. All ones

*more than "away" away from integrality will be investigated (default 0.05)*

- void [setAway](#) (double value)  
*Set away.*
- double [getAway](#) () const  
*Get away.*
- void [setAwayAtRoot](#) (double value)  
*Set away at root.*
- double [getAwayAtRoot](#) () const  
*Get away at root.*

#### Change criterion on which the cut id relaxed if the code

*thinks the factorization has inaccuracies.*

*The relaxation to RHS is smallest of - 1) 1.0e-4 2) conditionNumberMultiplier \* condition number of factorization 3) largestFactorMultiplier \* largest (dual\*element) forming tableau row*

- void [setConditionNumberMultiplier](#) (double value)  
*Set ConditionNumberMultiplier.*
- double [getConditionNumberMultiplier](#) () const  
*Get ConditionNumberMultiplier.*
- void [setLargestFactorMultiplier](#) (double value)  
*Set LargestFactorMultiplier.*
- double [getLargestFactorMultiplier](#) () const  
*Get LargestFactorMultiplier.*

#### change factorization

- void [useAlternativeFactorization](#) (bool yes=true)  
*Set/unset alternative factorization.*
- bool [alternativeFactorization](#) () const  
*Get whether alternative factorization being used.*

#### Constructors and destructors

- [CglGomory](#) ()  
*Default constructor.*
- [CglGomory](#) (const [CglGomory](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglGomory](#) & [operator=](#) (const [CglGomory](#) &rhs)  
*Assignment operator.*
- virtual [~CglGomory](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [refreshSolver](#) (OsiSolverInterface \*solver)  
*This can be used to refresh any inforamtion.*

## Friends

- void [CglGomoryUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
A function that tests the methods in the [CglGomory](#) class.

## Additional Inherited Members

### 6.16.1 Detailed Description

Gomory Cut Generator Class.

Definition at line 14 of file CglGomory.hpp.

### 6.16.2 Member Function Documentation

6.16.2.1 virtual void CglGomory::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]

Generate Mixed Integer Gomory cuts for the model of the solver interface, si.

Insert the generated cuts into OsiCut, cs.

There is a limit option, which will only generate cuts with less than this number of entries.

We can also only look at 0-1 variables a certain distance from integer.

Implements [CglCutGenerator](#).

### 6.16.3 Friends And Related Function Documentation

6.16.3.1 void CglGomoryUnitTest ( const OsiSolverInterface \* siP, const std::string mpdDir ) [friend]

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

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

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

- CglGomory.hpp

## 6.17 CglHashLink Struct Reference

Only store unique row cuts.

```
#include <CglPreProcess.hpp>
```

### 6.17.1 Detailed Description

Only store unique row cuts.

Definition at line 456 of file CglPreProcess.hpp.

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

- CglPreProcess.hpp

## 6.18 CgIImplication Class Reference

This just uses implication info.

```
#include <CglProbing.hpp>
```

Inheritance diagram for CgIImplication:

Collaboration diagram for CgIImplication:

### Public Member Functions

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate cuts from implication table Insert generated cuts into the cut set cs.*

#### Constructors and destructors

- [CgIImplication](#) ()  
*Default constructor.*
- [CgIImplication](#) ([CglTreeProbingInfo](#) \*info)  
*Constructor with info.*
- [CgIImplication](#) (const [CgIImplication](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CgIImplication](#) & [operator=](#) (const [CgIImplication](#) &rhs)  
*Assignment operator.*
- virtual [~CgIImplication](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

#### Set implication

- void [setProbingInfo](#) ([CglTreeProbingInfo](#) \*info)  
*Set implication.*

### Additional Inherited Members

#### 6.18.1 Detailed Description

This just uses implication info.

Definition at line 490 of file CglProbing.hpp.

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

- CglProbing.hpp

## 6.19 CglKnapsackCover Class Reference

Knapsack Cover Cut Generator Class.

```
#include <CglKnapsackCover.hpp>
```

Inheritance diagram for CglKnapsackCover:

Collaboration diagram for CglKnapsackCover:

### Public Member Functions

- void [setTestedRowIndices](#) (int num, const int \*ind)  
*A method to set which rows should be tested for knapsack covers.*

### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate knapsack cover cuts for the model of the solver interface, si.*

### Constructors and destructors

- [CglKnapsackCover](#) ()  
*Default constructor.*
- [CglKnapsackCover](#) (const [CglKnapsackCover](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglKnapsackCover](#) & [operator=](#) (const [CglKnapsackCover](#) &rhs)  
*Assignment operator.*
- virtual [~CglKnapsackCover](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [refreshSolver](#) (OsiSolverInterface \*solver)  
*This can be used to refresh any information.*

### Sets and gets

- void [setMaxInKnapsack](#) (int value)  
*Set limit on number in knapsack.*
- int [getMaxInKnapsack](#) () const  
*get limit on number in knapsack*
- void [switchOffExpensive](#) ()  
*Switch off expensive cuts.*
- void [switchOnExpensive](#) ()  
*Switch on expensive cuts.*

### Friends

- void [CglKnapsackCoverUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
*A function that tests the methods in the [CglKnapsackCover](#) class.*



## Private methods

- `int createCliques (OsiSolverInterface &si, int minimumSize=2, int maximumSize=100, bool extendCliques=false)`  
*Creates cliques for use by probing.*

## Additional Inherited Members

### 6.19.1 Detailed Description

Knapsack Cover Cut Generator Class.

Definition at line 15 of file CglKnapsackCover.hpp.

### 6.19.2 Member Function Documentation

6.19.2.1 `virtual void CglKnapsackCover::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]`

Generate knapsack cover cuts for the model of the solver interface, si.

Insert the generated cuts into OsiCut, cs.

Implements [CglCutGenerator](#).

6.19.2.2 `int CglKnapsackCover::createCliques ( OsiSolverInterface & si, int minimumSize = 2, int maximumSize = 100, bool extendCliques = false )`

Creates cliques for use by probing.

Only cliques  $\geq$  minimumSize and  $<$  maximumSize created Can also try and extend cliques as a result of probing (root node). Returns number of cliques found.

### 6.19.3 Friends And Related Function Documentation

6.19.3.1 `void CglKnapsackCoverUnitTest ( const OsiSolverInterface * siP, const std::string mpdDir ) [friend]`

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

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

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

- CglKnapsackCover.hpp

## 6.20 CglLandP Class Reference

Inheritance diagram for CglLandP:

Collaboration diagram for CglLandP:

## Classes

- class [NoBasisError](#)
- class [Parameters](#)  
*Class storing parameters.*
- class [SimplexInterfaceError](#)

## Public Types

- enum [SelectionRules](#) { [mostNegativeRc](#), [bestPivot](#), [initialReducedCosts](#) }
- enum [ExtraCutsMode](#) { [none](#), [AtOptimalBasis](#), [WhenEnteringBasis](#), [AllViolatedMigs](#) }
- enum [SeparationSpaces](#) { [Fractional\\_rc](#), [Full](#) }  
*Space where cuts are optimized.*
- enum [Normalization](#)  
*Normalization.*
- enum [RhsWeightType](#) { [Dynamic](#) }  
*RHS weight in normalization.*

## Public Member Functions

- [CglLandP](#) (const [CglLandP::Parameters](#) &params=[CglLandP::Parameters](#)(), const [LAP::Validator](#) &validator=[LAP::Validator](#)())  
*Constructor for the class.*
- [~CglLandP](#) ()  
*Destructor.*
- [CglLandP](#) (const [CglLandP](#) &source)  
*Copy constructor.*
- [CglLandP](#) & [operator=](#) (const [CglLandP](#) &rhs)  
*Assignment operator.*
- [CglCutGenerator](#) \* [clone](#) () const  
*Clone function.*
- virtual bool [needsOptimalBasis](#) () const  
*Return true if needs optimal basis to do cuts.*
- void [setLogLevel](#) (int level)  
*set level of log for cut generation procedure :*

## Generate Cuts

- virtual void [generateCuts](#) (const [OsiSolverInterface](#) &si, [OsiCuts](#) &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate cuts for the model data contained in si.*

## Friends

- class [LAP::CglLandPSimplex](#)

## Additional Inherited Members

### 6.20.1 Detailed Description

Definition at line 49 of file CglLandP.hpp.

### 6.20.2 Member Enumeration Documentation

#### 6.20.2.1 enum CglLandP::SelectionRules

Enumerator

- mostNegativeRc*** select most negative reduced cost
- bestPivot*** select best possible pivot.
- initialReducedCosts*** Select only those rows which had initially a 0 reduced cost.

Definition at line 58 of file CglLandP.hpp.

#### 6.20.2.2 enum CglLandP::ExtraCutsMode

Enumerator

- none*** Generate no extra cuts.
- AtOptimalBasis*** Generate cuts from the optimal basis.
- WhenEnteringBasis*** Generate cuts as soon as a structural enters the basis.
- AllViolatedMigs*** Generate all violated Mixed integer Gomory cuts in the course of the optimization.

Definition at line 65 of file CglLandP.hpp.

#### 6.20.2.3 enum CglLandP::SeparationSpaces

Space where cuts are optimized.

Enumerator

- Fractional\_rc*** Use fractional space only for computing reduced costs.
- Full*** Work in full space.

Definition at line 74 of file CglLandP.hpp.

#### 6.20.2.4 enum CglLandP::RhsWeightType

RHS weight in normalization.

Enumerator

- Dynamic*** 2 \* current number of constraints.

Definition at line 100 of file CglLandP.hpp.

### 6.20.3 Member Function Documentation

6.20.3.1 `virtual void CgILandP::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]`

Generate cuts for the model data contained in si.

The generated cuts are inserted into and returned in the collection of cuts cs.

Implements [CglCutGenerator](#).

6.20.3.2 `void CgILandP::setLogLevel ( int level ) [inline]`

set level of log for cut generation procedure :

1. for none
2. for log at begin and end of procedure + at some time interval
3. for log at every cut generated

Definition at line 213 of file CgILandP.hpp.

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

- CgILandP.hpp

## 6.21 LAP::CgILandPSimplex Class Reference

### Public Member Functions

- [CgILandPSimplex](#) (const OsiSolverInterface &si, const CgILandP::CachedData &cached, const [CgILandP::Parameters](#) &params, [Validator](#) &validator)  
*Usefull onstructor.*
- [~CgILandPSimplex](#) ()  
*Destructor.*
- void [cacheUpdate](#) (const CgILandP::CachedData &cached, bool reducedSpace=0)  
*Update cached information in case of basis change in a round.*
- bool [resetSolver](#) (const **CoinWarmStartBasis** \*basis)  
*reset the solver to optimal basis*
- bool [optimize](#) (int var, OsiRowCut &cut, const CgILandP::CachedData &cached, const [CgILandP::Parameters](#) &params)  
*Perform pivots to find the best cuts.*
- bool [generateMig](#) (int row, OsiRowCut &cut, const [CgILandP::Parameters](#) &params)  
*Find Gomory cut (i.e.*
- int [generateExtraCuts](#) (const CgILandP::CachedData &cached, const [CgILandP::Parameters](#) &params)  
*Find extra constraints in current tableau.*
- int [generateExtraCut](#) (int i, const CgILandP::CachedData &cached, const [CgILandP::Parameters](#) &params)  
*Generate a constrainte for a row of the tableau different from the source row.*
- int [insertAllExtr](#) (OsiCuts &cs, **CoinRelFitEq** eq)  
*insert all extra cuts in cs.*

## Protected Member Functions

- bool [changeBasis](#) (int incoming, int leaving, int direction, bool modularize)  
*Perform a change in the basis (direction is 1 if leaving variable is going to ub, 0 otherwise)*
- int [fastFindCutImprovingPivotRow](#) (int &direction, int &gammaSign, double tolerance, bool flagPositiveRows)  
*Find a row which can be used to perform an improving pivot the fast way (i.e., find the leaving variable).*
- int [rescanReducedCosts](#) (int &direction, int &gammaSign, double tolerance)  
*Rescan reduced costs tables.*
- int [fastFindBestPivotColumn](#) (int direction, int gammaSign, double pivotTol, double rhsTol, bool reducedSpace, bool allowNonImproving, double &bestSigma, bool modularize)  
*Find the column which leads to the best cut (i.e., find incoming variable).*
- int [findBestPivot](#) (int &leaving, int &direction, const [CgILandP::Parameters](#) &params)  
*Find incoming and leaving variables which lead to the most violated adjacent normalized lift-and-project cut.*
- double [computeCglpObjective](#) (const [TabRow](#) &row, bool modularize=false) const  
*Compute the objective value of the Cglp for given row and rhs (if strengthening shall be applied row should have been modularized).*
- double [strengthenedIntersectionCutCoef](#) (int i, double alpha\_i, double beta) const  
*return the coefficients of the strengthened intersection cut takes one extra argument seems needs to consider variable type.*
- double [newRowCoefficient](#) (int j, double gamma) const  
*return the coefficient of the new row (combining row\_k + gamma row\_i).*
- void [createIntersectionCut](#) ([TabRow](#) &row, [OsiRowCut](#) &cut) const  
*Create the intersection cut of row k.*
- double [normalizationFactor](#) (const [TabRow](#) &row) const  
*Compute the normalization factor of the cut.*
- void [scaleCut](#) ([OsiRowCut](#) &cut, double factor) const  
*Scale the cut by factor.*
- void [createMIG](#) ([TabRow](#) &row, [OsiRowCut](#) &cut) const  
*Create strenghtened row.*
- void [pullTableauRow](#) ([TabRow](#) &row) const  
*Get the row i of the tableau.*
- void [adjustTableauRow](#) (int var, [TabRow](#) &row, int direction)  
*Adjust the row of the tableau to reflect leaving variable direction.*
- void [resetOriginalTableauRow](#) (int var, [TabRow](#) &row, int direction)  
*reset the tableau row after a call to adjustTableauRow*
- double [getLoBound](#) (int index) const  
*Get lower bound for variable or constraint.*
- double [getUpBound](#) (int index) const  
*Get upper bound for variable or constraint.*
- double [getColsolToCut](#) (int index) const  
*Access to value in solution to cut (indexed in reduced problem)*
- void [setColsolToCut](#) (int index, double value)  
*Access to value in solution to cut (indexed in reduced problem)*
- **CoinWarmStartBasis::Status** [getStatus](#) (int index) const  
*Get the basic status of a variable (structural or slack).*
- bool [isInteger](#) (int index) const  
*Say if variable index by i in current tableau is integer.*

- void `computeWeights` (CglLandP::LHSnorm norm, CglLandP::Normalization type, CglLandP::RhsWeightType rhs)  
*Compute normalization weights.*
- double `normedCoef` (double a, int ii) const  
*Evenutaly multiply a by w if normed\_weights\_ is not empty.*
- void `printTableau` (std::ostream &os)  
*print the tableau of current basis.*
- void `printEverything` ()  
*Print everything .*
- void `printTableauLateX` (std::ostream &os)  
*print the tableau of current basis.*
- void `printCglpBasis` (std::ostream &os=std::cout)  
*Print CGLP basis corresponding to current tableau and source row.*
- void `get_M1_M2_M3` (const TabRow &row, std::vector< int > &M1, std::vector< int > &M2, std::vector< int > &M3)  
*Put variables in M1 M2 and M3 according to their sign.*
- void `eliminate_slacks` (double \*vec) const  
*Put a vector in structural sapce.*

#### Slow versions of the function (old versions do not work).

- double `computeCglpRedCost` (int direction, int gammaSign, double tau)  
*Compute the reduced cost of Cglp.*
- double `computeRedCostConstantsInRow` ()  
*Compute the value of sigma and thau (which are constants for a row i as defined in Mike Perregaard thesis.*
- double `computeCglpObjective` (double gamma, bool strengthen, TabRow &row)  
*Compute the objective value of the Cglp with linear combintation of the two rows by gamma.*
- double `computeCglpObjective` (double gamma, bool strengthen)  
*Compute the objective value of the Cglp with linear combintation of the row\_k\_ and gamma row\_i\_.*
- int `findCutImprovingPivotRow` (int &direction, int &gammaSign, double tolerance)  
*Find a row which can be used to perform an improving pivot return index of the cut or -1 if none exists (i.e., find the leaving variable).*
- int `findBestPivotColumn` (int direction, double pivotTol, bool reducedSpace, bool allowDegeneratePivot, bool modularize)  
*Find the column which leads to the best cut (i.e., find incoming variable).*
- int `plotCGLPobj` (int direction, double gammaTolerance, double pivotTol, bool reducedSpace, bool allow← Degenerate, bool modularize)

### 6.21.1 Detailed Description

Definition at line 42 of file CglLandPSimplex.hpp.

### 6.21.2 Member Function Documentation

#### 6.21.2.1 bool LAP::CglLandPSimplex::generateMig ( int row, OsiRowCut & cut, const CglLandP::Parameters & params )

Find Gomory cut (i.e.

don't do extra setup required for pivots).

6.21.2.2 `int LAP::CglLandPSimplex::generateExtraCuts ( const CglLandP::CachedData & cached, const CglLandP::Parameters & params )`

Find extra constraints in current tableau.

6.21.2.3 `int LAP::CglLandPSimplex::generateExtraCut ( int i, const CglLandP::CachedData & cached, const CglLandP::Parameters & params )`

Generate a constraint for a row of the tableau different from the source row.

6.21.2.4 `int LAP::CglLandPSimplex::insertAllExtr ( OsiCuts & cs, CoinRelFltEq eq )`

insert all extra cuts in *cs*.

6.21.2.5 `int LAP::CglLandPSimplex::fastFindCutImprovingPivotRow ( int & direction, int & gammaSign, double tolerance, bool flagPositiveRows )` [protected]

Find a row which can be used to perform an improving pivot the fast way (i.e., find the leaving variable).

Returns

index of the row on which to pivot or -1 if none exists.

6.21.2.6 `int LAP::CglLandPSimplex::fastFindBestPivotColumn ( int direction, int gammaSign, double pivotTol, double rhsTol, bool reducedSpace, bool allowNonImproving, double & bestSigma, bool modularize )` [protected]

Find the column which leads to the best cut (i.e., find incoming variable).

6.21.2.7 `int LAP::CglLandPSimplex::findBestPivot ( int & leaving, int & direction, const CglLandP::Parameters & params )` [protected]

Find incoming and leaving variables which lead to the most violated adjacent normalized lift-and-project cut.

Remarks

At this point reduced costs should be already computed.

Returns

incoming variable variable,

Parameters

<i>leaving</i>	variable
<i>direction</i>	leaving direction

6.21.2.8 `double LAP::CglLandPSimplex::computeCglpObjective ( const TabRow & row, bool modularize = false )` const [protected]

Compute the objective value of the Cglp for given row and rhs (if strengthening shall be applied row should have been modularized).

**6.21.2.9** `double LAP::CglLandPSimplex::strengthenedIntersectionCutCoef ( int i, double alpha_i, double beta ) const`  
`[inline], [protected]`

return the coefficients of the strengthened intersection cut takes one extra argument seems needs to consider variable type.

return the coefficients of the strengthened intersection cut

Definition at line 426 of file CglLandPSimplex.hpp.

**6.21.2.10** `double LAP::CglLandPSimplex::normalizationFactor ( const TabRow & row ) const` `[protected]`

Compute the normalization factor of the cut.

**6.21.2.11** `void LAP::CglLandPSimplex::scaleCut ( OsiRowCut & cut, double factor ) const` `[protected]`

Scale the cut by factor.

**6.21.2.12** `void LAP::CglLandPSimplex::createMIG ( TabRow & row, OsiRowCut & cut ) const` `[protected]`

Create strenghtened row.

Create MIG cut from row k

**6.21.2.13** `CoinWarmStartBasis::Status LAP::CglLandPSimplex::getStatus ( int index ) const` `[inline], [protected]`

Get the basic status of a variable (structural or slack).

Definition at line 229 of file CglLandPSimplex.hpp.

**6.21.2.14** `bool LAP::CglLandPSimplex::isInteger ( int index ) const` `[inline], [protected]`

Say if variable index by i in current tableau is integer.

Definition at line 235 of file CglLandPSimplex.hpp.

**6.21.2.15** `void LAP::CglLandPSimplex::computeWeights ( CglLandP::LHSnorm norm, CglLandP::Normalization type, CglLandP::RhsWeightType rhs )` `[protected]`

Compute normalization weights.

**6.21.2.16** `double LAP::CglLandPSimplex::normedCoef ( double a, int ii ) const` `[inline], [protected]`

Evenutaly multiply a by w if normed\_weights\_ is not empty.

Definition at line 243 of file CglLandPSimplex.hpp.

**6.21.2.17** `void LAP::CglLandPSimplex::printTableau ( std::ostream & os )` `[protected]`

print the tableau of current basis.



6.21.2.18 void LAP::CglLandPSimplex::printEverything ( ) [protected]

Print everything .

6.21.2.19 void LAP::CglLandPSimplex::printTableauLateX ( std::ostream & os ) [protected]

print the tableau of current basis.

6.21.2.20 void LAP::CglLandPSimplex::printCglpBasis ( std::ostream & os = std::cout ) [protected]

Print CGLP basis corresponding to current tableau and source row.

6.21.2.21 void LAP::CglLandPSimplex::get\_M1\_M2\_M3 ( const TabRow & row, std::vector< int > & M1, std::vector< int > & M2, std::vector< int > & M3 ) [protected]

Put variables in M1 M2 and M3 according to their sign.

6.21.2.22 void LAP::CglLandPSimplex::eliminate\_slacks ( double \* vec ) const [protected]

Put a vector in structural sapce.

6.21.2.23 int LAP::CglLandPSimplex::findCutImprovingPivotRow ( int & direction, int & gammaSign, double tolerance ) [protected]

Find a row which can be used to perform an improving pivot return index of the cut or -1 if none exists (i.e., find the leaving variable).

6.21.2.24 int LAP::CglLandPSimplex::findBestPivotColumn ( int direction, double pivotTol, bool reducedSpace, bool allowDegeneratePivot, bool modularize ) [protected]

Find the column which leads to the best cut (i.e., find incoming variable).

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

- CglLandPSimplex.hpp

## 6.22 CglLiftAndProject Class Reference

Lift And Project Cut Generator Class.

```
#include <CglLiftAndProject.hpp>
```

Inheritance diagram for CglLiftAndProject:

Collaboration diagram for CglLiftAndProject:

### Public Member Functions

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
Generate lift-and-project cuts for the model of the solver interface, si.
- double [getBeta](#) () const  
Get the normalization : Either beta=+1 or beta=-1.
- void [setBeta](#) (int oneOrMinusOne)  
Set the normalization : Either beta=+1 or beta=-1.

### Constructors and destructors

- [CglLiftAndProject](#) ()  
Default constructor.
- [CglLiftAndProject](#) (const [CglLiftAndProject](#) &)  
Copy constructor.
- virtual [CglCutGenerator](#) \* [clone](#) () const  
Clone.
- [CglLiftAndProject](#) & [operator=](#) (const [CglLiftAndProject](#) &rhs)  
Assignment operator.
- virtual [~CglLiftAndProject](#) ()  
Destructor.
- virtual std::string [generateCpp](#) (FILE \*fp)  
Create C++ lines to get to current state.

### Friends

- void [CglLiftAndProjectUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
A function that tests the methods in the [CglLiftAndProject](#) class.

### Additional Inherited Members

#### 6.22.1 Detailed Description

Lift And Project Cut Generator Class.

Definition at line 13 of file [CglLiftAndProject.hpp](#).

#### 6.22.2 Member Function Documentation

6.22.2.1 virtual void [CglLiftAndProject::generateCuts](#) ( const OsiSolverInterface & si, OsiCuts & cs, const [CglTreeInfo](#) info = [CglTreeInfo](#) () ) [virtual]

Generate lift-and-project cuts for the model of the solver interface, si.

Insert the generated cuts into OsiCut, cs.

Implements [CglCutGenerator](#).

6.22.2.2 void [CglLiftAndProject::setBeta](#) ( int *oneOrMinusOne* ) [inline]

Set the normalization : Either beta=+1 or beta=-1.

Default value is 1.

Definition at line 37 of file [CglLiftAndProject.hpp](#).

### 6.22.3 Friends And Related Function Documentation

6.22.3.1 `void CglLiftAndProjectUnitTest ( const OsiSolverInterface * siP, const std::string mpdDir ) [friend]`

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

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

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

- [CglLiftAndProject.hpp](#)

## 6.23 CglMessage Class Reference

This deals with Cgl messages (as against Osi messages etc)

```
#include <CglMessage.hpp>
```

Inheritance diagram for CglMessage:

Collaboration diagram for CglMessage:

### Public Member Functions

#### Constructors etc

- [CglMessage](#) (**Language** *language*=us\_en)  
*Constructor.*

### 6.23.1 Detailed Description

This deals with Cgl messages (as against Osi messages etc)

Definition at line 38 of file [CglMessage.hpp](#).

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

- [CglMessage.hpp](#)

## 6.24 CglMixedIntegerRounding Class Reference

Mixed Integer Rounding Cut Generator Class.

```
#include <CglMixedIntegerRounding.hpp>
```

Inheritance diagram for CglMixedIntegerRounding:

Collaboration diagram for CglMixedIntegerRounding:

### Public Member Functions

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate Mixed Integer Rounding cuts for the model data contained in si.*

### Constructors and destructors

- [CglMixedIntegerRounding](#) ()  
*Default constructor.*
- [CglMixedIntegerRounding](#) (const int maxaggr, const bool multiply, const int criterion, const int preproc=-1)  
*Alternate Constructor.*
- [CglMixedIntegerRounding](#) (const [CglMixedIntegerRounding](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglMixedIntegerRounding](#) & [operator=](#) (const [CglMixedIntegerRounding](#) &rhs)  
*Assignment operator.*
- virtual [~CglMixedIntegerRounding](#) ()  
*Destructor.*
- virtual void [refreshSolver](#) (OsiSolverInterface \*solver)  
*This can be used to refresh any information.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

### Set and get methods

- void [setMAXAGGR\\_](#) (int maxaggr)  
*Set MAXAGGR\_.*
- int [getMAXAGGR\\_](#) () const  
*Get MAXAGGR\_.*
- void [setMULTIPLY\\_](#) (bool multiply)  
*Set MULTIPLY\_.*
- bool [getMULTIPLY\\_](#) () const  
*Get MULTIPLY\_.*
- void [setCRITERION\\_](#) (int criterion)  
*Set CRITERION\_.*
- int [getCRITERION\\_](#) () const  
*Get CRITERION\_.*
- void [setDoPreproc](#) (int value)  
*Set doPreproc.*
- bool [getDoPreproc](#) () const  
*Get doPreproc.*

### Additional Inherited Members

#### 6.24.1 Detailed Description

Mixed Integer Rounding Cut Generator Class.

Definition at line 86 of file [CglMixedIntegerRounding.hpp](#).

### 6.24.2 Member Function Documentation

6.24.2.1 `virtual void CglMixedIntegerRounding2::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo () ) [virtual]`

Generate Mixed Integer Rounding cuts for the model data contained in si.

The generated cuts are inserted in the collection of cuts cs.

Implements [CglCutGenerator](#).

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

- `CglMixedIntegerRounding2.hpp`

## 6.25 CglMixedIntegerRounding2 Class Reference

Mixed Integer Rounding Cut Generator Class.

```
#include <CglMixedIntegerRounding2.hpp>
```

Inheritance diagram for CglMixedIntegerRounding2:

Collaboration diagram for CglMixedIntegerRounding2:

### Public Member Functions

#### Generate Cuts

- `virtual void generateCuts (const OsiSolverInterface &si, OsiCuts &cs, const CglTreeInfo info=CglTreeInfo())`  
*Generate Mixed Integer Rounding cuts for the model data contained in si.*

#### Constructors and destructors

- `CglMixedIntegerRounding2 ()`  
*Default constructor.*
- `CglMixedIntegerRounding2 (const int maxaggr, const bool multiply, const int criterion, const int preproc=-1)`  
*Alternate Constructor.*
- `CglMixedIntegerRounding2 (const CglMixedIntegerRounding2 &)`  
*Copy constructor.*
- `virtual CglCutGenerator * clone () const`  
*Clone.*
- `CglMixedIntegerRounding2 & operator= (const CglMixedIntegerRounding2 &rhs)`  
*Assignment operator.*
- `virtual ~CglMixedIntegerRounding2 ()`  
*Destructor.*
- `virtual void refreshSolver (OsiSolverInterface *solver)`  
*This can be used to refresh any inforamtion.*
- `virtual std::string generateCpp (FILE *fp)`  
*Create C++ lines to get to current state.*

#### Set and get methods

- `void setMAXAGGR\_ (int maxaggr)`

- Set MAXAGGR\_.*
- int [getMAXAGGR\\_](#) () const
- Get MAXAGGR\_.*
- void [setMULTIPLY\\_](#) (bool multiply)
- Set MULTIPLY\_.*
- bool [getMULTIPLY\\_](#) () const
- Get MULTIPLY\_.*
- void [setCRITERION\\_](#) (int criterion)
- Set CRITERION\_.*
- int [getCRITERION\\_](#) () const
- Get CRITERION\_.*
- void [setDoPreproc](#) (int value)
- Set doPreproc.*
- bool [getDoPreproc](#) () const
- Get doPreproc.*

## Additional Inherited Members

### 6.25.1 Detailed Description

Mixed Integer Rounding Cut Generator Class.

Definition at line 87 of file `CglMixedIntegerRounding2.hpp`.

### 6.25.2 Member Function Documentation

- 6.25.2.1 `virtual void CglMixedIntegerRounding2::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo () ) [virtual]`

Generate Mixed Integer Rounding cuts for the model data contained in si.

The generated cuts are inserted in the collection of cuts cs.

Implements [CglCutGenerator](#).

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

- `CglMixedIntegerRounding2.hpp`

## 6.26 CglMixIntRoundVUB Class Reference

### 6.26.1 Detailed Description

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

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

- `CglMixedIntegerRounding.hpp`

## 6.27 CglMixIntRoundVUB2 Class Reference

### 6.27.1 Detailed Description

Definition at line 33 of file CglMixedIntegerRounding2.hpp.

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

- CglMixedIntegerRounding2.hpp

## 6.28 CglOddHole Class Reference

Odd Hole Cut Generator Class.

```
#include <CglOddHole.hpp>
```

Inheritance diagram for CglOddHole:

Collaboration diagram for CglOddHole:

### Public Member Functions

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate odd hole cuts for the model of the solver interface, si.*

#### Create Row List

- void [createRowList](#) (const OsiSolverInterface &si, const int \*possible=NULL)  
*Create a list of rows which might yield cuts this is to speed up process The possible parameter is a list to cut down search.*
- void [createRowList](#) (int numberOfRows, const int \*whichRow)  
*This version passes in a list - 1 marks possible.*

#### Create Clique List

- void [createCliqueList](#) (int numberCliques, const int \*cliqueStart, const int \*cliqueMember)  
*Create a list of extra row cliques which may not be in matrix At present these are classical cliques.*

#### Number Possibilities

- int [numberPossible](#) ()  
*Returns how many rows might give odd hole cuts.*

#### Gets and Sets

- double [getMinimumViolation](#) () const  
*Minimum violation.*
- void [setMinimumViolation](#) (double value)
- double [getMinimumViolationPer](#) () const  
*Minimum violation per entry.*
- void [setMinimumViolationPer](#) (double value)

- int [getMaximumEntries](#) () const  
*Maximum number of entries in a cut.*
- void **setMaximumEntries** (int value)

### Constructors and destructors

- [CglOddHole](#) ()  
*Default constructor.*
- [CglOddHole](#) (const [CglOddHole](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglOddHole](#) & **operator=** (const [CglOddHole](#) &rhs)  
*Assignment operator.*
- virtual [~CglOddHole](#) ()  
*Destructor.*
- virtual void [refreshSolver](#) (OsiSolverInterface \*solver)  
*This can be used to refresh any inforamtion.*

### Friends

- void [CglOddHoleUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
*A function that tests the methods in the [CglOddHole](#) class.*

### Additional Inherited Members

#### 6.28.1 Detailed Description

Odd Hole Cut Generator Class.

Definition at line 14 of file CglOddHole.hpp.

#### 6.28.2 Member Function Documentation

6.28.2.1 virtual void [CglOddHole::generateCuts](#) ( const OsiSolverInterface & *si*, OsiCuts & *cs*, const [CglTreeInfo](#) *info* = [CglTreeInfo](#) () ) [virtual]

Generate odd hole cuts for the model of the solver interface, si.

This looks at all rows of type  $\sum x(i) \leq 1$  (or  $\sum x(i) = 1$ ) ( $x(i) \in \{0,1\}$ ) and sees if there is an odd cycle cut. See Grotschel, Lovasz and Schrijver (1988) for method. This is then lifted by using the corresponding Chvatal cut i.e. Take all rows in cycle and add them together. RHS will be odd so weaken all odd coefficients so 1.0 goes to 0.0 etc - then constraint is  $\sum \text{even}(j) * x(j) \leq \text{odd}$  which can be replaced by  $\sum \text{even}(j) * x(j) \leq (\text{odd}-1.0)/2$ . A similar cut can be generated for  $\sum x(i) \geq 1$ .

Insert the generated cuts into OsiCut, cs.

This is only done for rows with unsatisfied 0-1 variables. If there are many of these it will be slow. Improvements would do a randomized subset and also speed up shortest path algorithm used.

Implements [CglCutGenerator](#).



### 6.28.3 Friends And Related Function Documentation

6.28.3.1 `void CglOddHoleUnitTest ( const OsiSolverInterface * siP, const std::string mpdDir )` [*friend*]

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

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

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

- [CglOddHole.hpp](#)

## 6.29 CglParam Class Reference

Class collecting parameters for all cut generators.

```
#include <CglParam.hpp>
```

Inheritance diagram for CglParam:

### Public Member Functions

#### Public Set/get methods

- virtual void [setINFINIT](#) (const double inf)  
*Set INFINIT.*
- double [getINFINIT](#) () const  
*Get value of INFINIT.*
- virtual void [setEPS](#) (const double eps)  
*Set EPS.*
- double [getEPS](#) () const  
*Get value of EPS.*
- virtual void [setEPS\\_COEFF](#) (const double eps\_c)  
*Set EPS\_COEFF.*
- double [getEPS\\_COEFF](#) () const  
*Get value of EPS\_COEFF.*
- virtual void [setMAX\\_SUPPORT](#) (const int max\_s)  
*Set MAX\_SUPPORT.*
- int [getMAX\\_SUPPORT](#) () const  
*Get value of MAX\_SUPPORT.*

#### Constructors and destructors

- [CglParam](#) (const double inf=COIN\_DBL\_MAX, const double eps=1e-6, const double eps\_c=1e-5, const int max\_s=COIN\_INT\_MAX)  
*Default constructor.*
- [CglParam](#) (const [CglParam](#) &)  
*Copy constructor.*
- virtual [CglParam](#) \* [clone](#) () const  
*Clone.*
- [CglParam](#) & [operator=](#) (const [CglParam](#) &rhs)  
*Assignment operator.*
- virtual [~CglParam](#) ()  
*Destructor.*

## Protected Attributes

### Protected member data

- double **INFINIT**
- double **EPS**
- double **EPS\_COEFF**
- int **MAX\_SUPPORT**

*Maximum number of non zero coefficients in a generated cut; Default: COIN\_INT\_MAX.*

## 6.29.1 Detailed Description

Class collecting parameters for all cut generators.

Each generator may have a derived class to add parameters. Each generator might also set different default values for the parameters in [CglParam](#).

Definition at line 22 of file CglParam.hpp.

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

- CglParam.hpp

## 6.30 CglPreProcess Class Reference

Class for preProcessing and postProcessing.

```
#include <CglPreProcess.hpp>
```

## Public Member Functions

### Main methods

- OsiSolverInterface \* [preProcess](#) (OsiSolverInterface &model, bool makeEquality=false, int numberPasses=5)  
*preProcess problem - returning new problem.*
- OsiSolverInterface \* [preProcessNonDefault](#) (OsiSolverInterface &model, int makeEquality=0, int number↵ Passes=5, int tuning=0)  
*preProcess problem - returning new problem.*
- void [postProcess](#) (OsiSolverInterface &model, bool deleteStuff=true)  
*Creates solution in original model.*
- int [tightenPrimalBounds](#) (OsiSolverInterface &model, double factor=0.0)  
*Tightens primal bounds to make dual and branch and cutfaster.*
- OsiSolverInterface \* [someFixed](#) (OsiSolverInterface &model, double fractionToKeep=0.25, bool fix↵ ContinuousAsWell=false, char \*keep=NULL) const  
*Fix some of problem - returning new problem.*
- OsiSolverInterface \* [cliquelt](#) (OsiSolverInterface &model, double cliquesNeeded=0.0) const  
*Replace cliques by more maximal cliques Returns NULL if rows not reduced by greater than cliquesNeeded\*rows.*
- int [reducedCostFix](#) (OsiSolverInterface &model)  
*If we have a cutoff - fix variables.*

### 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.*

- 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.*
- OsiSolverInterface \* [originalModel](#) () const  
*The original solver associated with this model.*
- OsiSolverInterface \* [startModel](#) () const  
*Solver after making clique equalities (may == original)*
- OsiSolverInterface \* [modelAtPass](#) (int iPass) const  
*Copies of solver at various stages after presolve.*
- OsiSolverInterface \* [modifiedModel](#) (int iPass) const  
*Copies of solver at various stages after presolve after modifications.*
- OsiPresolve \* [presolve](#) (int iPass) const  
*Matching presolve information.*
- const int \* [originalColumns](#) ()  
*Return a pointer to the original columns (with possible clique slacks) MUST be called before postProcess otherwise you just get 0,1,2.*
- const int \* [originalRows](#) ()  
*Return a pointer to the original rows MUST be called before postProcess otherwise you just get 0,1,2.*
- int [numberSOS](#) () const  
*Number of SOS if found.*
- const int \* [typeSOS](#) () const  
*Type of each SOS.*
- const int \* [startSOS](#) () const  
*Start of each SOS.*
- const int \* [whichSOS](#) () const  
*Columns in SOS.*
- const double \* [weightSOS](#) () const  
*Weights for each SOS column.*
- void [passInProhibited](#) (const char \*[prohibited](#), int numberColumns)  
*Pass in prohibited columns.*
- const char \* [prohibited](#) ()  
*Updated prohibited columns.*
- int [numberIterationsPre](#) () const  
*Number of iterations PreProcessing.*
- int [numberIterationsPost](#) () const  
*Number of iterations PostProcessing.*
- void [passInRowTypes](#) (const char \*[rowTypes](#), int numberOfRows)  
*Pass in row types 0 normal 1 cut rows - will be dropped if remain in At end of preprocess cut rows will be dropped and put into cuts.*
- const char \* [rowTypes](#) ()  
*Updated row types - may be NULL Carried around and corresponds to existing rows -1 added by preprocess e.g.*
- const [CglStored](#) & [cuts](#) () const  
*Return cuts from dropped rows.*
- const [CglStored](#) \* [cutsPointer](#) () const  
*Return pointer to cuts from dropped rows.*
- void [update](#) (const OsiPresolve \*pinfo, const OsiSolverInterface \*solver)  
*Update prohibited and rowType.*
- void [setOptions](#) (int value)  
*Set options.*

### Cut generator methods

- int `numberCutGenerators` () const  
*Get the number of cut generators.*
- `CglCutGenerator` \*\* `cutGenerators` () const  
*Get the list of cut generators.*
- `CglCutGenerator` \* `cutGenerator` (int i) const  
*Get the specified cut generator.*
- void `addCutGenerator` (`CglCutGenerator` \*generator)  
*Add one generator - up to user to delete generators.*

#### Setting/Accessing application data

- void `setApplicationData` (void \*appData)  
*Set application data.*
- void \* `getApplicationData` () const  
*Get application data.*

#### Message handling

- 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.*

#### Constructors and destructors etc

- `CglPreProcess` ()  
*Constructor.*
- `CglPreProcess` (const `CglPreProcess` &rhs)  
*Copy constructor.*
- `CglPreProcess` & `operator=` (const `CglPreProcess` &rhs)  
*Assignment operator.*
- `~CglPreProcess` ()  
*Destructor.*
- void `gutsOfDestructor` ()  
*Clears out as much as possible.*

### 6.30.1 Detailed Description

Class for preProcessing and postProcessing.

While cuts can be added at any time in the tree, some cuts are actually just stronger versions of existing constraints. In this case they can replace those constraints rather than being added as new constraints. This is awkward in the tree but reasonable at the root node.

This is a general process class which uses other cut generators to strengthen constraints, establish that constraints are redundant, fix variables and find relationships such as  $x + y == 1$ .

Presolve will also be done.

If row names existed they may be replaced by R0000000 etc

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

### 6.30.2 Member Function Documentation

**6.30.2.1** `OsiSolverInterface* CglPreProcess::preProcess ( OsiSolverInterface & model, bool makeEquality = false, int numberPasses = 5 )`

preProcess problem - returning new problem.

If makeEquality true then  $\leq$  cliques converted to  $=$ . Presolve will be done numberPasses times.

Returns NULL if infeasible

This version uses default strategy. For more control copy and edit code from this function i.e. call preProcessNonDefault

**6.30.2.2** `OsiSolverInterface* CglPreProcess::preProcessNonDefault ( OsiSolverInterface & model, int makeEquality = 0, int numberPasses = 5, int tuning = 0 )`

preProcess problem - returning new problem.

If makeEquality true then  $\leq$  cliques converted to  $=$ . Presolve will be done numberPasses times.

Returns NULL if infeasible

This version assumes user has added cut generators to [CglPreProcess](#) object before calling it. As an example use coding in preProcess If makeEquality is 1 add slacks to get cliques, if 2 add slacks to get sos (but only if looks plausible) and keep sos info

**6.30.2.3** `int CglPreProcess::tightenPrimalBounds ( OsiSolverInterface & model, double factor = 0.0 )`

Tightens primal bounds to make dual and branch and cut faster.

Unless fixed or integral, bounds are slightly looser than they could be. Returns non-zero if problem infeasible Fudge for branch and bound - put bounds on columns of factor \* largest value (at continuous) - should improve stability in branch and bound on infeasible branches (0.0 is off)

**6.30.2.4** `OsiSolverInterface* CglPreProcess::someFixed ( OsiSolverInterface & model, double fractionToKeep = 0.25, bool fixContinuousAsWell = false, char * keep = NULL ) const`

Fix some of problem - returning new problem.

Uses reduced costs. Optional signed character array 1 always keep, -1 always discard, 0 use djs

**6.30.2.5** `OsiSolverInterface* CglPreProcess::cliquelt ( OsiSolverInterface & model, double cliquesNeeded = 0.0 ) const`

Replace cliques by more maximal cliques Returns NULL if rows not reduced by greater than cliquesNeeded\*rows.

**6.30.2.6** `void CglPreProcess::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.

### 6.30.2.7 `const int* CglPreProcess::originalColumns ( )`

Return a pointer to the original columns (with possible clique slacks) MUST be called before postProcess otherwise you just get 0,1,2.

### 6.30.2.8 `const int* CglPreProcess::originalRows ( )`

Return a pointer to the original rows MUST be called before postProcess otherwise you just get 0,1,2.

### 6.30.2.9 `const char* CglPreProcess::rowTypes ( ) [inline]`

Updated row types - may be NULL Carried around and corresponds to existing rows -1 added by preprocess e.g.

x+y=1 0 normal 1 cut rows - can be dropped if wanted

Definition at line 178 of file CglPreProcess.hpp.

### 6.30.2.10 `void CglPreProcess::setApplicationData ( void * appData )`

Set application data.

This is a pointer that the application can store into and retrieve. This field is available for the application to optionally define and use.

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

- CglPreProcess.hpp

## 6.31 CglProbing Class Reference

Probing Cut Generator Class.

```
#include <CglProbing.hpp>
```

Inheritance diagram for CglProbing:

Collaboration diagram for CglProbing:

### Classes

- struct [CliqueType](#)  
*Clique type.*

### Public Member Functions

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate probing/disaggregation cuts for the model of the solver interface, si.*
- int **generateCutsAndModify** (const OsiSolverInterface &si, OsiCuts &cs, [CglTreeInfo](#) \*info)

**snapshot etc**

- int [snapshot](#) (const OsiSolverInterface &si, char \*possible=NULL, bool withObjective=true)  
*Create a copy of matrix which is to be used this is to speed up process and to give global cuts Can give an array with 1 set to select, 0 to ignore column bounds are tightened If array given then values of 1 will be set to 0 if redundant.*
- void [deleteSnapshot](#) ()  
*Deletes snapshot.*
- int [createCliques](#) (OsiSolverInterface &si, int minimumSize=2, int maximumSize=100)  
*Creates cliques for use by probing.*
- void [deleteCliques](#) ()  
*Delete all clique information.*
- OsiSolverInterface \* [cliqueModel](#) (const OsiSolverInterface \*model, int type)  
*Create a fake model by adding cliques if type&4 then delete rest of model first, if 1 then add proper cliques, 2 add fake cliques.*

#### Get tighter column bounds

- const double \* [tightLower](#) () const  
*Lower.*
- const double \* [tightUpper](#) () const  
*Upper.*
- const char \* [tightenBounds](#) () const  
*Array which says tighten continuous.*

#### Get possible freed up row bounds - only valid after mode==3

- const double \* [relaxedRowLower](#) () const  
*Lower.*
- const double \* [relaxedRowUpper](#) () const  
*Upper.*

#### Change mode

- void [setMode](#) (int mode)  
*Set.*
- int [getMode](#) () const  
*Get.*

#### Change maxima

- void [setMaxPass](#) (int value)  
*Set maximum number of passes per node.*
- int [getMaxPass](#) () const  
*Get maximum number of passes per node.*
- void [setLogLevel](#) (int value)  
*Set log level - 0 none, 1 - a bit, 2 - more details.*
- int [getLogLevel](#) () const  
*Get log level.*
- void [setMaxProbe](#) (int value)  
*Set maximum number of unsatisfied variables to look at.*
- int [getMaxProbe](#) () const  
*Get maximum number of unsatisfied variables to look at.*
- void [setMaxLook](#) (int value)  
*Set maximum number of variables to look at in one probe.*
- int [getMaxLook](#) () const

- *Get maximum number of variables to look at in one probe.*  
 • void [setMaxElements](#) (int value)
- *Set maximum number of elements in row for it to be considered.*  
 • int [getMaxElements](#) () const
- *Get maximum number of elements in row for it to be considered.*  
 • void [setMaxPassRoot](#) (int value)
- *Set maximum number of passes per node (root node)*  
 • int [getMaxPassRoot](#) () const
- *Get maximum number of passes per node (root node)*  
 • void [setMaxProbeRoot](#) (int value)
- *Set maximum number of unsatisfied variables to look at (root node)*  
 • int [getMaxProbeRoot](#) () const
- *Get maximum number of unsatisfied variables to look at (root node)*  
 • void [setMaxLookRoot](#) (int value)
- *Set maximum number of variables to look at in one probe (root node)*  
 • int [getMaxLookRoot](#) () const
- *Get maximum number of variables to look at in one probe (root node)*  
 • void [setMaxElementsRoot](#) (int value)
- *Set maximum number of elements in row for it to be considered (root node)*  
 • int [getMaxElementsRoot](#) () const
- *Get maximum number of elements in row for it to be considered (root node)*  
 • virtual bool [mayGenerateRowCutsInTree](#) () const
- *Returns true if may generate Row cuts in tree (rather than root node).*

#### Get information back from probing

- int [numberThisTime](#) () const  
*Number looked at this time.*
- const int \* [lookedAt](#) () const  
*Which ones looked at this time.*

#### Stop or restart row cuts (otherwise just fixing from probing)

- void [setRowCuts](#) (int type)  
*Set 0 no cuts, 1 just disaggregation type, 2 coefficient ( 3 both)*
- int [rowCuts](#) () const  
*Get.*

#### Information on cliques

- int [numberCliques](#) () const  
*Number of cliques.*
- [CliqueType](#) \* [cliqueType](#) () const  
*Clique type.*
- int \* [cliqueStart](#) () const  
*Start of each clique.*
- [CliqueEntry](#) \* [cliqueEntry](#) () const  
*Entries for clique.*

#### Whether use objective as constraint

- void [setUsingObjective](#) (int yesNo)



- *Set 0 don't 1 do -1 don't even think about it.*
- `int getUsingObjective () const`  
*Get.*

#### Mark which continuous variables are to be tightened

- `void tightenThese (const OsiSolverInterface &solver, int number, const int *which)`  
*Mark variables to be tightened.*

#### Constructors and destructors

- `CglProbing ()`  
*Default constructor.*
- `CglProbing (const CglProbing &)`  
*Copy constructor.*
- `virtual CglCutGenerator * clone () const`  
*Clone.*
- `CglProbing & operator= (const CglProbing &rhs)`  
*Assignment operator.*
- `virtual ~CglProbing ()`  
*Destructor.*
- `virtual void refreshSolver (OsiSolverInterface *solver)`  
*This can be used to refresh any information.*
- `virtual std::string generateCpp (FILE *fp)`  
*Create C++ lines to get to current state.*

#### Friends

- `void CglProbingUnitTest (const OsiSolverInterface *siP, const std::string mpdDir)`  
*A function that tests the methods in the [CglProbing](#) class.*

#### Additional Inherited Members

##### 6.31.1 Detailed Description

Probing Cut Generator Class.

Definition at line 25 of file `CglProbing.hpp`.

##### 6.31.2 Member Function Documentation

**6.31.2.1** `virtual void CglProbing::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo () ) [virtual]`

Generate probing/disaggregation cuts for the model of the solver interface, *si*.

This is a simplification of probing ideas put into OSL about ten years ago. The only known documentation is a copy of a talk handout - we think Robin Lougee-Heimer has a copy!

For selected integer variables (e.g. unsatisfied ones) the effect of setting them up or down is investigated. Setting a variable up may in turn set other variables (continuous as well as integer). There are various possible results:

1) It is shown that problem is infeasible (this may also be because objective function or reduced costs show worse than best solution). If the other way is feasible we can generate a column cut (and continue probing), if not feasible we can say problem infeasible.

2) If both ways are feasible, it can happen that  $x$  to 0 implies  $y$  to 1 and  $x$  to 1 implies  $y$  to 1 (again a column cut). More common is that  $x$  to 0 implies  $y$  to 1 and  $x$  to 1 implies  $y$  to 0 so we could substitute for  $y$  which might lead later to more powerful cuts. This is not done in this code as there is no mechanism for returning information.

3) When  $x$  to 1 a constraint went slack by  $c$ . We can tighten the constraint  $ax + \dots \leq b$  (where  $a$  may be zero) to  $(a+c)x + \dots \leq b$ . If this cut is violated then it is generated.

4) Similarly we can generate implied disaggregation cuts

Note - differences to cuts in OSL.

a) OSL had structures intended to make this faster. b) The "chaining" in 2) was done c) Row cuts modified original constraint rather than adding cut b) This code can cope with general integer variables.

Insert the generated cuts into `OsiCut`, `cs`.

If a "snapshot" of a matrix exists then this will be used. Presumably this will give global cuts and will be faster. No check is done to see if cuts will be global.

Otherwise use current matrix.

Both row cuts and column cuts may be returned

The mode options are: 0) Only unsatisfied integer variables will be looked at. If no information exists for that variable then probing will be done so as a by-product you "may" get a fixing or infeasibility. This will be fast and is only available if a snapshot exists (otherwise as 1). The bounds in the snapshot are the ones used. 1) Look at unsatisfied integer variables, using current bounds. Probing will be done on all looked at. 2) Look at all integer variables, using current bounds. Probing will be done on all

If `generateCutsAndModify` is used then new relaxed row bounds and tightened column bounds are generated Returns number of infeasibilities

Implements [CglCutGenerator](#).

**6.31.2.2** `int CglProbing::snapshot ( const OsiSolverInterface & si, char * possible = NULL, bool withObjective = true )`

Create a copy of matrix which is to be used this is to speed up process and to give global cuts Can give an array with 1 set to select, 0 to ignore column bounds are tightened If array given then values of 1 will be set to 0 if redundant.

Objective may be added as constraint Returns 1 if infeasible otherwise 0

**6.31.2.3** `int CglProbing::createCliques ( OsiSolverInterface & si, int minimumSize = 2, int maximumSize = 100 )`

Creates cliques for use by probing.

Only cliques  $\geq$  `minimumSize` and  $<$  `maximumSize` created Can also try and extend cliques as a result of probing (root node). Returns number of cliques found.

**6.31.2.4** `virtual bool CglProbing::mayGenerateRowCutsInTree ( ) const [virtual]`

Returns true if may generate Row cuts in tree (rather than root node).

Used so know if matrix will change in tree. Really meant so column cut generators can still be active without worrying code. Default is true

Reimplemented from [CglCutGenerator](#).

### 6.31.3 Friends And Related Function Documentation

6.31.3.1 void CglProbingUnitTest ( const OsiSolverInterface \* *siP*, const std::string *mpdDir* ) [friend]

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

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

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

- CglProbing.hpp

## 6.32 CglRedSplit Class Reference

Gomory Reduce-and-Split Cut Generator Class; See method [generateCuts\(\)](#).

```
#include <CglRedSplit.hpp>
```

Inheritance diagram for CglRedSplit:

Collaboration diagram for CglRedSplit:

### Public Member Functions

#### generateCuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate Reduce-and-Split Mixed Integer Gomory cuts for the model of the solver interface si.*
- virtual bool [needsOptimalBasis](#) () const  
*Return true if needs optimal basis to do cuts (will return true)*

#### Public Methods

- void **setParam** (const [CglRedSplitParam](#) &source)
- [CglRedSplitParam](#) **getParam** () const
- void **compute\_is\_lub** ()
- void **compute\_is\_integer** ()
- void [set\\_given\\_optsol](#) (const double \*given\_sol, const int card\_sol)  
*Set given\_optsol to the given optimal solution given\_sol.*
- void [print](#) () const  
*Print some of the data members.*
- void [printOptTab](#) (OsiSolverInterface \*solver) const  
*Print the current simplex tableau.*

#### Public Methods (soon to be obsolete)

- void [setLimit](#) (int limit)  
*Set limit, the maximum number of non zero coefficients in generated cut; Default: 50.*
- int [getLimit](#) () const  
*Get value of limit.*
- void [setAway](#) (double value)

Set away, the minimum distance from being integer used for selecting rows for cut generation; all rows whose pivot variable should be integer but is more than away from integrality will be selected; Default: 0.05.

- double [getAway](#) () const  
Get value of away.
- void [setLUB](#) (double value)  
Set the value of LUB, value considered large for the absolute value of a lower or upper bound on a variable; Default: 1000.
- double [getLUB](#) () const  
Get the value of LUB.
- void [setEPS](#) (double value)  
Set the value of EPS, epsilon for double computations; Default: 1e-7.
- double [getEPS](#) () const  
Get the value of EPS.
- void [setEPS\\_COEFF](#) (double value)  
Set the value of EPS\_COEFF, epsilon for values of coefficients; Default: 1e-8.
- double [getEPS\\_COEFF](#) () const  
Get the value of EPS\_COEFF.
- void [setEPS\\_COEFF\\_LUB](#) (double value)  
Set the value of EPS\_COEFF\_LUB, epsilon for values of coefficients for variables with absolute value of lower or upper bound larger than LUB; Default: 1e-13.
- double [getEPS\\_COEFF\\_LUB](#) () const  
Get the value of EPS\_COEFF\_LUB.
- void [setEPS\\_RELAX](#) (double value)  
Set the value of EPS\_RELAX, value used for relaxing the right hand side of each generated cut; Default: 1e-8.
- double [getEPS\\_RELAX](#) () const  
Get the value of EPS\_RELAX.
- void [setNormIsZero](#) (double value)  
Set the value of normIsZero, the threshold for considering a norm to be 0; Default: 1e-5.
- double [getNormIsZero](#) () const  
Get the value of normIsZero.
- void [setMinReduc](#) (double value)  
Set the value of minReduc, threshold for relative norm improvement for performing a reduction; Default: 0.05.
- double [getMinReduc](#) () const  
Get the value of minReduc.
- void [setMaxTab](#) (double value)  
Set the maximum allowed value for  $(mTab * mTab * CoinMax(mTab, nTab))$  where  $mTab$  is the number of rows used in the combinations and  $nTab$  is the number of continuous non basic variables.
- double [getMaxTab](#) () const  
Get the value of maxTab.

### Constructors and destructors

- [CglRedSplit](#) ()  
Default constructor.
- [CglRedSplit](#) (const [CglRedSplitParam](#) &RS\_param)  
Constructor with specified parameters.
- [CglRedSplit](#) (const [CglRedSplit](#) &)  
Copy constructor.
- virtual [CglCutGenerator](#) \* [clone](#) () const  
Clone.
- [CglRedSplit](#) & [operator=](#) (const [CglRedSplit](#) &rhs)  
Assignment operator.
- virtual [~CglRedSplit](#) ()  
Destructor.
- virtual std::string [generateCpp](#) (FILE \*fp)  
Create C++ lines to get to current state.

## Friends

- void [CglRedSplitUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)

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

## Additional Inherited Members

### 6.32.1 Detailed Description

Gomory Reduce-and-Split Cut Generator Class; See method [generateCuts\(\)](#).

Based on the paper by K. Anderson, G. Cornuejols, Yanjun Li, "Reduce-and-Split Cuts: Improving the Performance of Mixed Integer Gomory Cuts", Management Science 51 (2005).

Definition at line 26 of file CglRedSplit.hpp.

### 6.32.2 Member Function Documentation

**6.32.2.1** virtual void [CglRedSplit::generateCuts](#) ( const OsiSolverInterface & *si*, OsiCuts & *cs*, const [CglTreeInfo](#) *info* = [CglTreeInfo\(\)](#) ) [virtual]

Generate Reduce-and-Split Mixed Integer Gomory cuts for the model of the solver interface si.

Insert the generated cuts into OsiCuts cs.

Warning: This generator currently works only with the Lp solvers Clp or Cplex9.0 or higher. It requires access to the optimal tableau and optimal basis inverse and makes assumptions on the way slack variables are added by the solver. The Osi implementations for Clp and Cplex verify these assumptions.

When calling the generator, the solver interface si must contain an optimized problem and information related to the optimal basis must be available through the OsiSolverInterface methods (si->optimalBasisIsAvailable() must return 'true'). It is also essential that the integrality of structural variable i can be obtained using si->isInteger(i).

Reduce-and-Split cuts are variants of Gomory cuts: Starting from the current optimal tableau, linear combinations of the rows of the current optimal simplex tableau are used for generating Gomory cuts. The choice of the linear combinations is driven by the objective of reducing the coefficients of the non basic continuous variables in the resulting row. Note that this generator might not be able to generate cuts for some solutions violating integrality constraints.

Implements [CglCutGenerator](#).

**6.32.2.2** void [CglRedSplit::set\\_given\\_optsol](#) ( const double \* *given\_sol*, const int *card\_sol* )

Set given\_optsol to the given optimal solution given\_sol.

If given\_optsol is set using this method, the code will stop as soon as a generated cut is violated by the given solution; exclusively for debugging purposes.

**6.32.2.3** void [CglRedSplit::setMaxTab](#) ( double *value* )

Set the maximum allowed value for (mTab \* mTab \* CoinMax(mTab, nTab)) where mTab is the number of rows used in the combinations and nTab is the number of continuous non basic variables.

The work of the generator is proportional to (mTab \* mTab \* CoinMax(mTab, nTab)). Reducing the value of maxTab makes the generator faster, but weaker. Default: 1e7.

### 6.32.3 Friends And Related Function Documentation

6.32.3.1 `void CglRedSplitUnitTest ( const OsiSolverInterface * siP, const std::string mpdDir ) [friend]`

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

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

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

- [CglRedSplit.hpp](#)

## 6.33 CglRedSplit2 Class Reference

Reduce-and-Split Cut Generator Class; See method [generateCuts\(\)](#).

```
#include <CglRedSplit2.hpp>
```

Inheritance diagram for CglRedSplit2:

Collaboration diagram for CglRedSplit2:

### Public Member Functions

#### **generateCuts**

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate Reduce-and-Split Mixed Integer Gomory cuts for the model of the solver interface si.*
- virtual bool [needsOptimalBasis](#) () const  
*Return true if needs optimal basis to do cuts (will return true)*
- int **generateMultipliers** (const OsiSolverInterface &si, int \*lambda, int maxNumMultipliers, int \*basic↔ Variables=NULL, OsiCuts \*cs=NULL)
- int **tiltLandPcut** (const OsiSolverInterface \*si, double \*row, double rowRhs, int rownumber, const double \*xbar, const int \*newnonbasics, OsiRowCut \*cs, int \*lambda=NULL)

#### **Public Methods**

- void **setParam** (const [CglRedSplit2Param](#) &source)
- [CglRedSplit2Param](#) & **getParam** ()
- void [print](#) () const  
*Print some of the data members; used for debugging.*
- void [printOptTab](#) (OsiSolverInterface \*solver) const  
*Print the current simplex tableau.*

#### **Constructors and destructors**

- [CglRedSplit2](#) ()  
*Default constructor.*
- [CglRedSplit2](#) (const [CglRedSplit2Param](#) &RS\_param)  
*Constructor with specified parameters.*
- [CglRedSplit2](#) (const [CglRedSplit2](#) &)  
*Copy constructor.*

- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglRedSplit2](#) & [operator=](#) (const [CglRedSplit2](#) &rhs)  
*Assignment operator.*
- virtual [~CglRedSplit2](#) ()  
*Destructor.*

## Friends

- void [CglRedSplit2UnitTest](#) (const [OsiSolverInterface](#) \*siP, const std::string mpdDir)  
*A function that tests some of the methods in the [CglRedSplit2](#) class.*

## Additional Inherited Members

### 6.33.1 Detailed Description

Reduce-and-Split Cut Generator Class; See method [generateCuts\(\)](#).

Based on the papers "Practical strategies for generating rank-1 split cuts in mixed-integer linear programming" by G. Cornuejols and G. Nannicini, published on Mathematical Programming Computation, and "Combining Lift-and-Project and Reduce-and-Split" by E. Balas, G. Cornuejols, T. Kis and G. Nannicini, published on INFORMS Journal on Computing. Part of this code is based on [CglRedSplit](#) by F. Margot.

Definition at line 31 of file [CglRedSplit2.hpp](#).

### 6.33.2 Member Function Documentation

6.33.2.1 virtual void [CglRedSplit2::generateCuts](#) ( const [OsiSolverInterface](#) & si, [OsiCuts](#) & cs, const [CglTreeInfo](#) info = [CglTreeInfo](#) () ) [virtual]

Generate Reduce-and-Split Mixed Integer Gomory cuts for the model of the solver interface si.

Insert the generated cuts into [OsiCuts](#) cs.

This generator currently works only with the Lp solvers Clp or Cplex9.0 or higher. It requires access to the optimal tableau and optimal basis inverse and makes assumptions on the way slack variables are added by the solver. The [Osi](#) implementations for Clp and Cplex verify these assumptions.

When calling the generator, the solver interface si must contain an optimized problem and information related to the optimal basis must be available through the [OsiSolverInterface](#) methods ([si->optimalBasisIsAvailable\(\)](#) must return 'true'). It is also essential that the integrality of structural variable i can be obtained using [si->isInteger\(i\)](#).

Reduce-and-Split cuts are a class of split cuts. We compute linear combinations of the rows of the simplex tableau, trying to reduce some of the coefficients on the nonbasic continuous columns. We have a large number of heuristics to choose which coefficients should be reduced, and by using which rows. The paper explains everything in detail.

Note that this generator can potentially generate a huge number of cuts, depending on how it is parametered. Default parameters should be good for most situations; if you want to go heavy on split cuts, use more row selection strategies or a different number of rows in the linear combinations. Again, look at the paper for details. If you want to generate a small number of cuts, default parameters are not the best choice.

A combination of Reduce-and-Split with Lift & Project is described in the paper "Combining Lift-and-Project and Reduce-and-Split". The Reduce-and-Split code for the implementation used in that paper is included here.

This generator does not generate the same cuts as [CglRedSplit](#), therefore both generators can be used in conjunction.

Implements [CglCutGenerator](#).

### 6.33.3 Friends And Related Function Documentation

6.33.3.1 `void CglRedSplit2UnitTest ( const OsiSolverInterface * siP, const std::string mpdDir ) [friend]`

A function that tests some of the methods in the [CglRedSplit2](#) class.

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

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

- [CglRedSplit2.hpp](#)

## 6.34 CglRedSplit2Param Class Reference

Class collecting parameters the Reduced-and-split cut generator.

```
#include <CglRedSplit2Param.hpp>
```

Inheritance diagram for CglRedSplit2Param:

Collaboration diagram for CglRedSplit2Param:

### Public Types

- enum [RowSelectionStrategy](#)  
*Enumerations for parameters.*
- enum [ColumnSelectionStrategy](#)  
*Column selection strategies; again, look them up in the paper.*
- enum [ColumnScalingStrategy](#)  
*Scaling strategies for new nonbasic columns for Lift & Project; "factor" is the value of columnScalingBoundLAP\_.*

### Public Member Functions

#### Set/get methods

- virtual void [setAway](#) (double value)  
*Set away, the minimum distance from being integer used for selecting rows for cut generation; all rows whose pivot variable should be integer but is more than away from integrality will be selected; Default: 0.005.*
- double [getAway](#) () const  
*Get value of away.*
- void [setEPS\\_ELIM](#) (double value)  
*Set the value of EPS\_ELIM, epsilon for values of coefficients when eliminating slack variables; Default: 0.0.*
- double [getEPS\\_ELIM](#) () const  
*Get the value of EPS\_ELIM.*
- virtual void [setEPS\\_RELAX\\_ABS](#) (double eps\_ra)  
*Set EPS\_RELAX\_ABS.*
- double [getEPS\\_RELAX\\_ABS](#) () const  
*Get value of EPS\_RELAX\_ABS.*
- virtual void [setEPS\\_RELAX\\_REL](#) (double eps\_rr)  
*Set EPS\_RELAX\_REL.*



- double [getEPS\\_RELAX\\_REL](#) () const  
*Get value of EPS\_RELAX\_REL.*
- virtual void **setMAXDYN** (double value)
- double [getMAXDYN](#) () const  
*Get the value of MAXDYN.*
- virtual void [setMINVIOL](#) (double value)  
*Set the value of MINVIOL, the minimum violation for the current basic solution in a generated cut.*
- double [getMINVIOL](#) () const  
*Get the value of MINVIOL.*
- void [setMAX\\_SUPP\\_ABS](#) (int value)  
*Maximum absolute support of the cutting planes.*
- int **getMAX\_SUPP\_ABS** () const
- void [setMAX\\_SUPP\\_REL](#) (double value)  
*Maximum relative support of the cutting planes.*
- double **getMAX\_SUPP\_REL** () const
- virtual void [setUSE\\_INTSLACKS](#) (int value)  
*Set the value of USE\_INTSLACKS.*
- int [getUSE\\_INTSLACKS](#) () const  
*Get the value of USE\_INTSLACKS.*
- virtual void [setNormIsZero](#) (double value)  
*Set the value of normIsZero, the threshold for considering a norm to be 0; Default: 1e-5.*
- double [getNormIsZero](#) () const  
*Get the value of normIsZero.*
- virtual void [setMinNormReduction](#) (double value)  
*Set the value of minNormReduction; Default: 0.1.*
- double [getMinNormReduction](#) () const  
*Get the value of normIsZero.*
- virtual void [setMaxSumMultipliers](#) (int value)  
*Set the value of maxSumMultipliers; Default: 10.*
- int [getMaxSumMultipliers](#) () const  
*Get the value of maxSumMultipliers.*
- virtual void [setNormalization](#) (double value)  
*Set the value of normalization; Default: 0.0001.*
- double [getNormalization](#) () const  
*Get the value of normalization.*
- virtual void [addNumRowsReduction](#) (int value)  
*Set the value of numRowsReduction, max number of rows that are used for each row reduction step.*
- std::vector< int > [getNumRowsReduction](#) () const  
*get the value*
- void [resetNumRowsReduction](#) ()  
*reset*
- virtual void [addColumnSelectionStrategy](#) ([ColumnSelectionStrategy](#) value)  
*Add the value of columnSelectionStrategy.*
- std::vector< [ColumnSelectionStrategy](#) > [getColumnSelectionStrategy](#) () const  
*get the value*
- void [resetColumnSelectionStrategy](#) ()  
*reset*
- virtual void [addRowSelectionStrategy](#) ([RowSelectionStrategy](#) value)  
*Set the value for rowSelectionStrategy, which changes the way we choose the rows for the reduction step.*
- std::vector< [RowSelectionStrategy](#) > [getRowSelectionStrategy](#) () const  
*get the value*
- void [resetRowSelectionStrategy](#) ()

- reset*
- virtual void [addNumRowsReductionLAP](#) (int value)  
*Set the value of numRowsReductionLAP, max number of rows that are used for each row reduction step during Lift & Project.*
- std::vector< int > [getNumRowsReductionLAP](#) () const  
*get the value*
- void [resetNumRowsReductionLAP](#) ()  
*reset*
- virtual void [addColumnSelectionStrategyLAP](#) (ColumnSelectionStrategy value)  
*Add the value of columnSelectionStrategyLAP.*
- std::vector< ColumnSelectionStrategy > [getColumnSelectionStrategyLAP](#) () const  
*get the value*
- void [resetColumnSelectionStrategyLAP](#) ()  
*reset*
- virtual void [addRowSelectionStrategyLAP](#) (RowSelectionStrategy value)  
*Set the value for rowSelectionStrategyLAP, which changes the way we choose the rows for the reduction step.*
- std::vector< RowSelectionStrategy > [getRowSelectionStrategyLAP](#) () const  
*get the value*
- void [resetRowSelectionStrategyLAP](#) ()  
*reset*
- virtual void [setColumnScalingStrategyLAP](#) (ColumnScalingStrategy value)  
*Set the value for columnScalingStrategyLAP, which sets the way nonbasic columns that are basic in the fractional point to cut off are scaled.*
- [ColumnScalingStrategy](#) [getColumnScalingStrategyLAP](#) () const  
*get the value*
- virtual void [setColumnScalingBoundLAP](#) (double value)  
*Set the value for the bound in the column scaling factor.*
- double [getColumnScalingBoundLAP](#) () const  
*get the value*
- virtual void [setTimeLimit](#) (double value)  
*Set the value of the time limit for cut generation (in seconds)*
- double [getTimeLimit](#) () const  
*get the value*
- virtual void [setMaxNumCuts](#) (int value)  
*Set the value for the maximum number of cuts that can be returned.*
- int [getMaxNumCuts](#) () const  
*get the value*
- virtual void [setMaxNumComputedCuts](#) (int value)  
*Set the value for the maximum number of cuts that can be computed.*
- int [getMaxNumComputedCuts](#) () const  
*get the value*
- virtual void [setMaxNonzeroesTab](#) (int value)  
*Set the value for the maximum number of nonzeroes in a row of the simplex tableau for the row to be considered.*
- int [getMaxNonzeroesTab](#) () const  
*get the value*
- virtual void [setSkipGomory](#) (int value)  
*Set the value of skipGomory: should we skip simple Gomory cuts, i.e.*
- int [getSkipGomory](#) () const  
*get the value*

## Constructors and destructors

- [CglRedSplit2Param](#) (bool use\_default\_strategies=true, double eps=1e-12, double eps\_coeff=1e-11, double eps\_elim=0.0, double eps\_relax\_abs=1e-11, double eps\_relax\_rel=1e-13, double max\_dyn=1e6, double min\_viol=1e-3, int max\_supp\_abs=1000, double max\_supp\_rel=0.1, int use\_int\_slacks=0, double norm\_zero=1e-5, double minNormReduction=0.1, int maxSumMultipliers=10, double normalization=0.0001, double away=0.005, double timeLimit=60, int maxNumCuts=10000, int maxNumComputedCuts=10000, int maxNonzeroesTab=1000, double columnScalingBoundLAP=5.0, int skipGomory=1)  
*Default constructor.*
- [CglRedSplit2Param](#) (const [CglParam](#) &source, bool use\_default\_strategies=true, double eps\_elim=0.0, double eps\_relax\_abs=1e-11, double eps\_relax\_rel=1e-13, double max\_dyn=1e6, double min\_viol=1e-3, double max\_supp\_rel=0.1, int use\_int\_slacks=0, double norm\_zero=1e-5, double minNormReduction=0.1, int maxSumMultipliers=10, double normalization=0.0001, double away=0.005, double timeLimit=60, int maxNumCuts=10000, int maxNumComputedCuts=10000, int maxNonzeroesTab=1000, double columnScalingBoundLAP=5.0, int skipGomory=1)  
*Constructor from [CglParam](#).*
- [CglRedSplit2Param](#) (const [CglRedSplit2Param](#) &source)  
*Copy constructor.*
- virtual [CglRedSplit2Param](#) \* clone () const  
*Clone.*
- virtual [CglRedSplit2Param](#) & operator= (const [CglRedSplit2Param](#) &rhs)  
*Assignment operator.*
- virtual ~[CglRedSplit2Param](#) ()  
*Destructor.*

## Protected Attributes

### Parameters

- double [EPS\\_ELIM](#)  
*Epsilon for value of coefficients when eliminating slack variables.*
- double [EPS\\_RELAX\\_ABS](#)  
*Value added to the right hand side of each generated cut to relax it.*
- double [EPS\\_RELAX\\_REL](#)  
*For a generated cut with right hand side rhs\_val, EPS\_RELAX\_EPS \* fabs(rhs\_val) is used to relax the constraint.*
- double [MAXDYN](#)
- double [MINVIOL](#)  
*Minimum violation for the current basic solution in a generated cut.*
- double [MAX\\_SUPP\\_REL](#)  
*Maximum support - relative part of the formula.*
- int [USE\\_INTSLACKS](#)  
*Use integer slacks to generate cuts if USE\_INTSLACKS = 1. Default: 0.*
- double [normIsZero\\_](#)  
*Norm of a vector is considered zero if smaller than normIsZero; Default: 1e-5.*
- double [minNormReduction\\_](#)  
*Minimum reduction to accept a new row.*
- int [maxSumMultipliers\\_](#)  
*Maximum sum of the vector of row multipliers to generate a cut.*
- double [normalization\\_](#)  
*Normalization factor for the norm of lambda in the quadratic minimization problem that is solved during the coefficient reduction step.*
- double [away\\_](#)  
*Use row only if pivot variable should be integer but is more than away\_ from being integer.*
- std::vector< int > [numRowsReduction\\_](#)  
*Maximum number of rows to use for the reduction of a given row.*

- `std::vector< ColumnSelectionStrategy > columnSelectionStrategy_`  
*Column selection method.*
- `std::vector< RowSelectionStrategy > rowSelectionStrategy_`  
*Row selection method.*
- `std::vector< int > numRowsReductionLAP_`  
*Maximum number of rows to use for the reduction during Lift & Project.*
- `std::vector< ColumnSelectionStrategy > columnSelectionStrategyLAP_`  
*Column selection method for Lift & Project.*
- `std::vector< RowSelectionStrategy > rowSelectionStrategyLAP_`  
*Row selection method for Lift & Project.*
- `ColumnScalingStrategy columnScalingStrategyLAP_`  
*Column scaling strategy for the nonbasics columns that were basic in the point that we want to cut off (Lift & Project only)*
- `double columnScalingBoundLAP_`  
*Minimum value for column scaling (Lift & Project only)*
- `double timeLimit_`  
*Time limit.*
- `int maxNumCuts_`  
*Maximum number of returned cuts.*
- `int maxNumComputedCuts_`  
*Maximum number of computed cuts.*
- `int maxNonzeroesTab_`  
*Maximum number of nonzeros in tableau row for reduction.*
- `int skipGomory_`  
*Skip simple Gomory cuts.*

### 6.34.1 Detailed Description

Class collecting parameters the Reduced-and-split cut generator.

An important thing to note is that the cut generator allows for the selection of a number of strategies that can be combined together. By default, a selection that typically yields a good compromise between speed and cut strenght is made. The selection can be changed by resetting the default choices (see the functions whose name starts with "reset") or by setting the parameter `use_default_strategies` to false in the constructors. After this, the chosen strategies can be added to the list by using the functions whose name starts with "add". All strategies will be combined together: if we choose 3 row selection strategies, 2 column selection strategies, and 2 possible numbers of rows, we end up with a total of  $3*2*2$  combinations.

For a detailed explanation of the parameters and their meaning, see the paper by Cornuejols and Nannicini: "Practical strategies for generating rank-1 split cuts in mixed-integer linear programming", on Mathematical Programming Computation.

Parameters of the generator are listed below.

- MAXDYN: Maximum ratio between largest and smallest non zero coefficients in a cut. See method `setMaxDYN()`.
- EPS\_ELIM: Precision for deciding if a coefficient is zero when eliminating slack variables. See method `setEPS←_ELIM()`.
- MINVIOL: Minimum violation for the current basic solution in a generated cut. See method `setMINVIOL()`.
- EPS\_RELAX\_ABS: Absolute relaxation of cut rhs.
- EPS\_RELAX\_REL: Relative relaxation of cut rhs.

- MAX\_SUPP\_ABS: Maximum cut support (absolute).
- MAX\_SUPP\_REL: Maximum cut support (relative): the formula to compute maximum cut support is  $\text{MAX\_SUPP\_ABS} + \text{ncol} * \text{MAX\_SUPP\_REL}$ .
- USE\_INTSLACKS: Use integer slacks to generate cuts. (not implemented). See method [setUSE\\_INTSLACKS\(\)](#).
- normIsZero: Norm of a vector is considered zero if smaller than this value. See method [setNormIsZero\(\)](#).
- minNormReduction: a cut is generated if the new norm of the row on the continuous nonbasics is reduced by at least this factor (relative reduction).
- away: Look only at basic integer variables whose current value is at least this value from being integer. See method [setAway\(\)](#).
- maxSumMultipliers: maximum sum (in absolute value) of row multipliers
- normalization: normalization factor for the norm of lambda in the coefficient reduction algorithm (convex min problem)
- numRowsReduction: Maximum number of rows in the linear system for norm reduction.
- columnSelectionStrategy: parameter to select which columns should be used for coefficient reduction.
- rowSelectionStrategy: parameter to select which rows should be used for coefficient reduction.
- timeLimit: Time limit (in seconds) for cut generation.
- maxNumCuts: Maximum number of cuts that can be returned at each pass; we could generate more cuts than this number (see below)
- maxNumComputedCuts: Maximum number of cuts that can be computed by the generator at each pass
- maxNonzeroesTab : Rows of the simplex tableau with more than this number of nonzeroes will not be considered for reduction. Only works if RS\_FAST\_\* are defined in [CglRedSplit2](#).
- skipGomory: Skip traditional Gomory cuts, i.e. GMI cuts arising from a single row of the tableau (instead of a combination). Default is 1 (true), because we assume that they are generated by a traditional Gomory generator anyway.

Definition at line 88 of file CglRedSplit2Param.hpp.

## 6.34.2 Member Enumeration Documentation

### 6.34.2.1 enum CglRedSplit2Param::RowSelectionStrategy

Enumerations for parameters.

Row selection strategies; same names as in the paper

Definition at line 94 of file CglRedSplit2Param.hpp.

### 6.34.2.2 enum CglRedSplit2Param::ColumnSelectionStrategy

Column selection strategies; again, look them up in the paper.

Definition at line 122 of file CglRedSplit2Param.hpp.

### 6.34.3 Constructor & Destructor Documentation

**6.34.3.1** `CglRedSplit2Param::CglRedSplit2Param ( bool use_default_strategies = true, double eps = 1e-12, double eps_coeff = 1e-11, double eps_elim = 0.0, double eps_relax_abs = 1e-11, double eps_relax_rel = 1e-13, double max_dyn = 1e6, double min_viol = 1e-3, int max_supp_abs = 1000, double max_supp_rel = 0.1, int use_int_slacks = 0, double norm_zero = 1e-5, double minNormReduction = 0.1, int maxSumMultipliers = 10, double normalization = 0.0001, double away = 0.005, double timeLimit = 60, int maxNumCuts = 10000, int maxNumComputedCuts = 10000, int maxNonzeroesTab = 1000, double columnScalingBoundLAP = 5.0, int skipGomory = 1 )`

Default constructor.

If `use_default_strategies` is true, we add to the list of strategies the default ones. If is false, the list of strategies is left empty (must be populated before usage!).

**6.34.3.2** `CglRedSplit2Param::CglRedSplit2Param ( const CglParam & source, bool use_default_strategies = true, double eps_elim = 0.0, double eps_relax_abs = 1e-11, double eps_relax_rel = 1e-13, double max_dyn = 1e6, double min_viol = 1e-3, double max_supp_rel = 0.1, int use_int_slacks = 0, double norm_zero = 1e-5, double minNormReduction = 0.1, int maxSumMultipliers = 10, double normalization = 0.0001, double away = 0.005, double timeLimit = 60, int maxNumCuts = 10000, int maxNumComputedCuts = 10000, int maxNonzeroesTab = 1000, double columnScalingBoundLAP = 5.0, int skipGomory = 1 )`

Constructor from [CglParam](#).

If `use_default_strategies` is true, we add to the list of strategies the default ones. If is false, the list of strategies is left empty (must be populated before usage!).

### 6.34.4 Member Function Documentation

**6.34.4.1** `virtual void CglRedSplit2Param::setMINVIOL ( double value ) [virtual]`

Set the value of MINVIOL, the minimum violation for the current basic solution in a generated cut.

Default: 1e-3

**6.34.4.2** `void CglRedSplit2Param::setMax_SUPP_ABS ( int value ) [inline]`

Maximum absolute support of the cutting planes.

Default: INT\_MAX. Aliases for consistency with our naming scheme.

Definition at line 211 of file `CglRedSplit2Param.hpp`.

**6.34.4.3** `void CglRedSplit2Param::setMax_SUPP_REL ( double value ) [inline]`

Maximum relative support of the cutting planes.

Default: 0.0. The maximum support is `MAX_SUPP_ABS + MAX_SUPPREL*ncols`.

**6.34.4.4** `virtual void CglRedSplit2Param::setUSE_INTSLACKS ( int value ) [virtual]`

Set the value of USE\_INTSLACKS.

Default: 0

#### 6.34.4.5 virtual void CglRedSplit2Param::addNumRowsReduction ( int *value* ) [virtual]

Set the value of numRowsReduction, max number of rows that are used for each row reduction step.

In particular, the linear system will involve a numRowsReduction\*numRowsReduction matrix

#### 6.34.4.6 virtual void CglRedSplit2Param::addNumRowsReductionLAP ( int *value* ) [virtual]

Set the value of numRowsReductionLAP, max number of rows that are used for each row reduction step during Lift & Project.

In particular, the linear system will involve a numRowsReduction\*numRowsReduction matrix

#### 6.34.4.7 virtual void CglRedSplit2Param::setSkipGomory ( int *value* ) [virtual]

Set the value of skipGomory: should we skip simple Gomory cuts, i.e.

GMI cuts derived from a single row of the simple tableau? This is 1 (true) by default: we only generate cuts from linear combinations of at least two rows.

### 6.34.5 Member Data Documentation

#### 6.34.5.1 double CglRedSplit2Param::EPS\_ELIM [protected]

Epsilon for value of coefficients when eliminating slack variables.

Default: 0.0.

Definition at line 409 of file CglRedSplit2Param.hpp.

#### 6.34.5.2 double CglRedSplit2Param::EPS\_RELAX\_ABS [protected]

Value added to the right hand side of each generated cut to relax it.

Default: 1e-11

Definition at line 413 of file CglRedSplit2Param.hpp.

#### 6.34.5.3 double CglRedSplit2Param::EPS\_RELAX\_REL [protected]

For a generated cut with right hand side rhs\_val, EPS\_RELAX\_EPS \* fabs(rhs\_val) is used to relax the constraint.

Default: 1e-13

Definition at line 418 of file CglRedSplit2Param.hpp.

#### 6.34.5.4 double CglRedSplit2Param::MINVIOL [protected]

Minimum violation for the current basic solution in a generated cut.

Default: 1e-3.

Definition at line 426 of file CglRedSplit2Param.hpp.

#### 6.34.5.5 `double CglRedSplit2Param::normlsZero_` [protected]

Norm of a vector is considered zero if smaller than `normlsZero`; Default: 1e-5.

Definition at line 436 of file `CglRedSplit2Param.hpp`.

#### 6.34.5.6 `double CglRedSplit2Param::away_` [protected]

Use row only if pivot variable should be integer but is more than `away_` from being integer.

Default: 0.005

Definition at line 450 of file `CglRedSplit2Param.hpp`.

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

- `CglRedSplit2Param.hpp`

## 6.35 CglRedSplitParam Class Reference

Class collecting parameters the Reduced-and-split cut generator.

```
#include <CglRedSplitParam.hpp>
```

Inheritance diagram for `CglRedSplitParam`:

Collaboration diagram for `CglRedSplitParam`:

### Public Member Functions

#### Set/get methods

- virtual void `setAway` (const double value)  
*Set away, the minimum distance from being integer used for selecting rows for cut generation; all rows whose pivot variable should be integer but is more than away from integrality will be selected; Default: 0.05.*
- double `getAway` () const  
*Get value of away.*
- virtual void `setLUB` (const double value)  
*Set the value of LUB, value considered large for the absolute value of a lower or upper bound on a variable; Default: 1000.*
- double `getLUB` () const  
*Get the value of LUB.*
- void `setEPS_ELIM` (const double value)  
*Set the value of EPS\_ELIM, epsilon for values of coefficients when eliminating slack variables; Default: 1e-12.*
- double `getEPS_ELIM` () const  
*Get the value of EPS\_ELIM.*
- virtual void `setEPS_RELAX_ABS` (const double eps\_ra)  
*Set EPS\_RELAX\_ABS.*
- double `getEPS_RELAX_ABS` () const  
*Get value of EPS\_RELAX\_ABS.*
- virtual void `setEPS_RELAX_REL` (const double eps\_rr)  
*Set EPS\_RELAX\_REL.*
- double `getEPS_RELAX_REL` () const  
*Get value of EPS\_RELAX\_REL.*



- virtual void **setMAXDYN** (double value)
- double **getMAXDYN** () const  
*Get the value of MAXDYN.*
- virtual void **setMAXDYN\_LUB** (double value)
- double **getMAXDYN\_LUB** () const  
*Get the value of MAXDYN\_LUB.*
- virtual void **setEPS\_COEFF\_LUB** (const double value)  
*Set the value of EPS\_COEFF\_LUB, epsilon for values of coefficients for variables with absolute value of lower or upper bound larger than LUB; Default: 1e-13.*
- double **getEPS\_COEFF\_LUB** () const  
*Get the value of EPS\_COEFF\_LUB.*
- virtual void **setMINVIOL** (double value)  
*Set the value of MINVIOL, the minimum violation for the current basic solution in a generated cut.*
- double **getMINVIOL** () const  
*Get the value of MINVIOL.*
- virtual void **setUSE\_INTSLACKS** (int value)  
*Set the value of USE\_INTSLACKS.*
- int **getUSE\_INTSLACKS** () const  
*Get the value of USE\_INTSLACKS.*
- virtual void **setUSE\_CG2** (int value)  
*Set the value of USE\_CG2.*
- int **getUSE\_CG2** () const  
*Get the value of USE\_CG2.*
- virtual void **setNormIsZero** (const double value)  
*Set the value of normIsZero, the threshold for considering a norm to be 0; Default: 1e-5.*
- double **getNormIsZero** () const  
*Get the value of normIsZero.*
- virtual void **setMinReduc** (const double value)  
*Set the value of minReduc, threshold for relative norm improvement for performing a reduction; Default: 0.05.*
- double **getMinReduc** () const  
*Get the value of minReduc.*
- virtual void **setMaxTab** (const double value)  
*Set the maximum allowed value for (mTab \* mTab \* CoinMax(mTab, nTab)) where mTab is the number of rows used in the combinations and nTab is the number of continuous non basic variables.*
- double **getMaxTab** () const  
*Get the value of maxTab.*

### Constructors and destructors

- **CglRedSplitParam** (const double lub=1000.0, const double eps\_elim=1e-12, const double eps\_relax\_abs=1e-8, const double eps\_relax\_rel=0.0, const double max\_dyn=1e8, const double max\_dyn\_lub=1e13, const double eps\_coeff\_lub=1e-13, const double min\_viol=1e-7, const int use\_int\_slacks=0, const int use\_cg2=0, const double norm\_zero=1e-5, const double min\_reduc=0.05, const double away=0.05, const double max\_tab=1e7)  
*Default constructor.*
- **CglRedSplitParam** (const **CglParam** &source, const double lub=1000.0, const double eps\_elim=1e-12, const double eps\_relax\_abs=1e-8, const double eps\_relax\_rel=0.0, const double max\_dyn=1e8, const double max\_dyn\_lub=1e13, const double eps\_coeff\_lub=1e-13, const double min\_viol=1e-7, const int use\_int\_slacks=0, const int use\_cg2=0, const double norm\_zero=1e-5, const double min\_reduc=0.05, const double away=0.05, const double max\_tab=1e7)  
*Constructor from CglParam.*
- **CglRedSplitParam** (const **CglRedSplitParam** &source)  
*Copy constructor.*
- virtual **CglRedSplitParam** \* **clone** () const  
*Clone.*
- virtual **CglRedSplitParam** & **operator=** (const **CglRedSplitParam** &rhs)  
*Assignment operator.*
- virtual **~CglRedSplitParam** ()  
*Destructor.*

## Protected Attributes

### Parameters

- double [LUB](#)  
*Value considered large for the absolute value of lower or upper bound on a variable.*
- double [EPS\\_ELIM](#)  
*Epsilon for value of coefficients when eliminating slack variables.*
- double [EPS\\_RELAX\\_ABS](#)  
*Value added to the right hand side of each generated cut to relax it.*
- double [EPS\\_RELAX\\_REL](#)  
*For a generated cut with right hand side  $rhs\_val$ ,  $EPS\_RELAX\_EPS * fabs(rhs\_val)$  is used to relax the constraint.*
- double **MAXDYN**
- double **MAXDYN\_LUB**
- double [EPS\\_COEFF\\_LUB](#)  
*Epsilon for value of coefficients for variables with absolute value of lower or upper bound larger than LUB.*
- double [MINVIOL](#)  
*Minimum violation for the current basic solution in a generated cut.*
- int [USE\\_INTSLACKS](#)  
*Use integer slacks to generate cuts if  $USE\_INTSLACKS = 1$ . Default: 0.*
- int [USE\\_CG2](#)  
*Use second way to generate a mixed integer Gomory cut (see methods `generate_cgcut()` and `generate_cgcut_2()`).*
- double [normlsZero](#)  
*Norm of a vector is considered zero if smaller than `normlsZero`; Default:  $1e-5$ .*
- double [minReduc](#)  
*Minimum reduction in percent that must be achieved by a potential reduction step in order to be performed; Between 0 and 1, default: 0.05.*
- double [away\\_](#)  
*Use row only if pivot variable should be integer but is more than `away_` from being integer.*
- double [maxTab\\_](#)  
*Maximum value for  $(mTab * mTab * CoinMax(mTab, nTab))$ .*

### 6.35.1 Detailed Description

Class collecting parameters the Reduced-and-split cut generator.

Parameters of the generator are listed below. Modifying the default values for parameters other than the last four might result in invalid cuts.

- LUB: Value considered large for the absolute value of a lower or upper bound on a variable. See method [setLUB\(\)](#).
- MAXDYN: Maximum ratio between largest and smallest non zero coefficients in a cut. See method [setMAXDYN\(\)](#).
- MAXDYN\_LUB: Maximum ratio between largest and smallest non zero coefficients in a cut involving structural variables with lower or upper bound in absolute value larger than LUB. Should logically be larger or equal to MAXDYN. See method [setMAXDYN\\_LUB\(\)](#).
- EPS\_ELIM: Precision for deciding if a coefficient is zero when eliminating slack variables. See method [setEPS←\\_ELIM\(\)](#).
- EPS\_COEFF\_LUB: Precision for deciding if a coefficient of a generated cut is zero when the corresponding variable has a lower or upper bound larger than LUB in absolute value. See method [setEPS\\_COEFF\\_LUB\(\)](#).
- MINVIOL: Minimum violation for the current basic solution in a generated cut. See method [setMINVIOL\(\)](#).
- USE\_INTSLACKS: Use integer slacks to generate cuts. (not implemented). See method [setUSE\\_INTSLACKS\(\)](#).

- **USE\_CG2**: Use alternative formula to generate a mixed integer Gomory cut (see methods `CglRedSplit::generate_cgcut()` and `CglRedSplit::generate_cgcut_2()`). See method [setUSE\\_CG2\(\)](#).
- **normIsZero**: Norm of a vector is considered zero if smaller than this value. See method [setNormIsZero\(\)](#).
- **minReduc**: Reduction is performed only if the norm of the vector is reduced by this fraction. See method [setMinReduc\(\)](#).
- **away**: Look only at basic integer variables whose current value is at least this value from being integer. See method [setAway\(\)](#).
- **maxTab**: Controls the number of rows selected for the generation. See method [setMaxTab\(\)](#).

Definition at line 61 of file `CglRedSplitParam.hpp`.

### 6.35.2 Member Function Documentation

**6.35.2.1** `virtual void CglRedSplitParam::setMINVIOL ( double value ) [virtual]`

Set the value of MINVIOL, the minimum violation for the current basic solution in a generated cut.

Default: 1e-7

**6.35.2.2** `virtual void CglRedSplitParam::setUSE_INTSLACKS ( int value ) [virtual]`

Set the value of USE\_INTSLACKS.

Default: 0

**6.35.2.3** `virtual void CglRedSplitParam::setUSE_CG2 ( int value ) [virtual]`

Set the value of USE\_CG2.

Default: 0

**6.35.2.4** `virtual void CglRedSplitParam::setMaxTab ( const double value ) [virtual]`

Set the maximum allowed value for  $(mTab * mTab * \text{CoinMax}(mTab, nTab))$  where  $mTab$  is the number of rows used in the combinations and  $nTab$  is the number of continuous non basic variables.

The work of the generator is proportional to  $(mTab * mTab * \text{CoinMax}(mTab, nTab))$ . Reducing the value of  $maxTab$  makes the generator faster, but weaker. Default: 1e7.

### 6.35.3 Member Data Documentation

**6.35.3.1** `double CglRedSplitParam::LUB [protected]`

Value considered large for the absolute value of lower or upper bound on a variable.

Default: 1000.

Definition at line 213 of file `CglRedSplitParam.hpp`.

#### 6.35.3.2 `double CglRedSplitParam::EPS_ELIM` [protected]

Epsilon for value of coefficients when eliminating slack variables.

Default: 1e-12.

Definition at line 217 of file CglRedSplitParam.hpp.

#### 6.35.3.3 `double CglRedSplitParam::EPS_RELAX_ABS` [protected]

Value added to the right hand side of each generated cut to relax it.

Default: 1e-8

Definition at line 221 of file CglRedSplitParam.hpp.

#### 6.35.3.4 `double CglRedSplitParam::EPS_RELAX_REL` [protected]

For a generated cut with right hand side `rhs_val`, `EPS_RELAX_EPS * fabs(rhs_val)` is used to relax the constraint.

Default: 0

Definition at line 226 of file CglRedSplitParam.hpp.

#### 6.35.3.5 `double CglRedSplitParam::EPS_COEFF_LUB` [protected]

Epsilon for value of coefficients for variables with absolute value of lower or upper bound larger than LUB.

Default: 1e-13.

Definition at line 240 of file CglRedSplitParam.hpp.

#### 6.35.3.6 `double CglRedSplitParam::MINVIOL` [protected]

Minimum violation for the current basic solution in a generated cut.

Default: 1e-7.

Definition at line 244 of file CglRedSplitParam.hpp.

#### 6.35.3.7 `int CglRedSplitParam::USE_CG2` [protected]

Use second way to generate a mixed integer Gomory cut (see methods `generate_cgcut()` and `generate_cgcut_2()`).

Default: 0.

Definition at line 251 of file CglRedSplitParam.hpp.

#### 6.35.3.8 `double CglRedSplitParam::normIsZero` [protected]

Norm of a vector is considered zero if smaller than `normIsZero`; Default: 1e-5.

Definition at line 255 of file CglRedSplitParam.hpp.

**6.35.3.9** `double CglRedSplitParam::minReduc` `[protected]`

Minimum reduction in percent that must be achieved by a potential reduction step in order to be performed; Between 0 and 1, default: 0.05.

Definition at line 259 of file CglRedSplitParam.hpp.

**6.35.3.10** `double CglRedSplitParam::away_` `[protected]`

Use row only if pivot variable should be integer but is more than away\_ from being integer.

Definition at line 263 of file CglRedSplitParam.hpp.

**6.35.3.11** `double CglRedSplitParam::maxTab_` `[protected]`

Maximum value for  $(mTab * mTab * \text{CoinMax}(mTab, nTab))$ .

See method [setMaxTab\(\)](#).

Definition at line 267 of file CglRedSplitParam.hpp.

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

- CglRedSplitParam.hpp

## 6.36 CglResidualCapacity Class Reference

Residual Capacity Inequalities Cut Generator Class.

```
#include <CglResidualCapacity.hpp>
```

Inheritance diagram for CglResidualCapacity:

Collaboration diagram for CglResidualCapacity:

### Public Member Functions

#### Get and Set Parameters

- void [setEpsilon](#) (double value)  
*Set Epsilon.*
- double [getEpsilon](#) () const  
*Get Epsilon.*
- void [setTolerance](#) (double value)  
*Set Tolerance.*
- double [getTolerance](#) () const  
*Get Tolerance.*
- void [setDoPreproc](#) (int value)  
*Set doPreproc.*
- bool [getDoPreproc](#) () const  
*Get doPreproc.*

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate Residual Capacity cuts for the model data contained in si.*

### Constructors and destructors

- [CglResidualCapacity](#) ()  
*Default constructor.*
- [CglResidualCapacity](#) (const double tolerance)  
*Alternate Constructor.*
- [CglResidualCapacity](#) (const [CglResidualCapacity](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglResidualCapacity](#) & [operator=](#) (const [CglResidualCapacity](#) &rhs)  
*Assignment operator.*
- virtual [~CglResidualCapacity](#) ()  
*Destructor.*
- virtual void [refreshPrep](#) ()  
*This is to refresh preprocessing.*

### Friends

- void [CglResidualCapacityUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
*A function that tests the methods in the [CglResidualCapacity](#) class.*

## Additional Inherited Members

### 6.36.1 Detailed Description

Residual Capacity Inequalities Cut Generator Class.

References: T Magnanti, P Mirchandani, R Vachani, "The convex hull of two core capacitated network design problems," Math Programming 60 (1993), 233-250.

A Atamturk, D Rajan, "On splittable and unsplittable flow capacitated network design arc-set polyhedra," Math Programming 92 (2002), 315-333.

Definition at line 47 of file [CglResidualCapacity.hpp](#).

### 6.36.2 Member Function Documentation

- 6.36.2.1 virtual void [CglResidualCapacity::generateCuts](#) ( const OsiSolverInterface & *si*, OsiCuts & *cs*, const [CglTreeInfo](#) *info* = [CglTreeInfo](#)() ) [virtual]

Generate Residual Capacity cuts for the model data contained in si.

The generated cuts are inserted in the collection of cuts cs.

Implements [CglCutGenerator](#).

### 6.36.3 Friends And Related Function Documentation

6.36.3.1 void CglResidualCapacityUnitTest ( const OsiSolverInterface \* *siP*, const std::string *mpdDir* ) [friend]

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

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

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

- CglResidualCapacity.hpp

## 6.37 CglSimpleRounding Class Reference

Simple Rounding Cut Generator Class.

```
#include <CglSimpleRounding.hpp>
```

Inheritance diagram for CglSimpleRounding:

Collaboration diagram for CglSimpleRounding:

### Public Member Functions

#### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate simple rounding cuts for the model accessed through the solver interface.*

#### Constructors and destructors

- [CglSimpleRounding](#) ()  
*Default constructor.*
- [CglSimpleRounding](#) (const [CglSimpleRounding](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglSimpleRounding](#) & [operator=](#) (const [CglSimpleRounding](#) &rhs)  
*Assignment operator.*
- virtual [~CglSimpleRounding](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*

### Friends

- void [CglSimpleRoundingUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
*A function that tests the methods in the [CglSimpleRounding](#) class.*

## Additional Inherited Members

### 6.37.1 Detailed Description

Simple Rounding Cut Generator Class.

This class generates simple rounding cuts via the following method: For each constraint, attempt to derive a  $\leq$  inequality in all integer variables by netting out any continuous variables. Divide the resulting integer inequality through by the greatest common denominator (gcd) of the lhs coefficients. Round down the rhs.

Warning: Use with careful attention to data precision.

(Reference: Nemhauser and Wolsey, Integer and Combinatorial Optimization, 1988, pg 211.)

Definition at line 29 of file CglSimpleRounding.hpp.

### 6.37.2 Member Function Documentation

**6.37.2.1** `virtual void CglSimpleRounding::generateCuts ( const OsiSolverInterface & si, OsiCuts & cs, const CglTreeInfo info = CglTreeInfo() ) [virtual]`

Generate simple rounding cuts for the model accessed through the solver interface.

Insert generated cuts into the cut set cs.

Implements [CglCutGenerator](#).

### 6.37.3 Friends And Related Function Documentation

**6.37.3.1** `void CglSimpleRoundingUnitTest ( const OsiSolverInterface * siP, const std::string mpdDir ) [friend]`

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

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

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

- CglSimpleRounding.hpp

## 6.38 CglStored Class Reference

Stored Cut Generator Class.

```
#include <CglStored.hpp>
```

Inheritance diagram for CglStored:

Collaboration diagram for CglStored:

## 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.*

#### Change criterion on whether to include cut.

*Violations of more than this will be added to current cut list (default 1.0e-5)*

- void [setRequiredViolation](#) (double value)  
*Set.*
- double [getRequiredViolation](#) () const  
*Get.*
- void [setProbingInfo](#) ([CglTreeProbingInfo](#) \*info)  
*Takes over ownership of probing info.*

#### Cut stuff

- void [addCut](#) (const OsiCuts &cs)  
*Add cuts.*
- void [addCut](#) (const OsiRowCut &cut)  
*Add a row cut.*
- void [addCut](#) (double lb, double ub, const **CoinPackedVector** &vector)  
*Add a row cut from a packed vector.*
- void [addCut](#) (double lb, double ub, int size, const int \*colIndices, const double \*elements)  
*Add a row cut from elements.*
- int **sizeRowCuts** () const
- const OsiRowCut \* **rowCutPointer** (int index) const
- void [saveStuff](#) (double [bestObjective](#), const double \*[bestSolution](#), const double \*lower, const double \*upper)  
*Save stuff.*
- const double \* [bestSolution](#) () const  
*Best solution (or NULL)*
- double [bestObjective](#) () const  
*Best objective.*
- const double \* [tightLower](#) () const  
*Tight lower bounds.*
- const double \* [tightUpper](#) () const  
*Tight upper bounds.*

#### Constructors and destructors

- [CglStored](#) (int numberColumns=0)  
*Default constructor.*
- [CglStored](#) (const [CglStored](#) &rhs)  
*Copy constructor.*
- [CglStored](#) (const char \*fileName)  
*Constructor from file.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglStored](#) & [operator=](#) (const [CglStored](#) &rhs)  
*Assignment operator.*
- virtual ~[CglStored](#) ()  
*Destructor.*

## Protected Attributes

### Protected member data

- double [requiredViolation\\_](#)  
*Only add if more than this requiredViolation.*
- [CglTreeProbingInfo](#) \* [probingInfo\\_](#)  
*Pointer to probing information.*
- [OsiCuts](#) [cuts\\_](#)  
*Cuts.*
- int [numberColumns\\_](#)  
*Number of columns in model.*
- double \* [bestSolution\\_](#)  
*Best solution (objective at end)*
- double \* [bounds\\_](#)  
*Tight bounds.*

## Additional Inherited Members

### 6.38.1 Detailed Description

Stored Cut Generator Class.

Definition at line 16 of file `CglStored.hpp`.

### 6.38.2 Member Function Documentation

6.38.2.1 `virtual void CglStored::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

Implements [CglCutGenerator](#).

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

- `CglStored.hpp`

## 6.39 CglTreeInfo Class Reference

Information about where the cut generator is invoked from.

```
#include <CglTreeInfo.hpp>
```

Inheritance diagram for `CglTreeInfo`:

Collaboration diagram for `CglTreeInfo`:

## Public Member Functions

- [CglTreeInfo](#) ()  
*Default constructor.*
- [CglTreeInfo](#) (const [CglTreeInfo](#) &)  
*Copy constructor.*
- virtual [CglTreeInfo](#) \* [clone](#) () const  
*Clone.*
- [CglTreeInfo](#) & [operator=](#) (const [CglTreeInfo](#) &rhs)  
*Assignment operator.*
- virtual [~CglTreeInfo](#) ()  
*Destructor.*
- virtual bool [fixes](#) (int, int, int, bool)  
*Take action if cut generator can fix a variable (toValue -1 for down, +1 for up)*
- virtual int [initializeFixing](#) (const OsiSolverInterface \*)  
*Initializes fixing arrays etc - returns >0 if we want to save info 0 if we don't and -1 if is to be used.*

## Public Attributes

- int [level](#)  
*The level of the search tree node.*
- int [pass](#)  
*How many times the cut generator was already invoked in this search tree node.*
- int [formulation\\_rows](#)  
*The number of rows in the original formulation.*
- int [options](#)  
*Options 1 - treat costed integers as important 2 - switch off some stuff as variables semi-integer 4 - set global cut flag if at root node 8 - set global cut flag if at root node and first pass 16 - set global cut flag and make cuts globally valid 32 - last round of cuts did nothing - maybe be more aggressive 64 - in preprocessing stage 128 - looks like solution 256 - want alternate cuts 512 - in sub tree (i.e.*
- bool [inTree](#)  
*Set true if in tree (to avoid ambiguity at first branch)*
- OsiRowCut \*\* [strengthenRow](#)  
*Replacement array.*
- **CoinThreadRandom** \* [randomNumberGenerator](#)  
*Optional pointer to thread specific random number generator.*

### 6.39.1 Detailed Description

Information about where the cut generator is invoked from.

Definition at line 15 of file CglTreeInfo.hpp.

### 6.39.2 Member Data Documentation

#### 6.39.2.1 `int CglTreeInfo::formulation_rows`

The number of rows in the original formulation.

Some generators may not want to consider already generated rows when generating new ones.

Definition at line 24 of file `CglTreeInfo.hpp`.

#### 6.39.2.2 `int CglTreeInfo::options`

Options 1 - treat costed integers as important 2 - switch off some stuff as variables semi-integer 4 - set global cut flag if at root node 8 - set global cut flag if at root node and first pass 16 - set global cut flag and make cuts globally valid 32 - last round of cuts did nothing - maybe be more aggressive 64 - in preprocessing stage 128 - looks like solution 256 - want alternate cuts 512 - in sub tree (i.e.

parent model) 1024 - in must call again mode or after everything mode

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

#### 6.39.2.3 `OsiRowCut** CglTreeInfo::strengthenRow`

Replacement array.

Before Branch and Cut it may be beneficial to strengthen rows rather than adding cuts. If this array is not NULL then the cut generator can place a pointer to the stronger cut in this array which is number of rows in size.

A null (i.e. zero elements and free rhs) cut indicates that the row is useless and can be removed.

The calling function can then replace those rows.

Definition at line 50 of file `CglTreeInfo.hpp`.

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

- `CglTreeInfo.hpp`

## 6.40 CglTreeProbingInfo Class Reference

Inheritance diagram for `CglTreeProbingInfo`:

Collaboration diagram for `CglTreeProbingInfo`:

### Public Member Functions

- [CglTreeProbingInfo](#) ()  
*Default constructor.*
- [CglTreeProbingInfo](#) (const [OsiSolverInterface](#) \*model)  
*Constructor from model.*
- [CglTreeProbingInfo](#) (const [CglTreeProbingInfo](#) &)  
*Copy constructor.*
- virtual [CglTreeInfo](#) \* [clone](#) () const  
*Clone.*

- [CglTreeProbingInfo](#) & [operator=](#) (const [CglTreeProbingInfo](#) &rhs)  
*Assignment operator.*
- virtual [~CglTreeProbingInfo](#) ()  
*Destructor.*
- virtual bool [fixes](#) (int variable, int toValue, int fixedVariable, bool fixedToLower)  
*Take action if cut generator can fix a variable (toValue -1 for down, +1 for up) Returns true if still room, false if not.*
- virtual int [initializeFixing](#) (const OsiSolverInterface \*model)  
*Initializes fixing arrays etc - returns >0 if we want to save info 0 if we don't and -1 if is to be used.*
- int [fixColumns](#) (OsiSolverInterface &si) const  
*Fix entries in a solver using implications.*
- int [fixColumns](#) (int iColumn, int value, OsiSolverInterface &si) const  
*Fix entries in a solver using implications for one variable.*
- int [packDown](#) ()  
*Packs down entries.*
- void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info) const  
*Generate cuts from implications.*
- [CliqueEntry](#) \* [fixEntries](#) ()  
*Entries for fixing variables.*
- int \* [toZero](#) ()  
*Starts of integer variable going to zero.*
- int \* [toOne](#) ()  
*Starts of integer variable going to one.*
- int \* [integerVariable](#) () const  
*List of 0-1 integer variables.*
- int \* [backward](#) () const  
*Backward look up.*
- int [numberVariables](#) () const  
*Number of variables.*
- int [numberIntegers](#) () const  
*Number of 0-1 variables.*

## Protected Attributes

- [CliqueEntry](#) \* [fixEntry\\_](#)  
*Entries for fixing variables.*
- int \* [toZero\\_](#)  
*Starts of integer variable going to zero.*
- int \* [toOne\\_](#)  
*Starts of integer variable going to one.*
- int \* [integerVariable\\_](#)  
*List of 0-1 integer variables.*
- int \* [backward\\_](#)  
*Backward look up.*
- int \* [fixingEntry\\_](#)  
*Entries for fixing variable when collecting.*
- int [numberVariables\\_](#)  
*Number of variables.*

- int [numberIntegers\\_](#)  
*Number of 0-1 variables.*
- int [maximumEntries\\_](#)  
*Maximum number in fixEntry\_.*
- int [numberEntries\\_](#)  
*Number entries in fixingEntry\_ (and fixEntry\_) or -2 if correct style.*

## Additional Inherited Members

### 6.40.1 Detailed Description

Definition at line 85 of file CglTreeInfo.hpp.

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

- CglTreeInfo.hpp

## 6.41 CglTwomir Class Reference

Twostep MIR Cut Generator Class.

```
#include <CglTwomir.hpp>
```

Inheritance diagram for CglTwomir:

Collaboration diagram for CglTwomir:

## Public Member Functions

### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate Two step MIR cuts either from the tableau rows or from the formulation rows.*
- virtual bool [needsOptimalBasis](#) () const  
*Return true if needs optimal basis to do cuts (will return true)*

### Change criterion on which scalings to use (default = 1,1,1,1)

- void [setMirScale](#) (int tmin, int tmax)  
*Set.*
- void [setTwomirScale](#) (int qmin, int qmax)
- void [setAMax](#) (int a)
- void [setMaxElements](#) (int n)
- void [setMaxElementsRoot](#) (int n)
- void [setCutTypes](#) (bool mir, bool twomir, bool tab, bool form)
- void [setFormulationRows](#) (int n)
- int [getTmin](#) () const  
*Get.*
- int [getTmax](#) () const
- int [getQmin](#) () const
- int [getQmax](#) () const
- int [getAMax](#) () const

- int **getMaxElements** () const
- int **getMaxElementsRoot** () const
- int **getIfMir** () const
- int **getIfTwomir** () const
- int **getIfTableau** () const
- int **getIfFormulation** () const

#### Change criterion on which variables to look at. All ones

*more than "away" away from integrality will be investigated (default 0.05)*

- void **setAway** (double value)  
*Set away.*
- double **getAway** () const  
*Get away.*
- void **setAwayAtRoot** (double value)  
*Set away at root.*
- double **getAwayAtRoot** () const  
*Get away at root.*
- virtual int **maxLengthOfCutInTree** () const  
*Return maximum length of cut in tree.*

#### Change way TwoMir works

- void **passInOriginalSolver** (OsiSolverInterface \*solver)  
*Pass in a copy of original solver (clone it)*
- OsiSolverInterface \* **originalSolver** () const  
*Returns original solver.*
- void **setTwomirType** (int type)  
*Set type - 0 normal, 1 add original matrix one, 2 replace.*
- int **twomirType** () const  
*Return type.*

#### Constructors and destructors

- **CglTwomir** ()  
*Default constructor.*
- **CglTwomir** (const **CglTwomir** &)  
*Copy constructor.*
- virtual **CglCutGenerator** \* **clone** () const  
*Clone.*
- **CglTwomir** & **operator=** (const **CglTwomir** &rhs)  
*Assignment operator.*
- virtual **~CglTwomir** ()  
*Destructor.*
- virtual std::string **generateCpp** (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void **refreshSolver** (OsiSolverInterface \*solver)  
*This can be used to refresh any information.*

#### Public Attributes

- std::string **probnam\_**  
*Problem name.*

## Friends

- void [CglTwomirUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)

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

### 6.41.1 Detailed Description

Twostep MIR Cut Generator Class.

Definition at line 91 of file CglTwomir.hpp.

### 6.41.2 Friends And Related Function Documentation

6.41.2.1 void [CglTwomirUnitTest](#) ( const OsiSolverInterface \* *siP*, const std::string *mpdDir* ) [friend]

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

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

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

- CglTwomir.hpp

## 6.42 CglUniqueRowCuts Class Reference

### 6.42.1 Detailed Description

Definition at line 460 of file CglPreProcess.hpp.

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

- CglPreProcess.hpp

## 6.43 CglZeroHalf Class Reference

Zero Half Cut Generator Class.

```
#include <CglZeroHalf.hpp>
```

Inheritance diagram for CglZeroHalf:

Collaboration diagram for CglZeroHalf:

## Public Member Functions

### Generate Cuts

- virtual void [generateCuts](#) (const OsiSolverInterface &si, OsiCuts &cs, const [CglTreeInfo](#) info=[CglTreeInfo](#)())  
*Generate zero half cuts for the model accessed through the solver interface.*



**Sets and Gets**

- int [getFlags](#) () const  
*Get flags.*
- void [setFlags](#) (int value)  
*Set flags.*

**Constructors and destructors**

- [CglZeroHalf](#) ()  
*Default constructor.*
- [CglZeroHalf](#) (const [CglZeroHalf](#) &)  
*Copy constructor.*
- virtual [CglCutGenerator](#) \* [clone](#) () const  
*Clone.*
- [CglZeroHalf](#) & [operator=](#) (const [CglZeroHalf](#) &rhs)  
*Assignment operator.*
- virtual [~CglZeroHalf](#) ()  
*Destructor.*
- virtual std::string [generateCpp](#) (FILE \*fp)  
*Create C++ lines to get to current state.*
- virtual void [refreshSolver](#) (OsiSolverInterface \*solver)  
*This can be used to refresh any information.*

**Friends**

- void [CglZeroHalfUnitTest](#) (const OsiSolverInterface \*siP, const std::string mpdDir)  
*A function that tests the methods in the [CglZeroHalf](#) class.*

**Additional Inherited Members****6.43.1 Detailed Description**

Zero Half Cut Generator Class.

This class generates zero half cuts via the following method:

See -

G. Andreello, A. Caprara, M. Fischetti, "Embedding Cuts in a Branch and Cut Framework: a Computational Study with {0,1/2}-Cuts", INFORMS Journal on Computing 19(2), 229-238, 2007.

Definition at line 26 of file CglZeroHalf.hpp.

**6.43.2 Member Function Documentation**

**6.43.2.1** virtual void [CglZeroHalf::generateCuts](#) ( const OsiSolverInterface & *si*, OsiCuts & *cs*, const [CglTreeInfo](#) *info* = [CglTreeInfo](#) () ) [virtual]

Generate zero half cuts for the model accessed through the solver interface.

Insert generated cuts into the cut set cs.

Implements [CglCutGenerator](#).

### 6.43.3 Friends And Related Function Documentation

6.43.3.1 `void CglZeroHalfUnitTest ( const OsiSolverInterface * siP, const std::string mpdDir ) [friend]`

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

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

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

- CglZeroHalf.hpp

## 6.44 CliqueEntry Struct Reference

Derived class to pick up probing info.

```
#include <CglTreeInfo.hpp>
```

### 6.44.1 Detailed Description

Derived class to pick up probing info.

Definition at line 79 of file CglTreeInfo.hpp.

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

- CglTreeInfo.hpp

## 6.45 CglProbing::CliqueType Struct Reference

Clique type.

```
#include <CglProbing.hpp>
```

### 6.45.1 Detailed Description

Clique type.

Definition at line 230 of file CglProbing.hpp.

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

- CglProbing.hpp

## 6.46 cut Struct Reference

### 6.46.1 Detailed Description

Definition at line 153 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.47 cut\_list Struct Reference

Collaboration diagram for cut\_list:

### 6.47.1 Detailed Description

Definition at line 167 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.48 cutParams Struct Reference

### 6.48.1 Detailed Description

Definition at line 33 of file CglTwomir.hpp.

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

- CglTwomir.hpp

## 6.49 LAP::Cuts Struct Reference

To store extra cuts generated by columns from which they origin.

```
#include <CglLandPUtils.hpp>
```

### Public Member Functions

- int [insertAll](#) (OsiCuts &cs, **CoinRelFitEq** &eq)  
*Puts all the cuts into an OsiCuts.*
- [~Cuts](#) ()  
*Destructor.*
- OsiRowCut \* [rowCut](#) (unsigned int i)  
*Access to row cut indexed by i.*
- const OsiRowCut \* [rowCut](#) (unsigned int i) const  
*const access to row cut indexed by i*
- void [insert](#) (int i, OsiRowCut \*cut)  
*insert a cut for variable i and count number of cuts.*
- int [numberCuts](#) ()  
*Access to number of cuts.*
- void [resize](#) (unsigned int i)  
*resize vector.*

### 6.49.1 Detailed Description

To store extra cuts generated by columns from which they origin.

Definition at line 59 of file CglLandPUtils.hpp.

### 6.49.2 Constructor & Destructor Documentation

#### 6.49.2.1 LAP::Cuts::~~Cuts ( ) `[inline]`

Destructor.

Definition at line 67 of file CglLandPUtils.hpp.

### 6.49.3 Member Function Documentation

#### 6.49.3.1 void LAP::Cuts::insert ( int *i*, OsiRowCut \* *cut* )

insert a cut for variable *i* and count number of cuts.

#### 6.49.3.2 int LAP::Cuts::numberCuts ( ) `[inline]`

Access to number of cuts.

Definition at line 81 of file CglLandPUtils.hpp.

#### 6.49.3.3 void LAP::Cuts::resize ( unsigned int *i* ) `[inline]`

resize vector.

Definition at line 86 of file CglLandPUtils.hpp.

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

- CglLandPUtils.hpp

## 6.50 cycle Struct Reference

Collaboration diagram for cycle:

### 6.50.1 Detailed Description

Definition at line 142 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.51 cycle\_list Struct Reference

Collaboration diagram for cycle\_list:

### 6.51.1 Detailed Description

Definition at line 148 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.52 DGG\_constraint\_t Struct Reference

### 6.52.1 Detailed Description

Definition at line 13 of file CglTwomir.hpp.

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

- CglTwomir.hpp

## 6.53 DGG\_data\_t Struct Reference

Collaboration diagram for DGG\_data\_t:

### 6.53.1 Detailed Description

Definition at line 42 of file CglTwomir.hpp.

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

- CglTwomir.hpp

## 6.54 DGG\_list\_t Struct Reference

Collaboration diagram for DGG\_list\_t:

### 6.54.1 Detailed Description

Definition at line 25 of file CglTwomir.hpp.

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

- CglTwomir.hpp

## 6.55 disaggregationAction Struct Reference

Only useful type of disaggregation is most normal For now just done for 0-1 variables Can be used for building cliques.

```
#include <CglProbing.hpp>
```

### 6.55.1 Detailed Description

Only useful type of disaggregation is most normal For now just done for 0-1 variables Can be used for building cliques.

Definition at line 16 of file CglProbing.hpp.

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

- CglProbing.hpp

## 6.56 edge Struct Reference

Collaboration diagram for edge:

### 6.56.1 Detailed Description

Definition at line 104 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.57 ilp Struct Reference

### 6.57.1 Detailed Description

Definition at line 60 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.58 info\_weak Struct Reference

### 6.58.1 Detailed Description

Definition at line 98 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.59 LAP::LandPMessages Class Reference

Message handler for lift-and-project simplex.

```
#include <CglLandPMessages.hpp>
```

Inheritance diagram for LAP::LandPMessages:

Collaboration diagram for LAP::LandPMessages:

### Public Member Functions

- [LandPMessages](#) ()

*Constructor.*

#### 6.59.1 Detailed Description

Message handler for lift-and-project simplex.

Definition at line 50 of file CglLandPMessages.hpp.

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

- CglLandPMessages.hpp

## 6.60 LAP::LapMessages Class Reference

Output messages for Cgl.

```
#include <CglLandP.hpp>
```

Inheritance diagram for LAP::LapMessages:

Collaboration diagram for LAP::LapMessages:

### Public Member Functions

- [LapMessages](#) ()

*Constructor.*

- virtual [~LapMessages](#) ()

*destructor.*

#### 6.60.1 Detailed Description

Output messages for Cgl.

Definition at line 38 of file CglLandP.hpp.

#### 6.60.2 Constructor & Destructor Documentation

6.60.2.1 `virtual LAP::LapMessages::~~LapMessages ( ) [inline],[virtual]`

destructor.

Definition at line 44 of file CglLandP.hpp.

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

- CglLandP.hpp

## 6.61 log\_var Struct Reference

### 6.61.1 Detailed Description

Definition at line 197 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.62 CglLandP::NoBasisError Class Reference

Inheritance diagram for CglLandP::NoBasisError:

Collaboration diagram for CglLandP::NoBasisError:

### 6.62.1 Detailed Description

Definition at line 218 of file CglLandP.hpp.

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

- CglLandP.hpp

## 6.63 CglLandP::Parameters Class Reference

Class storing parameters.

```
#include <CglLandP.hpp>
```

Inheritance diagram for CglLandP::Parameters:

Collaboration diagram for CglLandP::Parameters:

### Public Member Functions

- [Parameters](#) ()  
*Default constructor (with default values)*
- [Parameters](#) (const [Parameters](#) &other)  
*Copy constructor.*



- [Parameters](#) & [operator=](#) (const [Parameters](#) &other)  
*Assignment operator.*

## Public Attributes

### integer parameters

- int [pivotLimit](#)  
*Max number of pivots before we generate the cut 20.*
- int [pivotLimitInTree](#)  
*Max number of pivots at regular nodes.*
- int [maxCutPerRound](#)  
*Maximum number of cuts generated at a given round.*
- int [failedPivotLimit](#)  
*Maximum number of failed pivots before aborting.*
- int [degeneratePivotLimit](#)  
*maximum number of consecutive degenerate pivots 0*
- int [extraCutsLimit](#)  
*Maximum number of extra rows to generate per round.*

### double parameters

- double [pivotTol](#)  
*Tolerance for small pivots values (should be the same as the solver.*
- double [away](#)  
*A variable have to be at least away from integrity to be generated.*
- double [timeLimit](#)  
*Total time limit for cut generation.*
- double [singleCutTimeLimit](#)  
*Time limit for generating a single cut.*
- double [rhsWeight](#)  
*Weight to put in RHS of normalization if static.*

## Flags

- bool [useTableauRow](#)  
*Do we use tableau row or the disjunction (I don't really get that there should be a way to always use the tableau)*
- bool [modularize](#)  
*Do we apply Egon Balas's Heuristic for modularized cuts.*
- bool [strengthen](#)  
*Do we strengthen the final cut (always do if modularize is 1)*
- bool [countMistakenRc](#)  
*Whether to limit or not the number of mistaken RC (when perturbation is applied).*
- [SeparationSpaces](#) [sepSpace](#)  
*Work in the reduced space (only non-structurals enter the basis)*
- bool [perturb](#)  
*Apply perturbation procedure.*
- [Normalization](#) [normalization](#)  
*How to weight normalization.*
- [RhsWeightType](#) [rhsWeightType](#)  
*How to weight RHS of normalization.*
- LHSnorm [lhs\\_norm](#)  
*How to weight LHS of normalization.*
- [ExtraCutsMode](#) [generateExtraCuts](#)  
*Generate extra constraints from optimal lift-and-project basis.*
- [SelectionRules](#) [pivotSelection](#)  
*Which rule to apply for choosing entering and leaving variables.*

## Additional Inherited Members

### 6.63.1 Detailed Description

Class storing parameters.

#### Remarks

I take all parameters from Ionut's code

Definition at line 107 of file CglLandP.hpp.

### 6.63.2 Member Data Documentation

#### 6.63.2.1 `int CglLandP::Parameters::pivotLimitInTree`

Max number of pivots at regular nodes.

Put a value if you want it lower than the global pivot limit. 100.

Definition at line 124 of file CglLandP.hpp.

#### 6.63.2.2 `int CglLandP::Parameters::extraCutsLimit`

Maximum number of extra rows to generate per round.

Definition at line 133 of file CglLandP.hpp.

#### 6.63.2.3 `double CglLandP::Parameters::timeLimit`

Total time limit for cut generation.

Definition at line 142 of file CglLandP.hpp.

#### 6.63.2.4 `double CglLandP::Parameters::singleCutTimeLimit`

Time limit for generating a single cut.

Definition at line 144 of file CglLandP.hpp.

#### 6.63.2.5 `double CglLandP::Parameters::rhsWeight`

Weight to put in RHS of normalization if static.

Definition at line 146 of file CglLandP.hpp.

#### 6.63.2.6 `bool CglLandP::Parameters::countMistakenRc`

Whether to limit or not the number of mistaken RC (when perturbation is applied).

Definition at line 158 of file CglLandP.hpp.

#### 6.63.2.7 `bool CglLandP::Parameters::perturb`

Apply perturbation procedure.

Definition at line 162 of file CglLandP.hpp.

#### 6.63.2.8 `Normalization CglLandP::Parameters::normalization`

How to weight normalization.

Definition at line 164 of file CglLandP.hpp.

#### 6.63.2.9 `RhsWeightType CglLandP::Parameters::rhsWeightType`

How to weight RHS of normalization.

Definition at line 166 of file CglLandP.hpp.

#### 6.63.2.10 `LHSnorm CglLandP::Parameters::lhs_norm`

How to weight LHS of normalization.

Definition at line 168 of file CglLandP.hpp.

#### 6.63.2.11 `ExtraCutsMode CglLandP::Parameters::generateExtraCuts`

Generate extra constraints from optimal lift-and-project basis.

Definition at line 170 of file CglLandP.hpp.

#### 6.63.2.12 `SelectionRules CglLandP::Parameters::pivotSelection`

Which rule to apply for choosing entering and leaving variables.

Definition at line 172 of file CglLandP.hpp.

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

- CglLandP.hpp

## 6.64 `parity_ilp` Struct Reference

### 6.64.1 Detailed Description

Definition at line 75 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.65 pool\_cut Struct Reference

### 6.65.1 Detailed Description

Definition at line 172 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.66 pool\_cut\_list Struct Reference

Collaboration diagram for pool\_cut\_list:

### 6.66.1 Detailed Description

Definition at line 184 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.67 select\_cut Struct Reference

### 6.67.1 Detailed Description

Definition at line 190 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.68 separation\_graph Struct Reference

Collaboration diagram for separation\_graph:

### 6.68.1 Detailed Description

Definition at line 112 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.69 short\_path\_node Struct Reference

### 6.69.1 Detailed Description

Definition at line 137 of file Cgl012cut.hpp.

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

- Cgl012cut.hpp

## 6.70 CglLandP::SimplexInterfaceError Class Reference

Inheritance diagram for CglLandP::SimplexInterfaceError:

Collaboration diagram for CglLandP::SimplexInterfaceError:

### 6.70.1 Detailed Description

Definition at line 224 of file CglLandP.hpp.

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

- CglLandP.hpp

## 6.71 LAP::TabRow Struct Reference

Inheritance diagram for LAP::TabRow:

Collaboration diagram for LAP::TabRow:

### Public Attributes

- int [num](#)  
*Row number.*
- double [rhs](#)  
*Row right-hand-side.*
- const [CglLandPSimplex](#) \* [si\\_](#)  
*Row of what?*
- bool [modularized\\_](#)  
*Flag to say if row is modularized.*

### 6.71.1 Detailed Description

Definition at line 21 of file CglLandPTabRow.hpp.

### 6.71.2 Member Data Documentation

#### 6.71.2.1 int LAP::TabRow::num

Row number.

Definition at line 24 of file CglLandPTabRow.hpp.

### 6.71.2.2 double LAP::TabRow::rhs

Row right-hand-side.

Definition at line 26 of file CglLandPTabRow.hpp.

### 6.71.2.3 bool LAP::TabRow::modularized\_

Flag to say if row is modularized.

Definition at line 31 of file CglLandPTabRow.hpp.

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

- CglLandPTabRow.hpp

## 6.72 LAP::Validator Class Reference

Class to validate or reject a cut.

```
#include <CglLandPValidator.hpp>
```

### Public Types

- enum [RejectionsReasons](#) { ,  
[SmallViolation](#), [SmallCoefficient](#), [BigDynamic](#), [DenseCut](#),  
[EmptyCut](#), [DummyEnd](#) }  
*Reasons for rejecting a cut.*

### Public Member Functions

- [Validator](#) (double maxFillIn=1., double maxRatio=1e8, double minViolation=0, bool scale=false, double rhs←  
Scale=1)  
*Constructor with default values.*
- int [cleanCut](#) (OsiRowCut &aCut, const double \*solCut, const OsiSolverInterface &si, const [CglParam](#) &par, const  
double \*colLower, const double \*colUpper)  
*Clean an OsiCut.*
- int [cleanCut2](#) (OsiRowCut &aCut, const double \*solCut, const OsiSolverInterface &si, const [CglParam](#) &par, const  
double \*colLower, const double \*colUpper)  
*Clean an OsiCut by another method.*
- int [operator\(\)](#) (OsiRowCut &aCut, const double \*solCut, const OsiSolverInterface &si, const [CglParam](#) &par, const  
double \*colLower, const double \*colUpper)  
*Call the cut cleaner.*

### set functions

- void **setMaxFillIn** (double value)
- void **setMaxRatio** (double value)
- void **setMinViolation** (double value)
- void **setRhsScale** (double v)

**get functions**

- double **getMaxFillIn** ()
- double **getMaxRatio** ()
- double **getMinViolation** ()

**6.72.1 Detailed Description**

Class to validate or reject a cut.

Definition at line 26 of file CglLandPValidator.hpp.

**6.72.2 Member Enumeration Documentation****6.72.2.1 enum LAP::Validator::RejectionsReasons**

Reasons for rejecting a cut.

Enumerator

***SmallViolation*** Violation of the cut is too small.

***SmallCoefficient*** There is a small coefficient we can not get rid off.

***BigDynamic*** Dynamic of coefficient is too important.

***DenseCut*** cut is too dense

***EmptyCut*** After cleaning cut has become empty.

***DummyEnd*** dummy

Definition at line 30 of file CglLandPValidator.hpp.

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

- CglLandPValidator.hpp

File Documentation





# Index

- ~Cuts
  - LAP::Cuts, [110](#)
- ~LapMessages
  - LAP::LapMessages, [113](#)
- AWAY
  - CglGMIParam, [44](#)
- addNumRowsReduction
  - CglRedSplit2Param, [88](#)
- addNumRowsReductionLAP
  - CglRedSplit2Param, [89](#)
- aggressive\_
  - CglCutGenerator, [29](#)
- AllViolatedMigs
  - CglLandP, [53](#)
- AtOptimalBasis
  - CglLandP, [53](#)
- auxiliary\_graph, [19](#)
- away\_
  - CglRedSplit2Param, [90](#)
  - CglRedSplitParam, [95](#)
- bestPivot
  - CglLandP, [53](#)
- BigDynamic
  - LAP::Validator, [121](#)
- Cgl012Cut, [19](#)
- cgl\_arc, [20](#)
- cgl\_graph, [20](#)
- cgl\_node, [20](#)
- CglAllDifferent, [20](#)
  - mayGenerateRowCutsInTree, [22](#)
- CglBK, [22](#)
- CglClique, [23](#)
  - CglClique, [25](#)
  - CglCliqueUnitTest, [25](#)
  - cl\_del\_indices, [27](#)
  - cl\_indices, [26](#)
  - cl\_perm\_indices, [26](#)
  - do\_row\_clique, [26](#)
  - do\_star\_clique, [26](#)
  - generateCuts, [25](#)
  - node\_node, [26](#)
  - petol, [26](#)
  - rcl\_candidate\_length\_threshold, [26](#)
  - scl\_candidate\_length\_threshold, [26](#)
- CglCliqueUnitTest
  - CglClique, [25](#)
- CglCutGenerator, [27](#)
  - aggressive\_, [29](#)
  - generateCpp, [28](#)
  - generateCuts, [28](#)
  - getAggressiveness, [29](#)
  - mayGenerateRowCutsInTree, [29](#)
  - setAggressiveness, [29](#)
- CglDuplicateRow, [29](#)
  - generateCuts, [31](#)
  - outDuplicates, [32](#)
- CglFakeClique, [32](#)
  - CglFakeClique, [33](#)
  - generateCuts, [33](#)
- CglFlowCover, [34](#)
  - CglFlowCoverUnitTest, [35](#)
  - flowPreprocess, [35](#)
  - generateCuts, [35](#)
- CglFlowCoverUnitTest
  - CglFlowCover, [35](#)
- CglFlowVUB, [35](#)
  - CglFlowVUB, [36](#)
  - upper\_, [36](#)
- CglGMI, [36](#)
  - CglGMIUnitTest, [38](#)
  - generateCuts, [38](#)
  - setTrackRejection, [38](#)
- CglGMIParam, [39](#)
  - AWAY, [44](#)
  - ENFORCE\_SCALING, [45](#)
  - EPS\_ELIM, [44](#)
  - EPS\_RELAX\_ABS, [44](#)
  - EPS\_RELAX\_REL, [44](#)
  - getCLEAN\_PROC, [43](#)
  - getENFORCE\_SCALING, [44](#)
  - getINTEGRAL\_SCALE\_CONT, [45](#)
  - INTEGRAL\_SCALE\_CONT, [45](#)
  - MAX\_SUPPORT\_REL, [45](#)
  - MAXDYN, [45](#)
  - MINVIOL, [45](#)
  - setCHECK\_DUPLICATES, [43](#)
  - setCLEAN\_PROC, [43](#)
  - setENFORCE\_SCALING, [44](#)

- setEps, [42](#)
  - setEpsCoeff, [43](#)
  - setINTEGRAL\_SCALE\_CONT, [44](#)
  - setInfinity, [42](#)
  - setMAX\_SUPPORT\_REL, [43](#)
  - setMINVIOL, [43](#)
  - setMaxSupport, [43](#)
  - setUSE\_INTSLACKS, [43](#)
  - USE\_INTSLACKS, [45](#)
- CglGMIUnitTest
  - CglGMI, [38](#)
- CglGomory, [46](#)
  - CglGomoryUnitTest, [48](#)
  - generateCuts, [48](#)
- CglGomoryUnitTest
  - CglGomory, [48](#)
- CglHashLink, [48](#)
- CglImplication, [49](#)
- CglKnapsackCover, [50](#)
  - CglKnapsackCoverUnitTest, [51](#)
  - createCliques, [51](#)
  - generateCuts, [51](#)
- CglKnapsackCoverUnitTest
  - CglKnapsackCover, [51](#)
- CglLandP, [51](#)
  - AllViolatedMigs, [53](#)
  - AtOptimalBasis, [53](#)
  - bestPivot, [53](#)
  - Dynamic, [53](#)
  - ExtraCutsMode, [53](#)
  - Fractional\_rc, [53](#)
  - Full, [53](#)
  - generateCuts, [54](#)
  - initialReducedCosts, [53](#)
  - mostNegativeRc, [53](#)
  - none, [53](#)
  - RhsWeightType, [53](#)
  - SelectionRules, [53](#)
  - SeparationSpaces, [53](#)
  - setLogLevel, [54](#)
  - WhenEnteringBasis, [53](#)
- CglLandP::NoBasisError, [114](#)
- CglLandP::Parameters, [114](#)
  - countMistakenRc, [116](#)
  - extraCutsLimit, [116](#)
  - generateExtraCuts, [117](#)
  - lhs\_norm, [117](#)
  - normalization, [117](#)
  - perturb, [116](#)
  - pivotLimitInTree, [116](#)
  - pivotSelection, [117](#)
  - rhsWeight, [116](#)
  - rhsWeightType, [117](#)
  - singleCutTimeLimit, [116](#)
  - timeLimit, [116](#)
- CglLandP::SimplexInterfaceError, [119](#)
- CglLiftAndProject, [59](#)
  - CglLiftAndProjectUnitTest, [61](#)
  - generateCuts, [60](#)
  - setBeta, [60](#)
- CglLiftAndProjectUnitTest
  - CglLiftAndProject, [61](#)
- CglMessage, [61](#)
- CglMixIntRoundVUB, [64](#)
- CglMixIntRoundVUB2, [65](#)
- CglMixedIntegerRounding, [61](#)
  - generateCuts, [63](#)
- CglMixedIntegerRounding2, [63](#)
  - generateCuts, [64](#)
- CglOddHole, [65](#)
  - CglOddHoleUnitTest, [67](#)
  - generateCuts, [66](#)
- CglOddHoleUnitTest
  - CglOddHole, [67](#)
- CglParam, [67](#)
- CglPreProcess, [68](#)
  - cliquelt, [71](#)
  - originalColumns, [71](#)
  - originalRows, [72](#)
  - preProcess, [71](#)
  - preProcessNonDefault, [71](#)
  - rowTypes, [72](#)
  - setApplicationData, [72](#)
  - setCutoff, [71](#)
  - someFixed, [71](#)
  - tightenPrimalBounds, [71](#)
- CglProbing, [72](#)
  - CglProbingUnitTest, [77](#)
  - createCliques, [76](#)
  - generateCuts, [75](#)
  - mayGenerateRowCutsInTree, [76](#)
  - snapshot, [76](#)
- CglProbing::CliqueType, [108](#)
- CglProbingUnitTest
  - CglProbing, [77](#)
- CglRedSplit, [77](#)
  - CglRedSplitUnitTest, [80](#)
  - generateCuts, [79](#)
  - set\_given\_optsol, [79](#)
  - setMaxTab, [79](#)
- CglRedSplit2, [80](#)
  - CglRedSplit2UnitTest, [82](#)
  - generateCuts, [81](#)
- CglRedSplit2Param, [82](#)
  - addNumRowsReduction, [88](#)
  - addNumRowsReductionLAP, [89](#)
  - away\_, [90](#)
  - CglRedSplit2Param, [88](#)

- ColumnSelectionStrategy, [87](#)
- EPS\_ELIM, [89](#)
- EPS\_RELAX\_ABS, [89](#)
- EPS\_RELAX\_REL, [89](#)
- MINVIOL, [89](#)
- normIsZero\_, [89](#)
- RowSelectionStrategy, [87](#)
- setMax\_SUPP\_ABS, [88](#)
- setMax\_SUPP\_REL, [88](#)
- setMINVIOL, [88](#)
- setSkipGomory, [89](#)
- setUSE\_INTSLACKS, [88](#)
- CglRedSplit2UnitTest
  - CglRedSplit2, [82](#)
- CglRedSplitParam, [90](#)
  - away\_, [95](#)
  - EPS\_COEFF\_LUB, [94](#)
  - EPS\_ELIM, [93](#)
  - EPS\_RELAX\_ABS, [94](#)
  - EPS\_RELAX\_REL, [94](#)
  - LUB, [93](#)
  - MINVIOL, [94](#)
  - maxTab\_, [95](#)
  - minReduc, [94](#)
  - normIsZero, [94](#)
  - setMINVIOL, [93](#)
  - setMaxTab, [93](#)
  - setUSE\_CG2, [93](#)
  - setUSE\_INTSLACKS, [93](#)
  - USE\_CG2, [94](#)
- CglRedSplitUnitTest
  - CglRedSplit, [80](#)
- CglResidualCapacity, [95](#)
  - CglResidualCapacityUnitTest, [97](#)
  - generateCuts, [96](#)
- CglResidualCapacityUnitTest
  - CglResidualCapacity, [97](#)
- CglSimpleRounding, [97](#)
  - CglSimpleRoundingUnitTest, [98](#)
  - generateCuts, [98](#)
- CglSimpleRoundingUnitTest
  - CglSimpleRounding, [98](#)
- CglStored, [98](#)
  - generateCuts, [100](#)
- CglTreeInfo, [100](#)
  - formulation\_rows, [102](#)
  - options, [102](#)
  - strengthenRow, [102](#)
- CglTreeProbingInfo, [102](#)
- CglTwomir, [104](#)
  - CglTwomirUnitTest, [106](#)
- CglTwomirUnitTest
  - CglTwomir, [106](#)
- CglUniqueRowCuts, [106](#)
- CglZeroHalf, [106](#)
  - CglZeroHalfUnitTest, [108](#)
  - generateCuts, [107](#)
- CglZeroHalfUnitTest
  - CglZeroHalf, [108](#)
- cl\_del\_indices
  - CglClique, [27](#)
- cl\_indices
  - CglClique, [26](#)
- cl\_perm\_indices
  - CglClique, [26](#)
- CliqueEntry, [108](#)
- cliquelt
  - CglPreProcess, [71](#)
- ColumnSelectionStrategy
  - CglRedSplit2Param, [87](#)
- computeCglpObjective
  - LAP::CglLandPSimplex, [57](#)
- computeWeights
  - LAP::CglLandPSimplex, [58](#)
- countMistakenRc
  - CglLandP::Parameters, [116](#)
- createCliques
  - CglKnapsackCover, [51](#)
  - CglProbing, [76](#)
- createMIG
  - LAP::CglLandPSimplex, [58](#)
- cut, [108](#)
- cut\_list, [109](#)
- cutParams, [109](#)
- cycle, [110](#)
- cycle\_list, [111](#)
- DGG\_constraint\_t, [111](#)
- DGG\_data\_t, [111](#)
- DGG\_list\_t, [111](#)
- DenseCut
  - LAP::Validator, [121](#)
- disaggregationAction, [112](#)
- do\_row\_clique
  - CglClique, [26](#)
- do\_star\_clique
  - CglClique, [26](#)
- DummyEnd
  - LAP::Validator, [121](#)
- Dynamic
  - CglLandP, [53](#)
- ENFORCE\_SCALING
  - CglGMIParam, [45](#)
- EPS\_COEFF\_LUB
  - CglRedSplitParam, [94](#)
- EPS\_ELIM
  - CglGMIParam, [44](#)
  - CglRedSplit2Param, [89](#)

- CglRedSplitParam, 93
- EPS\_RELAX\_ABS
  - CglGMIParam, 44
  - CglRedSplit2Param, 89
  - CglRedSplitParam, 94
- EPS\_RELAX\_REL
  - CglGMIParam, 44
  - CglRedSplit2Param, 89
  - CglRedSplitParam, 94
- edge, 112
- eliminate\_slacks
  - LAP::CglLandPSimplex, 59
- EmptyCut
  - LAP::Validator, 121
- extraCutsLimit
  - CglLandP::Parameters, 116
- ExtraCutsMode
  - CglLandP, 53
- fastFindBestPivotColumn
  - LAP::CglLandPSimplex, 57
- fastFindCutImprovingPivotRow
  - LAP::CglLandPSimplex, 57
- findBestPivot
  - LAP::CglLandPSimplex, 57
- findBestPivotColumn
  - LAP::CglLandPSimplex, 59
- findCutImprovingPivotRow
  - LAP::CglLandPSimplex, 59
- flowPreprocess
  - CglFlowCover, 35
- formulation\_rows
  - CglTreeInfo, 102
- Fractional\_rc
  - CglLandP, 53
- Full
  - CglLandP, 53
- generateCpp
  - CglCutGenerator, 28
- generateCuts
  - CglClique, 25
  - CglCutGenerator, 28
  - CglDuplicateRow, 31
  - CglFakeClique, 33
  - CglFlowCover, 35
  - CglGMI, 38
  - CglGomory, 48
  - CglKnapsackCover, 51
  - CglLandP, 54
  - CglLiftAndProject, 60
  - CglMixedIntegerRounding, 63
  - CglMixedIntegerRounding2, 64
  - CglOddHole, 66
  - CglProbing, 75
- CglRedSplit, 79
- CglRedSplit2, 81
- CglResidualCapacity, 96
- CglSimpleRounding, 98
- CglStored, 100
- CglZeroHalf, 107
- generateExtraCut
  - LAP::CglLandPSimplex, 57
- generateExtraCuts
  - CglLandP::Parameters, 117
  - LAP::CglLandPSimplex, 56
- generateMig
  - LAP::CglLandPSimplex, 56
- get\_M1\_M2\_M3
  - LAP::CglLandPSimplex, 59
- getAggressiveness
  - CglCutGenerator, 29
- getCLEAN\_PROC
  - CglGMIParam, 43
- getENFORCE\_SCALING
  - CglGMIParam, 44
- getINTEGRAL\_SCALE\_CONT
  - CglGMIParam, 44
- getStatus
  - LAP::CglLandPSimplex, 58
- INTEGRAL\_SCALE\_CONT
  - CglGMIParam, 45
- ilp, 112
- info\_weak, 112
- initialReducedCosts
  - CglLandP, 53
- insert
  - LAP::Cuts, 110
- insertAllExtr
  - LAP::CglLandPSimplex, 57
- isInteger
  - LAP::CglLandPSimplex, 58
- LAP, 17
  - LAP\_messages, 18
  - modularizeRow, 18
  - normCoef, 18
- LAP::CglLandPSimplex, 54
  - computeCglpObjective, 57
  - computeWeights, 58
  - createMIG, 58
  - eliminate\_slacks, 59
  - fastFindBestPivotColumn, 57
  - fastFindCutImprovingPivotRow, 57
  - findBestPivot, 57
  - findBestPivotColumn, 59
  - findCutImprovingPivotRow, 59
  - generateExtraCut, 57
  - generateExtraCuts, 56

- generateMig, [56](#)
- get\_M1\_M2\_M3, [59](#)
- getStatus, [58](#)
- insertAllExtr, [57](#)
- isInteger, [58](#)
- normalizationFactor, [58](#)
- normedCoef, [58](#)
- printCglpBasis, [59](#)
- printEverything, [58](#)
- printTableau, [58](#)
- printTableauLateX, [59](#)
- scaleCut, [58](#)
- strengthenedIntersectionCutCoef, [57](#)
- LAP::Cuts, [109](#)
  - ~Cuts, [110](#)
  - insert, [110](#)
  - numberCuts, [110](#)
  - resize, [110](#)
- LAP::LandPMessages, [113](#)
- LAP::LapMessages, [113](#)
  - ~LapMessages, [113](#)
- LAP::TabRow, [119](#)
  - modularized\_, [120](#)
  - num, [119](#)
  - rhs, [119](#)
- LAP::Validator, [120](#)
  - BigDynamic, [121](#)
  - DenseCut, [121](#)
  - DummyEnd, [121](#)
  - EmptyCut, [121](#)
  - RejectionsReasons, [121](#)
  - SmallCoefficient, [121](#)
  - SmallViolation, [121](#)
- LAP\_messages
  - LAP, [18](#)
- LUB
  - CglRedSplitParam, [93](#)
- lhs\_norm
  - CglLandP::Parameters, [117](#)
- log\_var, [114](#)
- MAX\_SUPPORT\_REL
  - CglGMIParam, [45](#)
- MAXDYN
  - CglGMIParam, [45](#)
- MINVIOL
  - CglGMIParam, [45](#)
  - CglRedSplit2Param, [89](#)
  - CglRedSplitParam, [94](#)
- maxTab\_
  - CglRedSplitParam, [95](#)
- mayGenerateRowCutsInTree
  - CglAllDifferent, [22](#)
  - CglCutGenerator, [29](#)
  - CglProbing, [76](#)
- minReduc
  - CglRedSplitParam, [94](#)
- modularizeRow
  - LAP, [18](#)
- modularized\_
  - LAP::TabRow, [120](#)
- mostNegativeRc
  - CglLandP, [53](#)
- node\_node
  - CglClique, [26](#)
- none
  - CglLandP, [53](#)
- normCoef
  - LAP, [18](#)
- normIsZero
  - CglRedSplitParam, [94](#)
- normIsZero\_
  - CglRedSplit2Param, [89](#)
- normalization
  - CglLandP::Parameters, [117](#)
- normalizationFactor
  - LAP::CglLandPSimplex, [58](#)
- normedCoef
  - LAP::CglLandPSimplex, [58](#)
- num
  - LAP::TabRow, [119](#)
- numberCuts
  - LAP::Cuts, [110](#)
- options
  - CglTreeInfo, [102](#)
- originalColumns
  - CglPreProcess, [71](#)
- originalRows
  - CglPreProcess, [72](#)
- outDuplicates
  - CglDuplicateRow, [32](#)
- parity\_ilp, [117](#)
- perturb
  - CglLandP::Parameters, [116](#)
- petol
  - CglClique, [26](#)
- pivotLimitInTree
  - CglLandP::Parameters, [116](#)
- pivotSelection
  - CglLandP::Parameters, [117](#)
- pool\_cut, [118](#)
- pool\_cut\_list, [118](#)
- preProcess
  - CglPreProcess, [71](#)
- preProcessNonDefault
  - CglPreProcess, [71](#)

printCglpBasis  
     LAP::CglLandPSimplex, 59  
 printEverything  
     LAP::CglLandPSimplex, 58  
 printTableau  
     LAP::CglLandPSimplex, 58  
 printTableauLateX  
     LAP::CglLandPSimplex, 59  
  
 rcl\_candidate\_length\_threshold  
     CglClique, 26  
 RejectionsReasons  
     LAP::Validator, 121  
 resize  
     LAP::Cuts, 110  
 rhs  
     LAP::TabRow, 119  
 rhsWeight  
     CglLandP::Parameters, 116  
 RhsWeightType  
     CglLandP, 53  
 rhsWeightType  
     CglLandP::Parameters, 117  
 RowSelectionStrategy  
     CglRedSplit2Param, 87  
 rowTypes  
     CglPreProcess, 72  
  
 scaleCut  
     LAP::CglLandPSimplex, 58  
 scl\_candidate\_length\_threshold  
     CglClique, 26  
 select\_cut, 118  
 SelectionRules  
     CglLandP, 53  
 separation\_graph, 118  
 SeparationSpaces  
     CglLandP, 53  
 set\_given\_optsol  
     CglRedSplit, 79  
 setAggressiveness  
     CglCutGenerator, 29  
 setApplicationData  
     CglPreProcess, 72  
 setBeta  
     CglLiftAndProject, 60  
 setCHECK\_DUPLICATES  
     CglGMIParam, 43  
 setCLEAN\_PROC  
     CglGMIParam, 43  
 setCutoff  
     CglPreProcess, 71  
 setENFORCE\_SCALING  
     CglGMIParam, 44  
 setEps  
     CglGMIParam, 42  
 setEpsCoeff  
     CglGMIParam, 43  
 setINTEGRAL\_SCALE\_CONT  
     CglGMIParam, 44  
 setInfinity  
     CglGMIParam, 42  
 setLogLevel  
     CglLandP, 54  
 setMAX\_SUPP\_ABS  
     CglRedSplit2Param, 88  
 setMAX\_SUPP\_REL  
     CglRedSplit2Param, 88  
 setMAX\_SUPPORT\_REL  
     CglGMIParam, 43  
 setMINVIOL  
     CglGMIParam, 43  
     CglRedSplit2Param, 88  
     CglRedSplitParam, 93  
 setMaxSupport  
     CglGMIParam, 43  
 setMaxTab  
     CglRedSplit, 79  
     CglRedSplitParam, 93  
 setSkipGomory  
     CglRedSplit2Param, 89  
 setTrackRejection  
     CglGMI, 38  
 setUSE\_CG2  
     CglRedSplitParam, 93  
 setUSE\_INTSLACKS  
     CglGMIParam, 43  
     CglRedSplit2Param, 88  
     CglRedSplitParam, 93  
 short\_path\_node, 118  
 singleCutTimeLimit  
     CglLandP::Parameters, 116  
 SmallCoefficient  
     LAP::Validator, 121  
 SmallViolation  
     LAP::Validator, 121  
 snapshot  
     CglProbing, 76  
 someFixed  
     CglPreProcess, 71  
 strengthenRow  
     CglTreeInfo, 102  
 strengthenedIntersectionCutCoef  
     LAP::CglLandPSimplex, 57  
  
 tightenPrimalBounds  
     CglPreProcess, 71  
 timeLimit  
     CglLandP::Parameters, 116

USE\_CG2  
    CglRedSplitParam, [94](#)  
USE\_INTSLACKS  
    CglGMIParam, [45](#)  
upper\_  
    CglFlowVUB, [36](#)  
  
WhenEnteringBasis  
    CglLandP, [53](#)