

Dip
trunk

Generated by Doxygen 1.8.9.1

Thu Oct 8 2015 22:52:33

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	17
2.1	Class List	17
3	File Index	19
3.1	File List	19
4	Class Documentation	21
4.1	AddOffset< T > Struct Template Reference	21
4.2	AlpsDecompModel Class Reference	21
4.2.1	Detailed Description	22
4.2.2	Member Function Documentation	23
4.2.2.1	fathomAllNodes	23
4.3	AlpsDecompNodeDesc Class Reference	23
4.3.1	Detailed Description	24
4.3.2	Constructor & Destructor Documentation	24
4.3.2.1	AlpsDecompNodeDesc	24
4.3.2.2	AlpsDecompNodeDesc	24
4.3.2.3	~AlpsDecompNodeDesc	24
4.3.3	Member Function Documentation	25
4.3.3.1	setBasis	25
4.3.3.2	getBasis	25
4.3.3.3	setBranchedDir	25
4.3.3.4	getBranchedDir	25
4.3.3.5	setBranched	25
4.3.3.6	getBranched	25
4.3.3.7	encodeAlpsDecomp	25

4.3.3.8	decodeAlpsDecomp	25
4.3.3.9	encode	26
4.3.3.10	decode	26
4.3.4	Member Data Documentation	26
4.3.4.1	branchedDir_	26
4.3.4.2	branched_	26
4.3.4.3	basis_	26
4.4	AlpsDecompParam Class Reference	26
4.4.1	Detailed Description	27
4.4.2	Member Data Documentation	27
4.4.2.1	logFileLevel	27
4.4.2.2	printSolution	27
4.4.2.3	checkMemory	28
4.4.2.4	msgLevel	28
4.4.2.5	nodeLimit	28
4.4.2.6	nodeLogInterval	28
4.5	AlpsDecompSolution Class Reference	28
4.5.1	Detailed Description	29
4.5.2	Member Function Documentation	29
4.5.2.1	getSize	29
4.5.2.2	getValues	29
4.5.2.3	getQuality	29
4.5.2.4	print	30
4.5.3	Member Data Documentation	30
4.5.3.1	m_size	30
4.5.3.2	m_values	30
4.5.3.3	m_quality	30
4.5.3.4	m_app	30
4.6	AlpsDecompTreeNode Class Reference	30
4.6.1	Detailed Description	31
4.6.2	Constructor & Destructor Documentation	31
4.6.2.1	AlpsDecompTreeNode	31
4.6.3	Member Function Documentation	31
4.6.3.1	createNewTreeNode	31
4.6.3.2	chooseBranchingObject	31
4.6.3.3	process	31
4.6.3.4	branch	31

4.7	BcpsDecompModel Class Reference	32
4.7.1	Detailed Description	32
4.7.2	Constructor & Destructor Documentation	32
4.7.2.1	BcpsDecompModel	32
4.7.2.2	BcpsDecompModel	32
4.8	BcpsDecompNodeDesc Class Reference	33
4.8.1	Detailed Description	34
4.8.2	Constructor & Destructor Documentation	34
4.8.2.1	BcpsDecompNodeDesc	34
4.8.2.2	BcpsDecompNodeDesc	34
4.8.2.3	~BcpsDecompNodeDesc	34
4.8.3	Member Function Documentation	34
4.8.3.1	setBasis	34
4.8.3.2	getBasis	34
4.8.3.3	setBranchedDir	34
4.8.3.4	getBranchedDir	35
4.8.3.5	setBranchedInd	35
4.8.3.6	getBranchedInd	35
4.8.3.7	setBranchedVal	35
4.8.3.8	getBranchedVal	35
4.8.3.9	encodeBcpsDecomp	35
4.8.3.10	decodeBcpsDecomp	35
4.8.3.11	encode	35
4.8.3.12	decode	36
4.8.4	Member Data Documentation	36
4.8.4.1	numberRows_	36
4.8.4.2	branchedDir_	36
4.8.4.3	branchedInd_	36
4.8.4.4	branchedVal_	36
4.8.4.5	basis_	36
4.9	BcpsDecompSolution Class Reference	36
4.9.1	Detailed Description	37
4.9.2	Member Function Documentation	37
4.9.2.1	print	37
4.10	BcpsDecompTreeNode Class Reference	37
4.10.1	Detailed Description	38
4.10.2	Constructor & Destructor Documentation	38

4.10.2.1	BcpsDecompTreeNode	38
4.10.3	Member Function Documentation	38
4.10.3.1	createNewTreeNode	38
4.10.3.2	chooseBranchingObject	38
4.10.3.3	process	38
4.10.3.4	branch	38
4.11	DecompAlgo Class Reference	39
4.11.1	Detailed Description	43
4.11.2	Member Function Documentation	43
4.11.2.1	recomposeSolution	43
4.11.2.2	postProcessNode	43
4.11.2.3	postProcessBranch	44
4.11.2.4	generateInitVars	44
4.11.3	Member Data Documentation	44
4.11.3.1	m_masterSI	44
4.11.3.2	m_cutgenSI	44
4.11.3.3	m_cutoffUB	44
4.12	DecompAlgoC Class Reference	44
4.12.1	Detailed Description	45
4.12.2	Member Function Documentation	45
4.12.2.1	recomposeSolution	45
4.12.2.2	generateInitVars	46
4.13	DecompAlgoCGL Class Reference	46
4.13.1	Detailed Description	46
4.14	DecompAlgoD Class Reference	47
4.14.1	Detailed Description	47
4.14.2	Member Function Documentation	47
4.14.2.1	recomposeSolution	47
4.15	DecompAlgoPC Class Reference	48
4.15.1	Detailed Description	48
4.15.2	Member Function Documentation	48
4.15.2.1	recomposeSolution	48
4.16	DecompAlgoRC Class Reference	49
4.16.1	Detailed Description	49
4.17	DecompApp Class Reference	49
4.17.1	Detailed Description	52
4.17.2	Constructor & Destructor Documentation	52

4.17.2.1	DecompApp	52
4.17.3	Member Function Documentation	52
4.17.3.1	setModelObjective	52
4.17.3.2	setModelCore	52
4.17.3.3	setModelRelax	52
4.17.3.4	initDualVector	53
4.17.3.5	APPisUserFeasible	53
4.17.3.6	getDualForGenerateVars	53
4.17.4	Member Data Documentation	53
4.17.4.1	m_decompAlgo	53
4.17.4.2	m_threadIndex	54
4.18	DecompConstraintSet Class Reference	54
4.18.1	Detailed Description	54
4.19	DecompCut Class Reference	54
4.19.1	Detailed Description	54
4.19.2	Member Function Documentation	55
4.19.2.1	increaseEffCnt	55
4.19.2.2	decreaseEffCnt	55
4.19.2.3	increaseEffCnt	55
4.19.2.4	decreaseEffCnt	55
4.20	DecompCutOsi Class Reference	55
4.20.1	Detailed Description	55
4.21	DecompCutPool Class Reference	56
4.21.1	Detailed Description	56
4.22	DecompMainParam Struct Reference	56
4.22.1	Detailed Description	56
4.23	DecompMemPool Class Reference	56
4.23.1	Detailed Description	56
4.24	DecompModel Class Reference	56
4.24.1	Detailed Description	57
4.24.2	Member Data Documentation	57
4.24.2.1	vars	57
4.25	DecompNodeStats Class Reference	57
4.25.1	Detailed Description	58
4.25.2	Member Data Documentation	58
4.25.2.1	objHistoryBound	58
4.26	DecompObjBound Class Reference	58

4.26.1 Detailed Description	59
4.26.2 Member Data Documentation	59
4.26.2.1 bestBound	59
4.26.2.2 bestBoundIP	59
4.27 DecompParam Class Reference	59
4.27.1 Detailed Description	61
4.27.2 Member Function Documentation	61
4.27.2.1 getSettingsImpl	61
4.27.2.2 dumpSettings	62
4.27.3 Member Data Documentation	62
4.27.3.1 BranchStrongIter	62
4.27.3.2 DebugCheckBlocksColumns	62
4.27.3.3 BlockFileFormat	62
4.28 DecompSolution Class Reference	62
4.28.1 Detailed Description	63
4.28.2 Constructor & Destructor Documentation	64
4.28.2.1 DecompSolution	64
4.28.2.2 DecompSolution	64
4.28.2.3 DecompSolution	64
4.28.2.4 DecompSolution	64
4.28.3 Member Function Documentation	64
4.28.3.1 getSize	64
4.28.3.2 getValues	64
4.28.3.3 getQuality	64
4.28.3.4 print	65
4.28.3.5 print	65
4.28.3.6 getSize	65
4.28.3.7 getValues	65
4.28.3.8 getQuality	65
4.28.4 Member Data Documentation	65
4.28.4.1 m_size	65
4.28.4.2 m_values	65
4.28.4.3 m_quality	65
4.29 DecompSolverResult Class Reference	66
4.29.1 Detailed Description	66
4.30 DecompStats Class Reference	66
4.30.1 Detailed Description	66

4.31	DecompSubModel Class Reference	67
4.31.1	Detailed Description	67
4.32	DecompVar Class Reference	67
4.32.1	Detailed Description	68
4.32.2	Member Function Documentation	68
4.32.2.1	increaseEffCnt	68
4.32.2.2	decreaseEffCnt	68
4.33	DecompVarPool Class Reference	68
4.33.1	Detailed Description	68
4.34	DecompWaitingCol Class Reference	68
4.34.1	Detailed Description	68
4.35	DecompWaitingRow Class Reference	69
4.35.1	Detailed Description	69
4.36	DippyAlgoC Class Reference	69
4.36.1	Detailed Description	69
4.36.2	Member Function Documentation	69
4.36.2.1	postProcessBranch	69
4.36.2.2	postProcessNode	69
4.37	DippyAlgoMixin Class Reference	70
4.37.1	Detailed Description	70
4.37.2	Constructor & Destructor Documentation	70
4.37.2.1	DippyAlgoMixin	70
4.38	DippyAlgoPC Class Reference	70
4.38.1	Detailed Description	71
4.38.2	Member Function Documentation	71
4.38.2.1	postProcessBranch	71
4.38.2.2	postProcessNode	71
4.39	DippyAlgoRC Class Reference	71
4.39.1	Detailed Description	72
4.39.2	Member Function Documentation	72
4.39.2.1	postProcessBranch	72
4.39.2.2	postProcessNode	72
4.40	DippyDecompApp Class Reference	72
4.40.1	Detailed Description	73
4.40.2	Member Function Documentation	73
4.40.2.1	APPisUserFeasible	73
4.41	DippyDecompCut Class Reference	74

4.41.1 Detailed Description	74
4.42 is_greater_thanD Class Reference	74
4.42.1 Detailed Description	74
4.43 is_less_thanD Class Reference	74
4.43.1 Detailed Description	74
4.44 OsiData Class Reference	75
4.44.1 Detailed Description	78
4.44.2 Member Function Documentation	78
4.44.2.1 setInfinity	78
4.44.2.2 getRowRhs	78
4.44.2.3 getRowActivity	78
4.44.2.4 setPrimalSol	79
4.44.2.5 getPrimalSol	79
4.44.2.6 setInfinity	79
4.44.2.7 getRowRhs	79
4.44.2.8 getRowActivity	79
4.44.2.9 setPrimalSol	79
4.44.2.10 getPrimalSol	79
4.44.3 Member Data Documentation	79
4.44.3.1 colType	79
4.45 OsiNullSolverInterface Class Reference	80
4.45.1 Detailed Description	87
4.45.2 Member Function Documentation	87
4.45.2.1 getEmptyWarmStart	87
4.45.2.2 getWarmStart	87
4.45.2.3 setWarmStart	87
4.45.2.4 getRowSense	88
4.45.2.5 getRightHandSide	88
4.45.2.6 isInteger	88
4.45.2.7 getRowActivity	88
4.45.2.8 getIterationCount	89
4.45.2.9 getDualRays	89
4.45.2.10 setColLower	89
4.45.2.11 setColUpper	89
4.45.2.12 setRowLower	90
4.45.2.13 setRowUpper	90
4.45.2.14 setObjSense	90

4.45.2.15 setColType	90
4.45.2.16 setRowPrice	90
4.45.2.17 addCol	91
4.45.2.18 deleteCols	91
4.45.2.19 addRow	91
4.45.2.20 deleteRows	91
4.45.2.21 loadProblem	91
4.45.2.22 assignProblem	92
4.45.2.23 loadProblem	92
4.45.2.24 assignProblem	92
4.45.2.25 loadProblem	93
4.45.2.26 loadProblem	93
4.45.2.27 writeMps	93
4.45.2.28 clone	93
4.45.2.29 applyRowCut	93
4.45.2.30 applyColCut	94
4.45.2.31 getEmptyWarmStart	94
4.45.2.32 getWarmStart	94
4.45.2.33 setWarmStart	94
4.45.2.34 getRowSense	94
4.45.2.35 getRightHandSide	95
4.45.2.36 isInteger	95
4.45.2.37 getRowActivity	95
4.45.2.38 getIterationCount	95
4.45.2.39 getDualRays	95
4.45.2.40 setColLower	96
4.45.2.41 setColUpper	96
4.45.2.42 setRowLower	96
4.45.2.43 setRowUpper	96
4.45.2.44 setObjSense	97
4.45.2.45 setColType	97
4.45.2.46 setRowPrice	97
4.45.2.47 addCol	97
4.45.2.48 deleteCols	97
4.45.2.49 addRow	98
4.45.2.50 deleteRows	98
4.45.2.51 loadProblem	98

4.45.2.52 assignProblem	98
4.45.2.53 loadProblem	99
4.45.2.54 assignProblem	99
4.45.2.55 loadProblem	99
4.45.2.56 loadProblem	99
4.45.2.57 writeMps	100
4.45.2.58 clone	100
4.45.2.59 applyRowCut	100
4.45.2.60 applyColCut	100
4.46 Perturb Struct Reference	100
4.46.1 Detailed Description	100
4.47 SOR_IntDbIArrT Struct Reference	101
4.47.1 Detailed Description	101
4.48 SOR_IntDbIT Struct Reference	101
4.48.1 Detailed Description	101
4.49 UtilApp Class Reference	101
4.49.1 Detailed Description	101
4.50 UtilGraphLib Class Reference	101
4.50.1 Detailed Description	101
4.51 UtilsGreaterThan< S, T > Class Template Reference	102
4.51.1 Detailed Description	102
4.52 UtilsLessThan< S, T > Class Template Reference	102
4.52.1 Detailed Description	102
4.53 UtilParameters Class Reference	102
4.53.1 Detailed Description	102
4.54 UtilParamT Struct Reference	102
4.54.1 Detailed Description	102
4.55 UtilTimer Class Reference	103
4.55.1 Detailed Description	103
4.55.2 Member Function Documentation	103
4.55.2.1 reset	103
4.55.2.2 start	103
4.55.2.3 stop	103
4.55.2.4 getCpuTime	103
4.55.2.5 getRealTime	104

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

<code>_EKKfactinfo</code>	<code>[external]</code>	
<code>AbcDualRowPivot</code>	<code>[external]</code>	
<code>AbcDualRowDantzig</code>	<code>[external]</code>	
<code>AbcDualRowSteepest</code>	<code>[external]</code>	
<code>AbcMatrix</code>	<code>[external]</code>	
<code>AbcMatrix2</code>	<code>[external]</code>	
<code>AbcMatrix3</code>	<code>[external]</code>	
<code>AbcNonLinearCost</code>	<code>[external]</code>	
<code>AbcPrimalColumnPivot</code>	<code>[external]</code>	
<code>AbcPrimalColumnDantzig</code>	<code>[external]</code>	
<code>AbcPrimalColumnSteepest</code>	<code>[external]</code>	
<code>AbcSimplexFactorization</code>	<code>[external]</code>	
<code>AbcTolerancesEtc</code>	<code>[external]</code>	
<code>AbcWarmStartOrganizer</code>	<code>[external]</code>	
<code>forcing_constraint_action::action</code>	<code>[external]</code>	
<code>doubleton_action::action</code>	<code>[external]</code>	
<code>tripleton_action::action</code>	<code>[external]</code>	
<code>remove_fixed_action::action</code>	<code>[external]</code>	
<code>std::allocator< T ></code>		
<code>ALPS_PS_STATS</code>	<code>[external]</code>	
<code>AlpsDecompParam</code>		26
<code>AlpsEncoded</code>	<code>[external]</code>	
<code>AlpsKnowledge</code>	<code>[external]</code>	
<code>AlpsModel</code>	<code>[external]</code>	
<code>AlpsDecompModel</code>		21
<code>AlpsSolution</code>	<code>[external]</code>	
<code>AlpsDecompSolution</code>		28
<code>BcpsDecompSolution</code>		36
<code>AlpsSubTree</code>	<code>[external]</code>	
<code>AlpsTreeNode</code>	<code>[external]</code>	
<code>AlpsDecompTreeNode</code>		30
<code>AlpsKnowledgeBroker</code>	<code>[external]</code>	
<code>AlpsKnowledgeBrokerMPI</code>	<code>[external]</code>	
<code>AlpsKnowledgeBrokerSerial</code>	<code>[external]</code>	

AlpsKnowledgePool[external]	
AlpsNodePool[external]	
AlpsSolutionPool[external]	
AlpsSubTreePool[external]	
AlpsNodeDesc[external]	
AlpsDecompNodeDesc	23
AlpsNodeSelection[external]	
AlpsNodeSelectionBest[external]	
AlpsNodeSelectionBreadth[external]	
AlpsNodeSelectionDepth[external]	
AlpsNodeSelectionEstimate[external]	
AlpsNodeSelectionHybrid[external]	
AlpsParameter[external]	
AlpsParameterSet[external]	
AlpsParams[external]	
AlpsPriorityQueue< T >[external]	
AlpsPriorityQueue< AlpsSubTree * >[external]	
AlpsPriorityQueue< AlpsTreeNode * >[external]	
AlpsStrLess[external]	
AlpsTimer[external]	
AlpsTreeSelection[external]	
AlpsTreeSelectionBest[external]	
AlpsTreeSelectionBreadth[external]	
AlpsTreeSelectionDepth[external]	
AlpsTreeSelectionEstimate[external]	
ampl_info[external]	
OsiSolverInterface::ApplyCutsReturnCode[external]	
std::array< T >	
std::auto_ptr< T >	
auxiliary_graph[external]	
CbcGenCtIBlk::babState_struct[external]	
std::basic_string< Char >	
std::string	
std::wstring	
std::basic_string< char >	
std::basic_string< wchar_t >	
BcpsModel	
BcpsDecompModel	32
BcpsNodeDesc	
BcpsDecompNodeDesc	33
BcpsTreeNode	
BcpsDecompTreeNode	37
std::bitset< Bits >	
BitVector128[external]	
blockStruct[external]	
blockStruct3[external]	
ClpNode::branchState[external]	
CbcBaseModel[external]	
CbcBranchDecision[external]	
CbcBranchDefaultDecision[external]	
CbcBranchDynamicDecision[external]	
CbcCompare[external]	
CbcCompareBase[external]	
CbcCompareDefault[external]	

- CbcCompareDepth [external]
- CbcCompareEstimate [external]
- CbcCompareObjective [external]
- CbcConsequence [external]
- CbcFixVariable [external]
- CbcCutGenerator [external]
- CbcCutModifier [external]
 - CbcCutSubsetModifier [external]
- CbcEventHandler [external]
- CbcFathom [external]
 - CbcFathomDynamicProgramming [external]
- CbcFeasibilityBase [external]
- CbcGenCtlBlk [external]
- CbcHeuristic [external]
 - CbcHeuristicCrossover [external]
 - CbcHeuristicDINS [external]
 - CbcHeuristicDive [external]
 - CbcHeuristicDiveCoefficient [external]
 - CbcHeuristicDiveFractional [external]
 - CbcHeuristicDiveGuided [external]
 - CbcHeuristicDiveLineSearch [external]
 - CbcHeuristicDivePseudoCost [external]
 - CbcHeuristicDiveVectorLength [external]
 - CbcHeuristicDW [external]
 - CbcHeuristicDynamic3 [external]
 - CbcHeuristicFPump [external]
 - CbcHeuristicGreedyCover [external]
 - CbcHeuristicGreedyEquality [external]
 - CbcHeuristicGreedySOS [external]
 - CbcHeuristicJustOne [external]
 - CbcHeuristicLocal [external]
 - CbcHeuristicNaive [external]
 - CbcHeuristicPartial [external]
 - CbcHeuristicPivotAndFix [external]
 - CbcHeuristicProximity [external]
 - CbcHeuristicRandRound [external]
 - CbcHeuristicRENS [external]
 - CbcHeuristicRINS [external]
 - CbcHeuristicVND [external]
 - CbcRounding [external]
 - CbcSerendipity [external]
- CbcHeuristicNode [external]
- CbcHeuristicNodeList [external]
- CbcModel [external]
- CbcNauty [external]
- CbcNodeInfo [external]
 - CbcFullNodeInfo [external]
 - CbcPartialNodeInfo [external]
- CbcObjectUpdateData [external]
- CbcOrClpParam [external]
- CbcParam [external]
- CbcGenCtlBlk::cbcParamsInfo_struct [external]
- CbcRowCuts [external]
- CbcSolver [external]

- CbcSolverUsefulData [external]
- CbcSolverUsefulData2 [external]
- CbcStatistics [external]
- CbcStopNow [external]
- CbcStrategy [external]
 - CbcStrategyDefault [external]
 - CbcStrategyDefaultSubTree [external]
 - CbcStrategyNull [external]
- CbcStrongInfo [external]
- CbcSymmetry [external]
- CbcThread [external]
- CbcTree [external]
 - CbcTreeLocal [external]
 - CbcTreeVariable [external]
- CbcUser [external]
- Cgl012Cut [external]
- cgl_arc [external]
- cgl_graph [external]
- cgl_node [external]
- CglBK [external]
- CglCutGenerator [external]
 - CglAllDifferent [external]
 - CglClique [external]
 - CglFakeClique [external]
 - CglDuplicateRow [external]
 - CglFlowCover [external]
 - CglGMI [external]
 - CglGomory [external]
 - CglImplication [external]
 - CglKnapsackCover [external]
 - CglLandP [external]
 - CglLiftAndProject [external]
 - CglMixedIntegerRounding [external]
 - CglMixedIntegerRounding2 [external]
 - CglOddHole [external]
 - CglProbing [external]
 - CglRedSplit [external]
 - CglRedSplit2 [external]
 - CglResidualCapacity [external]
 - CglSimpleRounding [external]
 - CglStored [external]
 - CglTemporary [external]
 - CglTwomir [external]
 - CglZeroHalf [external]
- CglFlowVUB [external]
- CglHashLink [external]
- LAP::CglLandPSimplex [external]
- CglMixIntRoundVUB [external]
- CglMixIntRoundVUB2 [external]
- CglParam [external]
 - CglGMIParam [external]
 - CglLandP::Parameters [external]
 - CglRedSplit2Param [external]
 - CglRedSplitParam [external]

- CglPreProcess [external]
- CglTreeInfo [external]
 - CglTreeProbingInfo [external]
- CglUniqueRowCuts [external]
- CbcGenCtlBlk::chooseStrongCtl_struct [external]
- CliqueEntry [external]
- CglProbing::CliqueType [external]
- ClpCholeskyBase [external]
 - ClpCholeskyDense [external]
 - ClpCholeskyMumps [external]
 - ClpCholeskyTaucs [external]
 - ClpCholeskyUfl [external]
 - ClpCholeskyWssmp [external]
 - ClpCholeskyWssmpKKT [external]
- ClpCholeskyDenseC [external]
- ClpConstraint [external]
 - ClpConstraintAmpl [external]
 - ClpConstraintLinear [external]
 - ClpConstraintQuadratic [external]
- ClpDataSave [external]
- ClpDisasterHandler [external]
 - OsiClpDisasterHandler [external]
- ClpDualRowPivot [external]
 - ClpDualRowDantzig [external]
 - ClpDualRowSteepest [external]
- ClpEventHandler [external]
 - MyEventHandler [external]
- ClpFactorization [external]
- ClpHashValue [external]
- ClpLsqr [external]
- ClpMatrixBase [external]
 - ClpDummyMatrix [external]
 - ClpNetworkMatrix [external]
 - ClpPackedMatrix [external]
 - ClpDynamicMatrix [external]
 - ClpDynamicExampleMatrix [external]
 - ClpGubMatrix [external]
 - ClpGubDynamicMatrix [external]
 - ClpPlusMinusOneMatrix [external]
- ClpModel [external]
 - ClpInterior [external]
 - ClpPdco [external]
 - ClpPredictorCorrector [external]
 - ClpSimplex [external]
 - AbcSimplex [external]
 - AbcSimplexDual [external]
 - AbcSimplexPrimal [external]
 - ClpSimplexDual [external]
 - ClpSimplexOther [external]
 - ClpSimplexPrimal [external]
 - ClpSimplexNonlinear [external]
- ClpNetworkBasis [external]
- ClpNode [external]
- ClpNodeStuff [external]

- ClpNonLinearCost [external]
- ClpObjective [external]
 - ClpAmplObjective [external]
 - ClpLinearObjective [external]
 - ClpQuadraticObjective [external]
- ClpPackedMatrix2 [external]
- ClpPackedMatrix3 [external]
- ClpPdcoBase [external]
- ClpPresolve [external]
- ClpPrimalColumnPivot [external]
 - ClpPrimalColumnDantzig [external]
 - ClpPrimalColumnSteepest [external]
 - ClpPrimalQuadraticDantzig [external]
- ClpSimplexProgress [external]
- ClpSolve [external]
- ClpTrustedData [external]
- CoinAbcAnyFactorization [external]
 - CoinAbcDenseFactorization [external]
 - CoinAbcTypeFactorization [external]
- CoinAbcStack [external]
- CoinAbcStatistics [external]
- CoinAbsFltEq [external]
- CoinArrayWithLength [external]
 - CoinArbitraryArrayWithLength [external]
 - CoinBigIndexArrayWithLength [external]
 - CoinDoubleArrayWithLength [external]
 - CoinFactorizationDoubleArrayWithLength [external]
 - CoinFactorizationLongDoubleArrayWithLength [external]
 - CoinIntArrayWithLength [external]
 - CoinUnsignedIntArrayWithLength [external]
 - CoinVoidStarArrayWithLength [external]
- CoinBaseModel [external]
 - CoinModel [external]
 - CoinStructuredModel [external]
- CoinBuild [external]
- CoinDenseVector< T > [external]
- CoinError [external]
 - CgILandP::NoBasisError [external]
 - CgILandP::SimplexInterfaceError [external]
- CoinExternalVectorFirstGreater_2< class, class, class > [external]
- CoinExternalVectorFirstGreater_3< class, class, class, class > [external]
- CoinExternalVectorFirstLess_2< class, class, class > [external]
- CoinExternalVectorFirstLess_3< class, class, class, class > [external]
- CoinFactorization [external]
- CoinFileIOBase [external]
 - CoinFileInput [external]
 - CoinFileOutput [external]
- CoinFirstAbsGreater_2< class, class > [external]
- CoinFirstAbsGreater_3< class, class, class > [external]
- CoinFirstAbsLess_2< class, class > [external]
- CoinFirstAbsLess_3< class, class, class > [external]
- CoinFirstGreater_2< class, class > [external]
- CoinFirstGreater_3< class, class, class > [external]
- CoinFirstLess_2< class, class > [external]

```
CoinFirstLess_3< class, class, class >[external]
ClpHashValue::CoinHashLink[external]
CoinLpIO::CoinHashLink[external]
CoinMpsIO::CoinHashLink[external]
CoinHashLink[external]
CoinIndexedVector[external]
    CoinPartitionedVector[external]
    LAP::TabRow[external]
CoinLpIO[external]
CoinMessageHandler[external]
    MyMessageHandler[external]
CoinMessages[external]
    AlpsMessage[external]
    CbcMessage[external]
    CglMessage[external]
    ClpMessage[external]
    CoinMessage[external]
    LAP::LandPMessages[external]
    LAP::LapMessages[external]
CoinModelHash[external]
CoinModelHash2[external]
CoinModelHashLink[external]
CoinModelInfo2[external]
CoinModelLink[external]
CoinModelLinkedList[external]
CoinModelTriple[external]
CoinMpsCardReader[external]
CoinMpsIO[external]
CoinOneMessage[external]
CoinOtherFactorization[external]
    CoinDenseFactorization[external]
    CoinOsiFactorization[external]
    CoinSimpFactorization[external]
CoinPackedMatrix[external]
CoinPackedVectorBase[external]
    CoinPackedVector[external]
    CoinShallowPackedVector[external]
CoinPair< S, T >[external]
CoinParam[external]
    CbcCbcParam[external]
    CbcGenParam[external]
    CbcOsiParam[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]
twoxtwo_action[external]
useless_constraint_action[external]
CoinPresolveMonitor[external]
CoinRational[external]
CoinRelFltEq[external]
CoinSearchTreeBase[external]
    CoinSearchTree< class >[external]
CoinSearchTreeCompareBest[external]
CoinSearchTreeCompareBreadth[external]
CoinSearchTreeCompareDepth[external]
CoinSearchTreeComparePreferred[external]
CoinSearchTreeManager[external]
CoinSet[external]
    CoinSosSet[external]
CoinSnapshot[external]
CoinThreadRandom[external]
CoinTimer[external]
CoinTreeNode[external]
    CbcNode[external]
CoinTreeSiblings[external]
CoinTriple< S, T, U >[external]
CoinWarmStart[external]
    CoinWarmStartBasis[external]
        AbcWarmStart[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
OsiCuts::const_iterator[external]
std::wstring::const_iterator
std::list< T >::const_iterator

```

std::forward_list< T >::const_iterator	
std::multimap< K, T >::const_iterator	
std::unordered_multimap< K, T >::const_iterator	
std::multiset< K >::const_iterator	
std::unordered_multiset< K >::const_iterator	
std::unordered_set< K >::const_iterator	
std::vector< T >::const_iterator	
std::set< K >::const_iterator	
std::unordered_map< K, T >::const_iterator	
std::map< K, T >::const_iterator	
std::deque< T >::const_iterator	
std::string::const_iterator	
std::basic_string< Char >::const_iterator	
std::basic_string< Char >::const_reverse_iterator	
std::string::const_reverse_iterator	
std::deque< T >::const_reverse_iterator	
std::map< K, T >::const_reverse_iterator	
std::unordered_map< K, T >::const_reverse_iterator	
std::set< K >::const_reverse_iterator	
std::unordered_set< K >::const_reverse_iterator	
std::vector< T >::const_reverse_iterator	
std::unordered_multiset< K >::const_reverse_iterator	
std::multiset< K >::const_reverse_iterator	
std::list< T >::const_reverse_iterator	
std::unordered_multimap< K, T >::const_reverse_iterator	
std::multimap< K, T >::const_reverse_iterator	
std::forward_list< T >::const_reverse_iterator	
std::wstring::const_reverse_iterator	
cut[external]	
cut_list[external]	
cutParams[external]	
LAP::Cuts[external]	
cycle[external]	
cycle_list[external]	
CbcGenCtlBlk::debugSolInfo_struct[external]	
DecompAlgo	39
DecompAlgoC	44
DippyAlgoC	69
DecompAlgoC	44
DecompAlgoD	47
DecompAlgoPC	48
DecompAlgoD	47
DippyAlgoPC	70
DecompAlgoPC	48
DecompAlgoRC	49
DippyAlgoRC	71
DecompAlgoRC	49
DecompAlgoCGL	46
DecompApp	49
DippyDecompApp	72
DecompConstraintSet	54
DecompCut	54
DecompCutOsi	55

DecompCutOsi	55
DippyDecompCut	74
DecompMainParam	56
DecompMemPool	56
DecompModel	56
DecompSubModel	67
DecompNodeStats	57
DecompObjBound	58
DecompParam	59
DecompSolution	62
DecompSolverResult	66
DecompStats	66
DecompVar	67
DecompWaitingCol	68
DecompWaitingRow	69
DeletePtrObject[external]	
std::deque< T >	
std::deque< StdVectorDouble >	
DGG_constraint_t[external]	
DGG_data_t[external]	
DGG_list_t[external]	
DippyAlgoMixin	70
DippyAlgoC	69
DippyAlgoPC	70
DippyAlgoRC	71
disaggregationAction[external]	
CbcGenCtlBlk::djFixCtl_struct[external]	
dropped_zero[external]	
dualColumnResult[external]	
edge[external]	
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 >	
CbcGenCtlBlk::genParamsInfo_struct[external]	
glp_prob[external]	
Idiot[external]	

```
IdiotResult[external]
ilp[external]
Info[external]
info_weak[external]
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_istringstream< char >
      basic_istringstream< 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_istringstream
        std::istringstream
        std::wistringstream
      std::istream
      std::wistream
    std::basic_ostream
      basic_iostream< char >
      basic_iostream< wchar_t >
      basic_ofstream< char >
      basic_ofstream< wchar_t >
      basic_ostringstream< char >
      basic_ostringstream< wchar_t >
      std::basic_iostream
      std::basic_ofstream
        std::ofstream
        std::wofstream
      std::basic_ostringstream
        std::ostringstream
        std::wostringstream
      std::ostream
      std::wostream
```

std::ios	
std::wios	
is_greater_thanD	74
is_less_thanD	74
OsiCuts::iterator[external]	
std::basic_string< Char >::iterator	
std::map< K, T >::iterator	
std::set< K >::iterator	
std::vector< T >::iterator	
std::unordered_set< K >::iterator	
std::unordered_map< K, T >::iterator	
std::unordered_multiset< K >::iterator	
std::multiset< K >::iterator	
std::multimap< K, T >::iterator	
std::unordered_multimap< K, T >::iterator	
std::deque< T >::iterator	
std::forward_list< T >::iterator	
std::list< T >::iterator	
std::string::iterator	
std::wstring::iterator	
std::list< T >	
std::list< DecompCut * >	
std::list< DecompVar * >	
log_var[external]	
std::map< K, T >	
std::map< AlpsKnowledgeType, AlpsKnowledgePool * >	
std::map< int, DecompConstraintSet * >	
std::map< int, DecompModel >	
std::map< int, DecompSubModel >	
std::map< int, int >	
std::map< int, OsiSolverInterface * >	
std::map< int, std::vector< DecompModel > >	
std::map< int, std::vector< DecompSubModel > >	
std::map< int, std::vector< int > >	
std::map< PyObject *, int >	
std::map< std::string, std::string >	
std::multimap< K, T >	
std::multiset< K >	
Options[external]	
OsiAuxInfo[external]	
OsiBabSolver[external]	
OsiBranchingInformation[external]	
OsiBranchingObject[external]	
CbcBranchingObject[external]	
CbcCliqueBranchingObject[external]	
CbcCutBranchingObject[external]	
CbcDummyBranchingObject[external]	
CbcFixingBranchingObject[external]	
CbcIntegerBranchingObject[external]	
CbcDynamicPseudoCostBranchingObject[external]	
CbcIntegerPseudoCostBranchingObject[external]	
CbcLongCliqueBranchingObject[external]	
CbcLotsizeBranchingObject[external]	
CbcNWayBranchingObject[external]	


```

    CbcOrbitalBranchingObject [external]
    CbcSOSBranchingObject [external]
    OsiTwoWayBranchingObject [external]
    OsiBiLinearBranchingObject [external]
    OsiIntegerBranchingObject [external]
    OsiLinkBranchingObject [external]
    OsiLotsizeBranchingObject [external]
    OsiSOSBranchingObject [external]
    OsiOldLinkBranchingObject [external]
    OsiChooseVariable [external]
    OsiChooseStrong [external]
    OsiChooseStrongSubset [external]
    OsiCut [external]
    OsiColCut [external]
    OsiRowCut [external]
    CbcCountRowCut [external]
    OsiRowCut2 [external]
    OsiCuts [external]

    OsiData ..... 75
    OsiHotInfo [external]
    OsiLinkedBound [external]
    OsiObject [external]
    CbcObject [external]
    CbcBranchCut [external]
    CbcBranchAllDifferent [external]
    CbcBranchToFixLots [external]
    CbcClique [external]
    CbcFollowOn [external]
    CbcGeneral [external]
    CbcIdiotBranch [external]
    CbcLotsize [external]
    CbcNWay [external]
    CbcSimpleInteger [external]
    CbcSimpleIntegerDynamicPseudoCost [external]
    CbcSimpleIntegerPseudoCost [external]
    CbcSOS [external]
    OsiObject2 [external]
    OsiBiLinear [external]
    OsiBiLinearEquality [external]
    OsiLotsize [external]
    OsiSimpleInteger [external]
    OsiSimpleFixedInteger [external]
    OsiUsesBiLinear [external]
    OsiSOS [external]
    OsiLink [external]
    OsiOldLink [external]
    OsiOneLink [external]
    CbcGenCtlBlk::osiParamsInfo_struct [external]
    OsiPresolve [external]
    OsiPseudoCosts [external]
    OsiRowCutDebugger [external]
    OsiSolverBranch [external]
    OsiSolverInterface [external]
    OsiCbcSolverInterface [external]

```

OsiClpSolverInterface [external]	
CbcOsiSolver [external]	
OsiSolverLink [external]	
OsiSolverLinearizedQuadratic [external]	
OsiCpxSolverInterface [external]	
OsiGlpkSolverInterface [external]	
OsiGrbSolverInterface [external]	
OsiMskSolverInterface [external]	
OsiNullSolverInterface	80
OsiNullSolverInterface	80
OsiSpxSolverInterface [external]	
OsiXprSolverInterface [external]	
OsiSolverResult [external]	
Outfo [external]	
ClpSimplexOther::parametricsData [external]	
parity_ilp [external]	
Perturb	100
AbcSimplexPrimal::pivotStruct [external]	
pool_cut [external]	
pool_cut_list [external]	
presolvehlink [external]	
std::priority_queue< T >	
CbcHeuristicDive::PriorityType [external]	
PseudoReducedCost [external]	
std::queue< T >	
Coin::ReferencedObject [external]	
std::string::reverse_iterator	
std::unordered_multiset< K >::reverse_iterator	
std::multimap< K, T >::reverse_iterator	
std::unordered_set< K >::reverse_iterator	
std::basic_string< Char >::reverse_iterator	
std::deque< T >::reverse_iterator	
std::set< K >::reverse_iterator	
std::wstring::reverse_iterator	
std::multiset< K >::reverse_iterator	
std::list< T >::reverse_iterator	
std::unordered_multimap< K, T >::reverse_iterator	
std::unordered_map< K, T >::reverse_iterator	
std::map< K, T >::reverse_iterator	
std::vector< T >::reverse_iterator	
std::forward_list< T >::reverse_iterator	
scatterStruct [external]	
select_cut [external]	
separation_graph [external]	
std::set< K >	
std::set< int >	
short_path_node [external]	
std::smart_ptr< T >	
Coin::SmartPtr< T > [external]	
SOR_IntDbIArrT	101
SOR_IntDbIT	101
std::stack< T >	
symrec [external]	
std::system_error	

OsiUnitTest::TestOutcome [external]	
OsiUnitTest::TestOutcomes [external]	
std::thread	
TotalWorkload [external]	
unary_function	
AddOffset< T >	21
AddOffset< T >	21
std::unique_ptr< T >	
std::unordered_map< K, T >	
std::unordered_multimap< K, T >	
std::unordered_multiset< K >	
std::unordered_set< K >	
UtilApp	101
UtilGraphLib	101
UtilsGreaterThan< S, T >	102
UtilsLessThan< S, T >	102
UtilParameters	102
UtilParamT	102
UtilTimer	103
std::valarray< T >	
LAP::Validator [external]	
std::vector< T >	
DecompCutPool	56
DecompCutPool	56
DecompVarPool	68
DecompVarPool	68
std::vector< bool >	
std::vector< CbcNode * >	
std::vector< char >	
std::vector< CoinBigIndex >	
std::vector< ColumnSelectionStrategy >	
std::vector< DecompColType >	
std::vector< DecompModel >	
std::vector< DecompObjBound >	
std::vector< DecompRowType >	
std::vector< DecompSolution * >	
std::vector< DecompSubModel >	
std::vector< DecompWaitingCol >	
std::vector< DecompWaitingRow >	
std::vector< double >	
std::vector< int >	
std::vector< RowSelectionStrategy >	
std::vector< std::pair< int, double > >	
std::vector< std::pair< std::string, AlpsParameter > >	
std::vector< std::string >	
std::vector< std::vector< double > >	
std::vector< string >	
std::vector< vector< double > >	
std::weak_ptr< T >	
K	
S	
T	
U	

Chapter 2

Class Index

2.1 Class List

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

AddOffset< T >	21
AlpsDecompModel	
Derivation of AlpsModel for DECOMP	21
AlpsDecompNodeDesc	
Derivation of AlpsNodeDesc for DECOMP	23
AlpsDecompParam	
Parameters passed through to Alps	26
AlpsDecompSolution	28
AlpsDecompTreeNode	30
BcpsDecompModel	32
BcpsDecompNodeDesc	33
BcpsDecompSolution	
This class holds a MIP feasible primal solution	36
BcpsDecompTreeNode	37
DecompAlgo	
Base class for DECOMP algorithms	39
DecompAlgoC	
Class for DECOMP algorithm Cutting Plane Method	44
DecompAlgoCGL	
An interface to CGL cut generator library	46
DecompAlgoD	
Class for DECOMP algorithm Decom	47
DecompAlgoPC	
Class for DECOMP algorithm Price and Cut	48
DecompAlgoRC	49
DecompApp	
The main application class	49
DecompConstraintSet	54
DecompCut	54
DecompCutOsi	55
DecompCutPool	56
DecompMainParam	56
DecompMemPool	56
DecompModel	56

DecompNodeStats	57
DecompObjBound	58
DecompParam	59
DecompSolution	62
DecompSolverResult	
Storage of solver result	66
DecompStats	66
DecompSubModel	67
DecompVar	67
DecompVarPool	68
DecompWaitingCol	68
DecompWaitingRow	69
DippyAlgoC	
Python-enabled DecompAlgoC	69
DippyAlgoMixin	
Mixin class for Dip Algorithms	70
DippyAlgoPC	
Python-enabled DecompAlgoPC	70
DippyAlgoRC	
Python-enabled DecompAlgoRC	71
DippyDecompApp	
A DecompApp that links Python to DIP	72
DippyDecompCut	74
is_greater_thanD	74
is_less_thanD	74
OsiData	
Class collecting pointers on data for OsiEmpty	75
OsiNullSolverInterface	80
Perturb	100
SOR_IntDblArrT	101
SOR_IntDblT	101
UtilApp	101
UtilGraphLib	101
UtilsGreaterThan< S, T >	102
UtilsLessThan< S, T >	102
UtilParameters	102
UtilParamT	102
UtilTimer	103

Chapter 3

File Index

3.1 File List

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

AlpsDecompModel.h	??
AlpsDecompNodeDesc.h	??
AlpsDecompParam.h	??
AlpsDecompSolution.h	??
AlpsDecompTreeNode.h	??
BcpsDecompModel.h	??
BcpsDecompNodeDesc.h	??
BcpsDecompSolution.h	??
BcpsDecompTreeNode.h	??
config_default.h	??
config_dip_default.h	??
Decomp.h	??
DecompAlgo.h	??
old/DecompAlgo.h	??
DecompAlgo.old.h	??
DecompAlgoC.h	??
old/DecompAlgoC.h	??
DecompAlgoCGL.h	??
DecompAlgoD.h	??
old/DecompAlgoD.h	??
DecompAlgoPC.h	??
old/DecompAlgoPC.h	??
DecompAlgoRC.h	??
old/DecompAlgoRC.h	??
DecompApp.h	??
old/DecompApp.h	??
DecompConfig.h	??
DecompConstants.h	??
DecompConstraintSet.h	??
old/DecompConstraintSet.h	??
DecompCut.h	??
old/DecompCut.h	??
DecompCutOsi.h	??
old/DecompCutOsi.h	??

DecompCutPool.h	??
old/DecompCutPool.h	??
DecompMemPool.h	??
old/DecompMemPool.h	??
DecompModel.h	??
old/DecompModel.h	??
DecompParam.h	??
old/DecompParam.h	??
DecompPortable.h	??
DecompSolution.h	??
old/DecompSolution.h	??
DecompSolverResult.h	??
DecompStats.h	??
old/DecompStats.h	??
DecompTypes.h	??
DecompVar.h	??
old/DecompVar.h	??
DecompVarPool.h	??
old/DecompVarPool.h	??
DecompWaitingCol.h	??
old/DecompWaitingCol.h	??
DecompWaitingRow.h	??
old/DecompWaitingRow.h	??
DippyDecompAlgo.h	??
DippyDecompApp.h	??
DippyDecompCut.h	??
DippyPythonUtils.h	??
hmetis.h	??
OsiData.hpp	??
OsiData2.hpp	??
OsiNullSolverInterface.hpp	??
OsiNullSolverInterface2.hpp	??
UtilApp.h	??
old/UtilGraphLib.h	??
UtilGraphLib.h	??
old/UtilHash.h	??
UtilHash.h	??
UtilKnapsack.h	??
old/UtilMacros.h	??
UtilMacros.h	??
old/UtilMacrosAlps.h	??
UtilMacrosAlps.h	??
UtilMacrosDecomp.h	??
old/UtilParameters.h	??
UtilParameters.h	??
UtilTimer.h	??

Chapter 4

Class Documentation

4.1 AddOffset< T > Struct Template Reference

Inheritance diagram for AddOffset< T >:

4.2 AlpsDecompModel Class Reference

Derivation of **AlpsModel** for DECOMP.

```
#include <AlpsDecompModel.h>
```

Inheritance diagram for AlpsDecompModel:

Collaboration diagram for AlpsDecompModel:

Public Member Functions

Constructors and destructor.

- [AlpsDecompModel](#) ()
Default constructors.
- **AlpsDecompModel** ([UtilParameters](#) &utilParam, [DecompAlgo](#) *decompAlgo)
- virtual [~AlpsDecompModel](#) ()
Destructor.

Virtual functions from AlpsModel.

- virtual **AlpsTreeNode** * [createRoot](#) ()
Create the root node of the search tree.
- virtual bool [fathomAllNodes](#) ()
Return true, if all nodes can be fathomed.

Helper functions.

- AlpsExitStatus [solve](#) ()
Solve with ALPS and DECOMP.
- void [setAlpsSettings](#) ()

Set the ALPS parameters.

- void [setDecompAlgo](#) ([DecompAlgo](#) *decompAlgo)

Solve with ALPS and DECOMP.

Set/get methods.

- [DecompAlgo](#) * [getDecompAlgo](#) ()
Get a ptr to the decomp algorithm vector.
- [AlpsDecompParam](#) & [getParam](#) ()
- const int [getNumCoreRows](#) () const
Get number of rows in core decomp model.
- const int [getNumCoreCols](#) () const
Get number of cols in core decomp model.
- const std::vector< std::string > & [getColNames](#) () const
Get the column names in core decomp model.
- const std::vector< std::string > & [getRowNames](#) () const
Get the row names in core decomp model.
- const [DecompSolution](#) * [getBestSolution](#) () const
Get the best solution found.
- const double [getGlobalLB](#) () const
- const double [getGlobalUB](#) () const
- const int [getSolStatus](#) () const
- const int [getNumNodesProcessed](#) () const

4.2.1 Detailed Description

Derivation of **AlpsModel** for DECOMP.

An object derived from **AlpsModel**. It interfaces with DECOMP methods through a pointer to the active [DecompAlgo](#).

- [AlpsDecompModel](#) is derived from **AlpsModel**
 - **AlpsModel** has no pure virtual functions
- **AlpsModel** is derived from **AlpsKnowledge**
 - **AlpsKnowledge** has no pure virtual functions

Virtual methods that should be derived here:

- createRoot

See also

AlpsModel
[DecompAlgo](#)

Definition at line 65 of file AlpsDecompModel.h.

4.2.2 Member Function Documentation

4.2.2.1 virtual bool AlpsDecompModel::fathomAllNodes () [virtual]

Return true, if all nodes can be fathomed.

Reimplemented from **AlpsModel**.

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

- AlpsDecompModel.h

4.3 AlpsDecompNodeDesc Class Reference

Derivation of **AlpsNodeDesc** for DECOMP.

```
#include <AlpsDecompNodeDesc.h>
```

Inheritance diagram for AlpsDecompNodeDesc:

Collaboration diagram for AlpsDecompNodeDesc:

Data.

- double * [lowerBounds_](#)
lower bounds in original space
- double * [upperBounds_](#)
upper bounds in original space
- int [numberCols_](#)
number of columns in original space
- int [branchedDir_](#)
Branched direction to create it.
- std::vector< std::pair< int, double > > [branched_](#)
Branched set of indices/values to create it.
- **CoinWarmStartBasis** * [basis_](#)
Warm start.
- [AlpsDecompNodeDesc](#) ()
Default constructor.
- [AlpsDecompNodeDesc](#) (**AlpsModel** *m)
Useful constructor.
- **AlpsDecompNodeDesc** ([AlpsDecompModel](#) *m, const double *lb, const double *ub)
- virtual [~AlpsDecompNodeDesc](#) ()
Destructor.
- void [setBasis](#) (**CoinWarmStartBasis** *&ws)
Set basis.
- **CoinWarmStartBasis** * [getBasis](#) () const
Get warm start basis.
- void [setBranchedDir](#) (int d)
Set branching direction.
- int [getBranchedDir](#) () const

- Get branching direction.*
- void [setBranched](#) (std::vector< std::pair< int, double > > b)
- Set branching set.*
- std::vector< std::pair< int, double > > [getBranched](#) () const
- Get branching set.*
- virtual AlpsReturnStatus [encode](#) (**AlpsEncoded** *encoded) const
- Pack node description into an encoded.*
- virtual AlpsReturnStatus [decode](#) (**AlpsEncoded** &encoded)
- Unpack a node description from an encoded.*
- AlpsReturnStatus [encodeAlpsDecomp](#) (**AlpsEncoded** *encoded) const
- Pack blis portion of node description into an encoded.*
- AlpsReturnStatus [decodeAlpsDecomp](#) (**AlpsEncoded** &encoded)
- Unpack blis portion of node description from an encoded.*

4.3.1 Detailed Description

Derivation of **AlpsNodeDesc** for DECOMP.

An object derived from **AlpsNodeDesc**. This stores the description of a search tree node. For DECOMP, we are not using differencing, so, we only need to store the bounds set during branching.

[AlpsDecompNodeDesc](#) is derived from **AlpsNodeDesc** **AlpsModel** has no pure virtual functions

Virtual methods that should be derived here: encode decode

See also

AlpsNodeDesc

Definition at line 56 of file AlpsDecompNodeDesc.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 AlpsDecompNodeDesc::AlpsDecompNodeDesc () [inline]

Default constructor.

Definition at line 91 of file AlpsDecompNodeDesc.h.

4.3.2.2 AlpsDecompNodeDesc::AlpsDecompNodeDesc (AlpsModel * m) [inline]

Useful constructor.

Definition at line 98 of file AlpsDecompNodeDesc.h.

4.3.2.3 virtual AlpsDecompNodeDesc::~~AlpsDecompNodeDesc () [inline],[virtual]

Destructor.

Definition at line 121 of file AlpsDecompNodeDesc.h.

4.3.3 Member Function Documentation

4.3.3.1 void AlpsDecompNodeDesc::setBasis (CoinWarmStartBasis * & *ws*) [inline]

Set basis.

Definition at line 136 of file AlpsDecompNodeDesc.h.

4.3.3.2 CoinWarmStartBasis* AlpsDecompNodeDesc::getBasis () const [inline]

Get warm start basis.

Definition at line 146 of file AlpsDecompNodeDesc.h.

4.3.3.3 void AlpsDecompNodeDesc::setBranchedDir (int *d*) [inline]

Set branching direction.

Definition at line 151 of file AlpsDecompNodeDesc.h.

4.3.3.4 int AlpsDecompNodeDesc::getBranchedDir () const [inline]

Get branching direction.

Definition at line 156 of file AlpsDecompNodeDesc.h.

4.3.3.5 void AlpsDecompNodeDesc::setBranched (std::vector< std::pair< int, double > > *b*) [inline]

Set branching set.

Definition at line 161 of file AlpsDecompNodeDesc.h.

4.3.3.6 std::vector< std::pair<int, double> > AlpsDecompNodeDesc::getBranched () const [inline]

Get branching set.

Definition at line 166 of file AlpsDecompNodeDesc.h.

4.3.3.7 AlpsReturnStatus AlpsDecompNodeDesc::encodeAlpsDecomp (AlpsEncoded * *encoded*) const [inline],
[protected]

Pack blis portion of node description into an encoded.

Definition at line 177 of file AlpsDecompNodeDesc.h.

4.3.3.8 AlpsReturnStatus AlpsDecompNodeDesc::decodeAlpsDecomp (AlpsEncoded & *encoded*) [inline],
[protected]

Unpack blis portion of node description from an encoded.

Definition at line 197 of file AlpsDecompNodeDesc.h.

4.3.3.9 `virtual AlpsReturnStatus AlpsDecompNodeDesc::encode (AlpsEncoded * encoded) const` `[inline],`
`[virtual]`

Pack node description into an encoded.

Reimplemented from **AlpsNodeDesc**.

Definition at line 220 of file AlpsDecompNodeDesc.h.

4.3.3.10 `virtual AlpsReturnStatus AlpsDecompNodeDesc::decode (AlpsEncoded & encoded)` `[inline], [virtual]`

Unpack a node description from an encoded.

Fill member data.

Reimplemented from **AlpsNodeDesc**.

Definition at line 227 of file AlpsDecompNodeDesc.h.

4.3.4 Member Data Documentation

4.3.4.1 `int AlpsDecompNodeDesc::branchedDir_`

Branched direction to create it.

Definition at line 80 of file AlpsDecompNodeDesc.h.

4.3.4.2 `std::vector< std::pair<int, double> > AlpsDecompNodeDesc::branched_`

Branched set of indices/values to create it.

Definition at line 82 of file AlpsDecompNodeDesc.h.

4.3.4.3 `CoinWarmStartBasis* AlpsDecompNodeDesc::basis_`

Warm start.

Definition at line 86 of file AlpsDecompNodeDesc.h.

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

- AlpsDecompNodeDesc.h

4.4 AlpsDecompParam Class Reference

Parameters passed through to Alps.

```
#include <AlpsDecompParam.h>
```

Public Member Functions

Helper functions.

- void **getSettings** ([UtilParameters](#) ¶m)

- void **dumpSettings** (std::ostream *os=&std::cout)

Constructors and destructor.

- [AlpsDecompParam](#) ()
Default constructors.
- **AlpsDecompParam** ([UtilParameters](#) &utilParam)
- [~AlpsDecompParam](#) ()
Destructor.

Public Attributes

Data.

- int [logFileLevel](#)
The level of log file.
- bool [printSolution](#)
Print solution to screen and log if have a solution and msgLevel and logFileLevel permits.
- bool [checkMemory](#)
Check memory.
- int [msgLevel](#)
The level of printing messages on screen.
- int [nodeLimit](#)
The max number of nodes can be processed.
- int [nodeLogInterval](#)
Node log interval.

4.4.1 Detailed Description

Parameters passed through to Alps.

Definition at line 32 of file AlpsDecompParam.h.

4.4.2 Member Data Documentation

4.4.2.1 int AlpsDecompParam::logFileLevel

The level of log file.

- 0: no print to screen (Default)
- 1: summary
- 2: moderate
- 3: verbose

Definition at line 48 of file AlpsDecompParam.h.

4.4.2.2 bool AlpsDecompParam::printSolution

Print solution to screen and log if have a solution and msgLevel and logFileLevel permits.

Default: false.

Definition at line 54 of file AlpsDecompParam.h.

4.4.2.3 bool AlpsDecompParam::checkMemory

Check memory.

Default: false

Definition at line 59 of file AlpsDecompParam.h.

4.4.2.4 int AlpsDecompParam::msgLevel

The level of printing messages on screen.

Used to control master and general messages.

- 0: no print to screen
- 1: summary
- 2: moderate (Default)
- 3: verbose

Definition at line 69 of file AlpsDecompParam.h.

4.4.2.5 int AlpsDecompParam::nodeLimit

The max number of nodes can be processed.

Default: ALPS_INT_MAX

Definition at line 74 of file AlpsDecompParam.h.

4.4.2.6 int AlpsDecompParam::nodeLogInterval

Node log interval.

Default: 100

Definition at line 79 of file AlpsDecompParam.h.

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

- AlpsDecompParam.h

4.5 AlpsDecompSolution Class Reference

Inheritance diagram for AlpsDecompSolution:

Collaboration diagram for AlpsDecompSolution:

Public Member Functions

Helper functions (public).

- const int [getSize](#) () const

- Get length of solution.*
- const double * [getValues](#) () const
Get solution values.
- const double [getQuality](#) () const
Get quality of solution.
- **AlpsDecompSolution** ()
- **AlpsDecompSolution** (const int size, const double *values, const double quality, const [DecompApp](#) *app=N←ULL, const int depth=-1, const AlpsNodeIndex_t index=-1)
- virtual ~**AlpsDecompSolution** ()
- virtual void [print](#) (std::ostream &os) const
Print out the solution.

Protected Attributes

- int [m_size](#)
Length of solution (number of columns).
- double * [m_values](#)
Solution values.
- double [m_quality](#)
Quality of solution (bound wrt to objective).
- const [DecompApp](#) * [m_app](#)
Pointer to [DecompApp](#) for the print function.

4.5.1 Detailed Description

Definition at line 24 of file AlpsDecompSolution.h.

4.5.2 Member Function Documentation

4.5.2.1 const int AlpsDecompSolution::getSize () const [inline]

Get length of solution.

Definition at line 42 of file AlpsDecompSolution.h.

4.5.2.2 const double* AlpsDecompSolution::getValues () const [inline]

Get solution values.

Definition at line 47 of file AlpsDecompSolution.h.

4.5.2.3 const double AlpsDecompSolution::getQuality () const [inline]

Get quality of solution.

Definition at line 52 of file AlpsDecompSolution.h.

4.5.2.4 `virtual void AlpsDecompSolution::print (std::ostream & os) const` `[inline],[virtual]`

Print out the solution.

Reimplemented from **AlpsSolution**.

Definition at line 86 of file AlpsDecompSolution.h.

4.5.3 Member Data Documentation

4.5.3.1 `int AlpsDecompSolution::m_size` `[protected]`

Length of solution (number of columns).

Definition at line 27 of file AlpsDecompSolution.h.

4.5.3.2 `double* AlpsDecompSolution::m_values` `[protected]`

Solution values.

Definition at line 30 of file AlpsDecompSolution.h.

4.5.3.3 `double AlpsDecompSolution::m_quality` `[protected]`

Quality of solution (bound wrt to objective).

Definition at line 33 of file AlpsDecompSolution.h.

4.5.3.4 `const DecompApp* AlpsDecompSolution::m_app` `[protected]`

Pointer to [DecompApp](#) for the print function.

Definition at line 36 of file AlpsDecompSolution.h.

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

- AlpsDecompSolution.h

4.6 AlpsDecompTreeNode Class Reference

Inheritance diagram for AlpsDecompTreeNode:

Collaboration diagram for AlpsDecompTreeNode:

Public Member Functions

- [AlpsDecompTreeNode](#) ()
Default constructor.
- virtual [~AlpsDecompTreeNode](#) ()
Destructor.
- **AlpsTreeNode** * [createNewTreeNode](#) (**AlpsNodeDesc** *&desc) const

Create a new node based on given desc.

- int `chooseBranchingObject` (**AlpsModel** *model)

To be defined.

- int `process` (bool isRoot=false, bool rampUp=false)

Performing the bounding operation.

- std::vector< **CoinTriple**< **AlpsNodeDesc** *, AlpsNodeStatus, double > > `branch` ()

Takes the explicit description of the current active node and creates the children's descriptions, which contain information about how the branching is to be done.

4.6.1 Detailed Description

Definition at line 28 of file AlpsDecompTreeNode.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 AlpsDecompTreeNode::AlpsDecompTreeNode () [inline]

Default constructor.

Definition at line 40 of file AlpsDecompTreeNode.h.

4.6.3 Member Function Documentation

4.6.3.1 AlpsTreeNode* AlpsDecompTreeNode::createNewTreeNode (AlpsNodeDesc *& desc) const [virtual]

Create a new node based on given desc.

Implements **AlpsTreeNode**.

4.6.3.2 int AlpsDecompTreeNode::chooseBranchingObject (AlpsModel * model)

To be defined.

??

4.6.3.3 int AlpsDecompTreeNode::process (bool isRoot = false, bool rampUp = false)

Performing the bounding operation.

4.6.3.4 std::vector< CoinTriple<AlpsNodeDesc*, AlpsNodeStatus, double> > AlpsDecompTreeNode::branch ()

Takes the explicit description of the current active node and creates the children's descriptions, which contain information about how the branching is to be done.

The stati of the children are AlpsNodeStatusCandidate.

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

- AlpsDecompTreeNode.h

4.7 BcpsDecompModel Class Reference

Inheritance diagram for BcpsDecompModel:

Collaboration diagram for BcpsDecompModel:

Public Member Functions

- [BcpsDecompModel](#) ()
Default constructor.
- [BcpsDecompModel](#) ([DecompAlgo](#) *decompAlgo)
Default constructor.
- virtual [~BcpsDecompModel](#) ()
Destructor.
- void [readInstance](#) (const char *dataFile)
Read in the instance data.
- **AlpsTreeNode** * [createRoot](#) ()
create the root node
- void [init](#) ()
initialize the model data
- [DecompAlgo](#) * [getDecompAlgo](#) () const
get a ptr to the decomp algo
- void [setActiveNode](#) (**AlpsTreeNode** *node)
set active node
- void [addNumNodes](#) (int newNodes=1)
increment node count

4.7.1 Detailed Description

Definition at line 40 of file BcpsDecompModel.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 [BcpsDecompModel::BcpsDecompModel](#) () `[inline]`

Default constructor.

Definition at line 64 of file BcpsDecompModel.h.

4.7.2.2 [BcpsDecompModel::BcpsDecompModel](#) ([DecompAlgo](#) * *decompAlgo*) `[inline]`

Default constructor.

Definition at line 71 of file BcpsDecompModel.h.

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

- [BcpsDecompModel.h](#)

4.8 BcpsDecompNodeDesc Class Reference

Inheritance diagram for BcpsDecompNodeDesc:

Collaboration diagram for BcpsDecompNodeDesc:

Public Member Functions

- [BcpsDecompNodeDesc](#) ()
Default constructor.
- [BcpsDecompNodeDesc](#) (BcpsModel *m)
Useful constructor.
- virtual [~BcpsDecompNodeDesc](#) ()
Destructor.
- void [setBasis](#) (CoinWarmStartBasis *&ws)
Set basis.
- **CoinWarmStartBasis** * [getBasis](#) () const
Get warm start basis.
- void [setBranchedDir](#) (int d)
Set branching direction.
- int [getBranchedDir](#) () const
Get branching direction.
- void [setBranchedInd](#) (int d)
Set branching object index.
- int [getBranchedInd](#) () const
Get branching object index.
- void [setBranchedVal](#) (double d)
Set branching value.
- double [getBranchedVal](#) () const
Get branching direction.
- virtual AlpsReturnStatus [encode](#) (**AlpsEncoded** *encoded) const
Pack node description into an encoded.
- virtual AlpsReturnStatus [decode](#) (**AlpsEncoded** &encoded)
Unpack a node description from an encoded.

Public Attributes

- int [numberRows](#)_
Number of rows in problem (before these cuts).
- int [branchedDir](#)_
Branched direction to create it.
- int [branchedInd](#)_
Branched object index to create it.
- double [branchedVal](#)_
Branched value to create it.
- **CoinWarmStartBasis** * [basis](#)_
Warm start.

Protected Member Functions

- AlpsReturnStatus [encodeBcpsDecomp](#) (**AlpsEncoded** *encoded) const
Pack blis portion of node description into an encoded.
- AlpsReturnStatus [decodeBcpsDecomp](#) (**AlpsEncoded** &encoded)
Unpack blis portion of node description from an encoded.

4.8.1 Detailed Description

Definition at line 40 of file BcpsDecompNodeDesc.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 BcpsDecompNodeDesc::BcpsDecompNodeDesc () [\[inline\]](#)

Default constructor.

Definition at line 79 of file BcpsDecompNodeDesc.h.

4.8.2.2 BcpsDecompNodeDesc::BcpsDecompNodeDesc (BcpsModel * *m*) [\[inline\]](#)

Useful constructor.

Definition at line 90 of file BcpsDecompNodeDesc.h.

4.8.2.3 virtual BcpsDecompNodeDesc::~~BcpsDecompNodeDesc () [\[inline\]](#), [\[virtual\]](#)

Destructor.

Definition at line 119 of file BcpsDecompNodeDesc.h.

4.8.3 Member Function Documentation

4.8.3.1 void BcpsDecompNodeDesc::setBasis (CoinWarmStartBasis *& *ws*) [\[inline\]](#)

Set basis.

Definition at line 132 of file BcpsDecompNodeDesc.h.

4.8.3.2 CoinWarmStartBasis* BcpsDecompNodeDesc::getBasis () const [\[inline\]](#)

Get warm start basis.

Definition at line 139 of file BcpsDecompNodeDesc.h.

4.8.3.3 void BcpsDecompNodeDesc::setBranchedDir (int *d*) [\[inline\]](#)

Set branching direction.

Definition at line 145 of file BcpsDecompNodeDesc.h.

4.8.3.4 `int BcpsDecompNodeDesc::getBranchedDir () const [inline]`

Get branching direction.

Definition at line 148 of file BcpsDecompNodeDesc.h.

4.8.3.5 `void BcpsDecompNodeDesc::setBranchedInd (int d) [inline]`

Set branching object index.

Definition at line 151 of file BcpsDecompNodeDesc.h.

4.8.3.6 `int BcpsDecompNodeDesc::getBranchedInd () const [inline]`

Get branching object index.

Definition at line 154 of file BcpsDecompNodeDesc.h.

4.8.3.7 `void BcpsDecompNodeDesc::setBranchedVal (double d) [inline]`

Set branching value.

Definition at line 157 of file BcpsDecompNodeDesc.h.

4.8.3.8 `double BcpsDecompNodeDesc::getBranchedVal () const [inline]`

Get branching direction.

Definition at line 160 of file BcpsDecompNodeDesc.h.

4.8.3.9 `AlpsReturnStatus BcpsDecompNodeDesc::encodeBcpsDecomp (AlpsEncoded * encoded) const [inline],
[protected]`

Pack blis portion of node description into an encoded.

Definition at line 169 of file BcpsDecompNodeDesc.h.

4.8.3.10 `AlpsReturnStatus BcpsDecompNodeDesc::decodeBcpsDecomp (AlpsEncoded & encoded) [inline],
[protected]`

Unpack blis portion of node description from an encoded.

Definition at line 193 of file BcpsDecompNodeDesc.h.

4.8.3.11 `virtual AlpsReturnStatus BcpsDecompNodeDesc::encode (AlpsEncoded * encoded) const [inline],
[virtual]`

Pack node description into an encoded.

Definition at line 220 of file BcpsDecompNodeDesc.h.

4.8.3.12 `virtual AlpsReturnStatus BcpsDecompNodeDesc::decode (AlpsEncoded & encoded)` `[inline], [virtual]`

Unpack a node description from an encoded.

Fill member data.

Definition at line 230 of file BcpsDecompNodeDesc.h.

4.8.4 Member Data Documentation

4.8.4.1 `int BcpsDecompNodeDesc::numberOfRows_`

Number of rows in problem (before these cuts).

This means that for top of chain it must be rows at continuous

Definition at line 58 of file BcpsDecompNodeDesc.h.

4.8.4.2 `int BcpsDecompNodeDesc::branchedDir_`

Branched direction to create it.

Definition at line 64 of file BcpsDecompNodeDesc.h.

4.8.4.3 `int BcpsDecompNodeDesc::branchedInd_`

Branched object index to create it.

Definition at line 67 of file BcpsDecompNodeDesc.h.

4.8.4.4 `double BcpsDecompNodeDesc::branchedVal_`

Branched value to create it.

Definition at line 70 of file BcpsDecompNodeDesc.h.

4.8.4.5 `CoinWarmStartBasis* BcpsDecompNodeDesc::basis_`

Warm start.

Definition at line 74 of file BcpsDecompNodeDesc.h.

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

- BcpsDecompNodeDesc.h

4.9 BcpsDecompSolution Class Reference

This class holds a MIP feasible primal solution.

```
#include <BcpsDecompSolution.h>
```

Inheritance diagram for BcpsDecompSolution:

Collaboration diagram for BcpsDecompSolution:

Public Member Functions

- double [getObjValue](#) () const
Get the objective value value.
- int [getSize](#) () const
Get the size of the solution.
- const double * [getColSolution](#) () const
Get the column solution.
- double [getColSolution](#) (int i) const
Get item i in the solution vector.
- virtual void [print](#) (std::ostream &os) const
Print out the solution.

4.9.1 Detailed Description

This class holds a MIP feasible primal solution.

Definition at line 35 of file BcpsDecompSolution.h.

4.9.2 Member Function Documentation

4.9.2.1 virtual void BcpsDecompSolution::print (std::ostream & os) const [virtual]

Print out the solution.

Reimplemented from **AlpsSolution**.

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

- BcpsDecompSolution.h

4.10 BcpsDecompTreeNode Class Reference

Inheritance diagram for BcpsDecompTreeNode:

Collaboration diagram for BcpsDecompTreeNode:

Public Member Functions

- [BcpsDecompTreeNode](#) ()
Default constructor.
- virtual [~BcpsDecompTreeNode](#) ()
Destructor.
- **AlpsTreeNode** * [createNewTreeNode](#) (**AlpsNodeDesc** *&desc) const
Create a new node based on given desc.
- int [chooseBranchingObject](#) (BcpsModel *model)
To be defined.
- int [installSubProblem](#) (BcpsModel *model)
intall subproblem

- `int bound (BcpsModel *model)`
Bounding procedure.
- `int process (bool isRoot, bool rampUp)`
Performing the bounding operation.
- `std::vector< CoinTriple< AlpsNodeDesc *, AlpsNodeStatus, double > > branch ()`
Takes the explicit description of the current active node and creates the children's descriptions, which contain information about how the branching is to be done.

4.10.1 Detailed Description

Definition at line 51 of file BcpsDecompTreeNode.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 BcpsDecompTreeNode::BcpsDecompTreeNode () [inline]

Default constructor.

Definition at line 68 of file BcpsDecompTreeNode.h.

4.10.3 Member Function Documentation

4.10.3.1 AlpsTreeNode* BcpsDecompTreeNode::createNewTreeNode (AlpsNodeDesc *& desc) const

Create a new node based on given desc.

4.10.3.2 int BcpsDecompTreeNode::chooseBranchingObject (BcpsModel * model)

To be defined.

??

4.10.3.3 int BcpsDecompTreeNode::process (bool isRoot, bool rampUp)

Performing the bounding operation.

4.10.3.4 std::vector< CoinTriple<AlpsNodeDesc*, AlpsNodeStatus, double> > BcpsDecompTreeNode::branch ()

Takes the explicit description of the current active node and creates the children's descriptions, which contain information about how the branching is to be done.

The stati of the children are AlpsNodeStatusCandidate.

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

- BcpsDecompTreeNode.h

4.11 DecompAlgo Class Reference

Base class for DECOMP algorithms.

```
#include <DecompAlgo.h>
```

Inheritance diagram for DecompAlgo:

Collaboration diagram for DecompAlgo:

Public Member Functions

Pure virtual functions.

- virtual void [createMasterProblem](#) (DecompVarList &initVars)
Create the master problem (all algorithms must define this function).
- void [loadSIFromModel](#) (**OsiSolverInterface** *si, bool doInt=false)
- virtual void [recomposeSolution](#) (const double *solution, double *rsolution)
Compose solution in x-space from current space.

Virtual functions.

- virtual DecompStatus [processNode](#) (const [AlpsDecompTreeNode](#) *node, const double globalLB=-DecompInf, const double globalUB=DecompInf)
The main DECOMP process loop for a node.
- const [AlpsDecompTreeNode](#) * [getCurrentNode](#) () const
Provide the current node the algorithm is solving.
- virtual void [postProcessNode](#) (DecompStatus decompStatus)
Do some information sending after the current node has been processed.
- virtual void [postProcessBranch](#) (DecompStatus decompStatus)
Do some information sending after the current node has been branched.
- virtual int [generateInitVars](#) (DecompVarList &initVars)
Generate initial variables for master problem (PC/DC/RC).
- virtual DecompStatus [solutionUpdate](#) (const DecompPhase phase, const bool resolve=true, const int max←InnerIter=COIN_INT_MAX, const int maxOuterIter=COIN_INT_MAX)
Update of the solution vectors (primal and/or dual).
- virtual void [phaseUpdate](#) (DecompPhase &phase, DecompStatus &status)
Update of the phase for process loop.
- virtual void [phaseInit](#) (DecompPhase &phase)
Run the initial phase for processing node.
- virtual void [phaseDone](#) ()
Run the done phase for processing node.
- virtual bool [updateObjBound](#) (const double mostNegRC=-DecompBigNum)
Calculate the current LB and update best/history.
- virtual void [solveMasterAsMIP](#) ()
- virtual int [adjustColumnsEffCnt](#) ()
- virtual int [compressColumns](#) ()

Helper functions.

- void [initSetup](#) ()
Initial setup of algorithm structures and solver interfaces.
- void [getModelsFromApp](#) ()
- void [createOsiSubProblem](#) ([DecompSubModel](#) &subModel)

- **OsiSolverInterface** * **getOsiLpSolverInterface** ()
- **OsiSolverInterface** * **getOsiIpSolverInterface** ()
- void **coreMatrixAppendColBounds** ()
Calculate gap: $|ub-lb|/|lb|$.
- void **checkMasterDualObj** ()
- bool **checkPointFeasible** (const **DecompConstraintSet** *modelCore, const double *x)
- bool **isDualRayInfProof** (const double *dualRay, const **CoinPackedMatrix** *rowMatrix, const double *colLB, const double *colUB, const double *rowRhs, std::ostream *os)
- bool **isDualRayInfProofCpx** (const double *dualRay, const **CoinPackedMatrix** *rowMatrix, const double *colLB, const double *colUB, const double *rowRhs, std::ostream *os)
- void **printBasisInfo** (**OsiSolverInterface** *si, std::ostream *os)
- void **printCurrentProblemDual** (**OsiSolverInterface** *si, const std::string baseName, const int nodeIndex, const int cutPass, const int pricePass)
- void **printCurrentProblem** (const **OsiSolverInterface** *si, const std::string baseName, const int nodeIndex, const int cutPass, const int pricePass, const int blockId=-1, const bool printMps=true, const bool printLp=true)
- void **printCurrentProblem** (const **OsiSolverInterface** *si, const std::string fileName, const bool printMps=true, const bool printLp=true)
- void **printVars** (std::ostream *os)
- void **printCuts** (std::ostream *os)
- void **checkDuals** ()
- void **checkReducedCost** (const double *u, const double *u_adjusted)
- void **createFullMps** (const std::string fileName)
- virtual **DecompSolverResult** * **solveDirect** (const **DecompSolution** *startSol=NULL)
- void **masterMatrixAddMOCols** (**CoinPackedMatrix** *masterM, double *colLB, double *colUB, double *objCoeff, std::vector< std::string > &colNames)
- void **masterMatrixAddArtCol** (std::vector< CoinBigIndex > &colBeg, std::vector< int > &colInd, std::vector< double > &colVal, char LorG, int rowIndex, int colIndex, DecompColType colType, double &colLB, double &colUB, double &objCoeff)
- virtual void **masterMatrixAddArtCols** (**CoinPackedMatrix** *masterM, double *colLB, double *colUB, double *objCoeff, std::vector< std::string > &colNames, int startRow, int endRow, DecompRowType rowType)
- void **masterPhaseTol** ()
- void **masterPhaseTol** ()
- bool **isMasterColMasterOnly** (const int index) const
- bool **isMasterColStructural** (const int index) const
- bool **isMasterColArtificial** (const int index) const
- void **breakOutPartial** (const double *xHat, DecompVarList &newVars, const double intTol=1.0e-5)
- void **generateVarsAdjustDuals** (const double *uOld, double *uNew)
Create an adjusted dual vector with the duals from the convexity constraints removed.
- void **generateVarsCalcRedCost** (const double *u, double *redCostX)
Calculated reduced cost vector (over vars in compact space) for a given dual vector.

Set/get methods.

- const double * **getColLBNode** () const
- const double * **getColUBNode** () const
- **DecompStats** & **getStats** ()
- const double * **getOrigObjective** () const
- const **DecompSubModel** & **getModelCore** () const
- const int **getAlgo** () const
- const **DecompParam** & **getParam** () const
- **DecompParam** & **getMutableParam** ()
- **OsiSolverInterface** * **getMasterOSI** ()
- **DecompSubModel** & **getModelRelax** (const int blockId)
- const double * **getXhat** () const
Get a ptr to the current solution (in x-space).
- void **setCutoffUB** (const double thisBound)
- const **DecompSolution** * **getXhatIPBest** () const

- const std::vector< [DecompSolution](#) * > & **getXhatIPFeas** () const
- const double **getCutoffUB** () const
- [DecompStats](#) & **getDecompStats** ()
- const [DecompParam](#) & **getDecompParam** () const
- const [DecompApp](#) * **getDecompApp** () const
- [DecompApp](#) * **getDecompAppMutable** ()
- const int **getNodeIndex** () const
- const int **getCutCallsTotal** () const
- const int **getPriceCallsTotal** () const
- const double * **getMasterPrimalSolution** () const
Get current primal solution for master problem.
- const double * **getMasterColReducedCost** () const
- virtual const double * **getMasterDualSolution** () const
Get current dual solution for master problem.
- virtual void **adjustMasterDualSolution** ()
Adjust the current dual solution for master problem.
- double **getMasterObjValue** () const
- const int **getStopCriteria** () const
- const double **getGlobalGap** () const
Get the current global (integrality) gap.
- const double **getNodeIPGap** () const
Get the current node (integrality) gap.
- const double **getNodeLPGap** () const
Get the current node (continuous) gap.
- const double **getObjBestBoundLB** () const
Get the current best LB.
- const void **setStrongBranchIter** (bool isStrongBranch=true)
Set the object to be in strong branching mode.
- const double **getObjBestBoundUB** () const
Get the current best UB.
- const double **getMasterRowType** (int row) const
Get a specific row type.
- virtual void **setObjBound** (const double thisBound, const double thisBoundUB)
Set the current continuous bounds and update best/history.
- virtual void **setObjBoundIP** (const double thisBound)
Set the current integer bound and update best/history.
- bool **isTailoffLB** (const int changeLen=10, const double changePerLimit=0.1)
- int **getNumRowType** ([DecompRowType](#) rowType)
- void **checkBlocksColumns** ()

Constructors and destructor.

- [DecompAlgo](#) (const [DecompAlgoType](#) algo, [DecompApp](#) *app, [UtilParameters](#) &utilParam, bool doSetup=true)
Default constructors.
- virtual [~DecompAlgo](#) ()
Destructor.

Protected Attributes

Data.

- std::string **m_classTag**
Store the name of the class (for logging/debugging) - "who am I?".

- [DecompParam m_param](#)
Parameters.
- [UtilParameters * m_utilParam](#)
- [DecompAlgoType m_algo](#)
Type of algorithm for this instance.
- [DecompStatus m_status](#)
The current algorithm status.
- [DecompPhase m_phase](#)
The current algorithm phase.
- [DecompPhase m_phaseLast](#)
- [DecompPhase m_phaseForce](#)
- [DecompApp * m_app](#)
Pointer to current active DECOMP application.
- [DecompStats m_stats](#)
Storage of statistics for run and node.
- [DecompNodeStats m_nodeStats](#)
- [DecompMemPool m_memPool](#)
Memory pool used to reduce the number of allocations needed.
- [std::ostream * m_osLog](#)
Stream for log file (default to stdout).
- [DecompAlgoCGL * m_cgl](#)
- [std::vector< double > m_origColLB](#)
Pointer (and label) to current active model core/relax.
- [std::vector< double > m_origColUB](#)
- [OsiSolverInterface * m_masterSI](#)
Solver interface(s) for subproblems (P').
- [OsiClpSolverInterface * m_cutgenSI](#)
Solver interface(s) for entire problem (Q").
- [int m_cutgenObjCutInd](#)
- [OsiSolverInterface * m_auxSI](#)
- [const double * m_objective](#)
- [DecompSubModel m_modelCore](#)
- [std::map< int, DecompSubModel > m_modelRelax](#)
- [std::map< int, std::vector< DecompSubModel > > m_modelRelaxNest](#)
- [DecompVarList m_vars](#)
Containers for variables (current and pool).
- [DecompVarPool m_varpool](#)
- [DecompCutList m_cuts](#)
Containers for cuts (current and pool).
- [DecompCutPool m_cutpool](#)
- [double * m_xhat](#)
Storage for current solution (in x-space).
- [double m_cutoffUB](#)
User-defined cutoff (global UB) for B&B fathoming and LR.
- [std::vector< \[DecompSolution\]\(#\) * > m_xhatIPFeas](#)
- [DecompSolution * m_xhatIPBest](#)
- [std::vector< double > m_primSolution](#)
- [std::vector< double > m_dualSolution](#)
- [std::vector< double > m_reducedCost](#)
- [int m_numCols](#)
- [bool m_isColGenExact](#)
- [int m_numConvexCon](#)
- [int m_rrLastBlock](#)
- [int m_rrIterSinceAll](#)
- [int m_nArtCols](#)

- int **m_nRowsOrig**
- int **m_nRowsBranch**
- int **m_nRowsConvex**
- int **m_nRowsCuts**
- std::vector< DecompRowType > **m_masterRowType**
- std::vector< DecompColType > **m_masterColType**
- std::vector< int > **m_masterArtCols**
- double * **m_colLBNode**
- double * **m_colUBNode**
- int **m_compressColsLastPrice**
- int **m_compressColsLastNumCols**
- double **m_relGap**
- Current node gap (bestUB-bestLB)/bestLB.*
- DecompAlgoStop **m_stopCriteria**
- int **m_colIndexUnique**
- double **m_masterObjLast**
- bool **m_objNoChange**
- double **m_stabEpsilon**
- bool **m_useInitLpDuals**
- std::map< int, int > **m_artColIndToRowInd**
- double **m_globalLB**
- double **m_globalUB**
- std::vector< double > **m_phaseObj**
- int **m_function**
- bool **m_firstPhase2Call**
- bool **m_isStrongBranch**
- const [AlpsDecompTreeNode](#) * **m_curNode**
- std::vector< int > **m_masterOnlyCols**
- std::map< int, int > **m_masterOnlyColsMap**
- Map from original index to master index for master-only vars.*
- DecompBranchingImplementation **m_branchingImplementation**

4.11.1 Detailed Description

Base class for DECOMP algorithms.

Definition at line 63 of file [DecompAlgo.h](#).

4.11.2 Member Function Documentation

4.11.2.1 `virtual void DecompAlgo::recomposeSolution (const double * solution, double * rsolution)` [virtual]

Compose solution in x-space from current space.

- PC: this recomposes x from lambda
- C : this just copies over LP solution

Reimplemented in [DecompAlgoPC](#), [DecompAlgoD](#), and [DecompAlgoC](#).

4.11.2.2 `virtual void DecompAlgo::postProcessNode (DecompStatus decompStatus)` [inline],[virtual]

Do some information sending after the current node has been processed.

Does nothing by default.

Reimplemented in [DippyAlgoRC](#), [DippyAlgoPC](#), and [DippyAlgoC](#).

Definition at line 321 of file [DecompAlgo.h](#).

4.11.2.3 `virtual void DecompAlgo::postProcessBranch (DecompStatus decompStatus)` `[inline],[virtual]`

Do some information sending after the current node has been branched.

Does nothing by default.

Reimplemented in [DippyAlgoRC](#), [DippyAlgoPC](#), and [DippyAlgoC](#).

Definition at line 328 of file `DecompAlgo.h`.

4.11.2.4 `virtual int DecompAlgo::generateInitVars (DecompVarList & initVars)` `[virtual]`

Generate initial variables for master problem (PC/DC/RC).

- in CPM, this does nothing

Reimplemented in [DecompAlgoC](#).

4.11.3 Member Data Documentation

4.11.3.1 `OsiSolverInterface * DecompAlgo::m_masterSI` `[protected]`

Solver interface(s) for subproblems (P').

Solver interface(s) for master problem (Q"). CPM: holds model core (and optionally relaxed) in original space PC : holds model core in reformulated space

Definition at line 143 of file `DecompAlgo.h`.

4.11.3.2 `OsiClpSolverInterface* DecompAlgo::m_cutgenSI` `[protected]`

Solver interface(s) for entire problem (Q").

CPM: not used (use m_masterSI) PC : holds model core (and optionally relaxed) in original space - used for CGL cuts

Definition at line 151 of file `DecompAlgo.h`.

4.11.3.3 `double DecompAlgo::m_cutoffUB` `[protected]`

User-defined cutoff (global UB) for B&B fathoming and LR.

This does not imply a feasible IP solution, just a bound.

Definition at line 183 of file `DecompAlgo.h`.

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

- `DecompAlgo.h`
- `DecompAlgo.old.h`

4.12 DecompAlgoC Class Reference

Class for DECOMP algorithm Cutting Plane Method.


```
#include <DecompAlgoC.h>
```

Inheritance diagram for DecompAlgoC:

Collaboration diagram for DecompAlgoC:

Public Member Functions

- void [createMasterProblem](#) (DecompVarList &initVars)
Create the master problem (all algorithms must define this function).
- void [recomposeSolution](#) (const double *solution, double *rsolution)
Compose solution in x-space from current space.
- int [generateInitVars](#) (DecompVarList &initVars)
Generate initial variables for master problem (PC/DC/RC).

Derived from virtual functions of DecompAlgo

- virtual [DecompSolverResult](#) * **solveDirect** (const [DecompSolution](#) *startSol=NULL)

Constructors and destructor.

- [DecompAlgoC](#) ([DecompApp](#) *app, [UtilParameters](#) &utilParam)
Default constructors.
- [~DecompAlgoC](#) ()
Destructor.

Additional Inherited Members

4.12.1 Detailed Description

Class for DECOMP algorithm Cutting Plane Method.

Definition at line 34 of file DecompAlgoC.h.

4.12.2 Member Function Documentation

4.12.2.1 void [DecompAlgoC::recomposeSolution](#) (const double * *solution*, double * *rsolution*) [virtual]

Compose solution in x-space from current space.

- PC: this recomposes x from lambda
- C : this just copies over LP solution

Reimplemented from [DecompAlgo](#).

4.12.2.2 `int DecompAlgoC::generateInitVars (DecompVarList & initVars) [inline],[virtual]`

Generate initial variables for master problem (PC/DC/RC).

- in CPM, this does nothing

Reimplemented from [DecompAlgo](#).

Definition at line 34 of file `old/DecompAlgoC.h`.

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

- `DecompAlgoC.h`

4.13 `DecompAlgoCGL` Class Reference

An interface to CGL cut generator library.

```
#include <DecompAlgoCGL.h>
```

Public Member Functions

Helper functions.

- `int initGenerators` (const int doClique, const int doOddHole, const int doFlowCover, const int doKnapCover, const int doMixIntRound, const int doGomory)
- `int generateCuts` (`OsiSolverInterface` *cutGenSI, `OsiSolverInterface` *masterSI, double *xhat, std::vector<int > &integerVars, `DecompCutList` &newCuts)

Set/get methods.

- void `setLogLevel` (const int logLevel)
- void `setLogStream` (std::ostream *logStream)

Constructors and destructor.

- [DecompAlgoCGL](#) (int logLevel=0, `DecompAlgoType` algo=CUT, std::ostream *logStream=&std::cout)
Default constructors.
- [~DecompAlgoCGL](#) ()
Destructor.

4.13.1 Detailed Description

An interface to CGL cut generator library.

Definition at line 42 of file `DecompAlgoCGL.h`.

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

- `DecompAlgoCGL.h`

4.14 DecompAlgoD Class Reference

Class for DECOMP algorithm Decomp.

```
#include <DecompAlgoD.h>
```

Inheritance diagram for DecompAlgoD:

Collaboration diagram for DecompAlgoD:

Public Member Functions

- void [createMasterProblem](#) (DecompVarList &initVars)
Create the master problem (all algorithms must define this function).
- void [recomposeSolution](#) (const double *solution, double *rsolution)
Compose solution in x-space from current space.

Derived from virtual functions of DecompAlgoPC

- void **solved** (DecompCutList *newCuts)

Constructors and destructor.

- [DecompAlgoD](#) ([DecompApp](#) *app, [UtilParameters](#) &utilParam, double *xhat, int numOrigCols)
Default constructors.
- [~DecompAlgoD](#) ()
Destructor.

Additional Inherited Members

4.14.1 Detailed Description

Class for DECOMP algorithm Decomp.

Definition at line 37 of file DecompAlgoD.h.

4.14.2 Member Function Documentation

4.14.2.1 void [DecompAlgoD::recomposeSolution](#) (const double * *solution*, double * *rsolution*) [virtual]

Compose solution in x-space from current space.

- PC: this recomposes x from lambda
- C : this just copies over LP solution

Reimplemented from [DecompAlgo](#).

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

- [DecompAlgoD.h](#)

4.15 DecompAlgoPC Class Reference

Class for DECOMP algorithm Price and Cut.

```
#include <DecompAlgoPC.h>
```

Inheritance diagram for DecompAlgoPC:

Collaboration diagram for DecompAlgoPC:

Public Member Functions

- virtual void [createMasterProblem](#) (DecompVarList &initVars)
Create the master problem (all algorithms must define this function).
- void [recomposeSolution](#) (const double *solution, double *rsolution)
Compose solution in x-space from current space.

Constructors and destructor.

- std::vector< double > & [getDualBest](#) ()
- std::vector< double > & [getDualRMP](#) ()
- [DecompAlgoPC](#) ([DecompApp](#) *app, [UtilParameters](#) &utilParam, bool doSetup=true, const DecompAlgoType algo=PRICE_AND_CUT)
Default constructors.
- [~DecompAlgoPC](#) ()
Destructor.

Additional Inherited Members

4.15.1 Detailed Description

Class for DECOMP algorithm Price and Cut.

Definition at line 32 of file DecompAlgoPC.h.

4.15.2 Member Function Documentation

4.15.2.1 void [DecompAlgoPC::recomposeSolution](#) (const double * *solution*, double * *rsolution*) [virtual]

Compose solution in x-space from current space.

- PC: this recomposes x from lambda
- C : this just copies over LP solution

Reimplemented from [DecompAlgo](#).

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

- DecompAlgoPC.h

4.16 DecompAlgoRC Class Reference

```
#include <DecompAlgoRC.h>
```

Inheritance diagram for DecompAlgoRC:

Collaboration diagram for DecompAlgoRC:

Public Member Functions

- void [createMasterProblem](#) (DecompVarList &initVars)
Create the master problem (all algorithms must define this function).

Constructors and destructor.

- [DecompAlgoRC](#) ([DecompApp](#) *app, [UtilParameters](#) &utilParam)
Default constructors.
- [~DecompAlgoRC](#) ()
Destructor.

Additional Inherited Members

4.16.1 Detailed Description

Definition at line 25 of file DecompAlgoRC.h.

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

- DecompAlgoRC.h

4.17 DecompApp Class Reference

The main application class.

```
#include <DecompApp.h>
```

Inheritance diagram for DecompApp:

Collaboration diagram for DecompApp:

Public Member Functions

- virtual void [initializeApp](#) ()
Initialize applications.
- void [createModels](#) ()
Create model parts.
- void [readBlockFile](#) ()
Read block file.
- void [readProblem](#) ()
Read Problem.

- void [singlyBorderStructureDetection](#) ()
Automatically detect singly bordered structure.
- void [findActiveColumns](#) (const std::vector< int > &rowsPart, std::set< int > &activeColsSet)
Find the active columns for some block.
- const std::string [getInstanceName](#) ()
Get Instance name.
- const **CoinPackedMatrix** * [getMatrix](#) ()
Get constraint matrix for analysis.
- [DecompApp](#) ([UtilParameters](#) &utilParam)
Constructor for base [DecompApp](#) class.
- virtual [~DecompApp](#) ()
Destructor.
- void [startupLog](#) ()
Initialize the [DecompApp](#) data.

Helper functions.

- void [preprocess](#) ()
Preprocess (standard): on the TODO list.
- void [startupLog](#) ()
Print startup message to log.
- int [createModel](#) ()
- const double [getBestKnownLB](#) () const
- const double [getBestKnownUB](#) () const
- void [setBestKnownLB](#) (const double bestKnownLB)
- void [setBestKnownUB](#) (const double bestKnownUB)
- void [setModelObjective](#) (const double *objective, const int length)
Set the model objective function.
- void [setDecompInf](#) ()
- void [setModelCore](#) ([DecompConstraintSet](#) *model, const std::string modelName)
Set the model core constraint matrix.
- void [setModelRelax](#) ([DecompConstraintSet](#) *model, const std::string modelName="", const int blockId=0)
Set the model relaxed constraint matrix (for a particular block).
- void [setModelRelaxNest](#) ([DecompConstraintSet](#) *model, const std::string modelName, const int blockId=0)
Set the model relaxed (nested) constraint matrix (for a particular block).
- [DecompAlgo](#) * [getDecompAlgo](#) () const
Get a pointer to the base algorithm class.

Interface methods for user derivation (virtual).

- virtual void [initDualVector](#) (std::vector< double > &dualVector)
Initialize the dual vector for PhaseII of PC.
- virtual bool [APPisUserFeasible](#) (const double *x, const int numCols, const double tolZero)
Method to determine if the solution (x) is feasible to the original model.
- virtual int [APPheuristics](#) (const double *xhat, const double *origCost, std::vector< [DecompSolution](#) * > &xhatIPFeas)
- virtual const double * [getDualForGenerateVars](#) (const double *dual)
This function allows the user to return their own dual vector to be used in the generation of new variables (in the reduced-cost calculation).
- virtual int [generateInitVars](#) ([DecompVarList](#) &initVars)
- virtual int [generateCuts](#) (const double *x, [DecompCutList](#) &newCuts)
- virtual void [solveRelaxedWhich](#) (std::vector< int > &blocksToSolve, std::map< int, std::vector< double > > &userDualsByBlock)

- virtual DecompSolverStatus **solveRelaxed** (const int whichBlock, const double *redCostX, const double target, DecompVarList &varList)
- virtual DecompSolverStatus **solveRelaxedNest** (const int whichBlock, const double *redCostX, const double target, DecompVarList &varList)
- virtual void **printOriginalColumn** (const int index, std::ostream *os=&std::cout) const
- virtual void **printOriginalSolution** (const int n_cols, const std::vector< std::string > &colNames, const double *solution, std::ostream *os=&std::cout) const

Public Attributes

- int [NumBlocks](#)
Number of Blocks default value 0 set by BlockNumInput parameter.
- [DecompParam](#) [m_param](#)
Parameters.
- const double * [m_objective](#)
Model data: objective function.
- [DecompModel](#) [m_modelCore](#)
Model data: the core model (A')
- std::map< int, [DecompModel](#) > [m_modelRelax](#)
Model data: the relaxed model(s) (A')
- std::map< int, std::vector< [DecompModel](#) > > [m_modelRelaxNest](#)
Model data: the relaxed (nested) model(s) (A')
- [DecompAlgo](#) * [m_decompAlgo](#)
Pointer to the base algorithmic object.
- [CoinMpsIO](#) [m_mpsIO](#)
MPS object for reading instances.
- [CoinLpIO](#) [m_lpIO](#)
LP object for reading instances.
- const [CoinPackedMatrix](#) * [m_matrix](#)
Original constraint matrix for the instance.
- [DecompConstraintSet](#) * [m_modelC](#)
The model constraint systems used for different algos.
- std::map< int, std::vector< int > > [m_blocks](#)
Definition of blocks (by rows)
- int [m_threadIndex](#)
serves as an index to track different [DecompApp](#) object during Concurrent process, where when [m_threadIndex](#) is 0, problem is solved by cutting plane from standalone solver, when it is greater than 0, it is solved by branch-and-price,
- [DecompModel](#) [m_model](#)
Model data object.

Protected Attributes

- std::ostream * [m_osLog](#)
Log file.
- double [m_bestKnownLB](#)
The best known LB/UB for this application (if known, for debugging).
- ostream * [m_osLog](#)
Log file.

4.17.1 Detailed Description

The main application class.

The main application class where the user will define the model decomposition and define any application specific methods.

The main application class where the user will define the model decomposition and define any application specific methods.

See also

[DecompModel](#) [DecompConstraintSet](#) [DecompParam](#)

Definition at line 50 of file `DecompApp.h`.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 `DecompApp::DecompApp (UtilParameters & utilParam)` `[inline]`

Constructor for base [DecompApp](#) class.

This accepts a generic parameters object ([UtilParameters](#)) and reads in the parameter settings into the [DecompApp](#) paramter object.

Definition at line 467 of file `DecompApp.h`.

4.17.3 Member Function Documentation

4.17.3.1 `void DecompApp::setModelObjective (const double * objective, const int length)` `[inline]`

Set the model objective function.

NOTE: The user application MUST call this method.

Definition at line 187 of file `DecompApp.h`.

4.17.3.2 `void DecompApp::setModelCore (DecompConstraintSet * model, const std::string modelName)` `[inline]`

Set the model core constraint matrix.

NOTE: The user application MUST call this method.

Definition at line 205 of file `DecompApp.h`.

4.17.3.3 `void DecompApp::setModelRelax (DecompConstraintSet * model, const std::string modelName = " ", const int blockId = 0)` `[inline]`

Set the model relaxed constraint matrix (for a particular block).

NOTE: The user application MUST call this method IF they are not deriving the function `DecompApp::solveRelaxed`.

Definition at line 241 of file `DecompApp.h`.

4.17.3.4 `virtual void DecompApp::initDualVector (std::vector< double > & dualVector) [inline],[virtual]`

Initialize the dual vector for Phasell of PC.

The user is passed a reference to the internal data and can manipulate it directly.

This is only called when dual stabilization is used, i.e., when `m_param.DualStab > 0`, at the first iteration of Phasell of PC. The vector is immediately smoothed with the initial restricted master duals. By default, the restricted mater is used as the initial dual and, therefore, no smoothing occurs in the first iteration.

Definition at line 311 of file `DecompApp.h`.

4.17.3.5 `virtual bool DecompApp::APPisUserFeasible (const double * x, const int numCols, const double tolZero) [inline],[virtual]`

Method to determine if the solution (x) is feasible to the original model.

For explicitly defined model components, like the model core constraints (A"), the feasibility of the solution is automatically checked against the constraints. In the case when the relaxed problem constraints (A') are explicitly defined - these are also checked automatically.

However, for some applications, a valid feasible constraint system cannot be explicitly defined (even for the core set of constraints). For example, think of the case of TSP, where A" is defined as the subtour elimination constraints. These constraints are implicitly defined by deriving the method `DecompApp::generateCuts`. Therefore, the framework cannot automatically tell if a solution is feasible by checking against the constraint system. In this case, the user must provide this method.

Parameters

in	<i>x</i>	The solution point to check.
in	<i>numCols</i>	The number of variables.
in	<i>tolZero</i>	The integrality tolerance (currently ignored).

Returns

True, if x is feasible; otherwise, false.

Reimplemented in [DippyDecompApp](#).

Definition at line 339 of file `DecompApp.h`.

4.17.3.6 `virtual const double* DecompApp::getDualForGenerateVars (const double * dual) [inline],[virtual]`

This function allows the user to return their own dual vector to be used in the generation of new variables (in the reduced-cost calculation).

For reference, the user is given the dual vector from the restricted master (or the stabilized dual, if using `m_param.DualStab`).

Definition at line 359 of file `DecompApp.h`.

4.17.4 Member Data Documentation

4.17.4.1 `DecompAlgo* DecompApp::m_decompAlgo`

Pointer to the base algorithmic object.

NOTE: only for the advanced user

Definition at line 108 of file DecompApp.h.

4.17.4.2 int DecompApp::m_threadIndex

serves as an index to track different [DecompApp](#) object during Concurrent process, where when m_threadIndex is 0, problem is solved by cutting plane from standalone solver, when it is greater than 0, it is solved by branch-and-price,

Definition at line 142 of file DecompApp.h.

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

- DecompApp.h

4.18 DecompConstraintSet Class Reference

Collaboration diagram for DecompConstraintSet:

4.18.1 Detailed Description

Definition at line 31 of file DecompConstraintSet.h.

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

- DecompConstraintSet.h

4.19 DecompCut Class Reference

Inheritance diagram for DecompCut:

Collaboration diagram for DecompCut:

Public Member Functions

- void [increaseEffCnt](#) ()
Increase the effectiveness count by 1 (or to 1 if it was negative).
- void [decreaseEffCnt](#) ()
Decrease the effectiveness count by 1 (or to -1 if it was positive).
- void [increaseEffCnt](#) ()
Increase the effectiveness count by 1 (or to 1 if it was negative).
- void [decreaseEffCnt](#) ()
Decrease the effectiveness count by 1 (or to -1 if it was positive).

4.19.1 Detailed Description

Definition at line 35 of file DecompCut.h.

4.19.2 Member Function Documentation

4.19.2.1 void DecompCut::increaseEffCnt () [inline]

Increase the effectiveness count by 1 (or to 1 if it was negative).

Return the new effectiveness count.

Definition at line 125 of file DecompCut.h.

4.19.2.2 void DecompCut::decreaseEffCnt () [inline]

Decrease the effectiveness count by 1 (or to -1 if it was positive).

Return the new effectiveness count.

Definition at line 131 of file DecompCut.h.

4.19.2.3 void DecompCut::increaseEffCnt () [inline]

Increase the effectiveness count by 1 (or to 1 if it was negative).

Return the new effectiveness count.

Definition at line 129 of file old/DecompCut.h.

4.19.2.4 void DecompCut::decreaseEffCnt () [inline]

Decrease the effectiveness count by 1 (or to -1 if it was positive).

Return the new effectiveness count.

Definition at line 135 of file old/DecompCut.h.

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

- DecompCut.h

4.20 DecompCutOsi Class Reference

Inheritance diagram for DecompCutOsi:

Collaboration diagram for DecompCutOsi:

Additional Inherited Members

4.20.1 Detailed Description

Definition at line 34 of file DecompCutOsi.h.

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

- DecompCutOsi.h

4.21 DecompCutPool Class Reference

Inheritance diagram for DecompCutPool:

Collaboration diagram for DecompCutPool:

4.21.1 Detailed Description

Definition at line 39 of file DecompCutPool.h.

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

- DecompCutPool.h

4.22 DecompMainParam Struct Reference

4.22.1 Detailed Description

Definition at line 113 of file Decomp.h.

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

- Decomp.h

4.23 DecompMemPool Class Reference

4.23.1 Detailed Description

Definition at line 23 of file DecompMemPool.h.

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

- DecompMemPool.h

4.24 DecompModel Class Reference

Inheritance diagram for DecompModel:

Collaboration diagram for DecompModel:

Public Attributes

- double * [objCoeff](#)
Model data objects (must be defined by users).
- DecompVarList [vars](#)
Model data objects will be used during algos.

4.24.1 Detailed Description

Definition at line 28 of file DecompModel.h.

4.24.2 Member Data Documentation

4.24.2.1 DecompVarList DecompModel::vars

Model data objects will be used during algos.

THINK: belong here or in algos?

Definition at line 51 of file old/DecompModel.h.

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

- DecompModel.h

4.25 DecompNodeStats Class Reference

Collaboration diagram for DecompNodeStats:

Public Attributes

- `std::vector< DecompObjBound > objHistoryBound`
Storage of the bounds.
- `std::pair< double, double > objBest`
The global lower (.first) and upper (.second) bound.
- `int nodeIndex`
The node index (in the branch-and-bound tree).
- `int cutsThisRound`
Number of cuts generated in this round of cut calls.
- `int varsThisRound`
Number of vars generated in this round of pricing calls.
- `int cutsThisCall`
Number of cuts generated in this particular cut call.
- `int varsThisCall`
Number of vars generated in this particular price call.
- `int cutCallsTotal`
Number of cut calls in this node in total.
- `int priceCallsTotal`
Number of price calls in this node in total.
- `int cutCallsRound`
Number of cut calls in this round.
- `int priceCallsRound`
Number of price calls in this round.

4.25.1 Detailed Description

Definition at line 94 of file DecompStats.h.

4.25.2 Member Data Documentation

4.25.2.1 `std::vector< DecompObjBound > DecompNodeStats::objHistoryBound`

Storage of the bounds.

For the continuous part: CPM : Bounds on the objective of optimal master linear relaxation. Typically, this is an LP solved to optimality, so, $LB = zCP = UB$. PC/RC: Given bounds on the objective of optimal restricted master linear relaxation $zPC_LB \leq zPC^* \leq zPC_UB$ and a lower bound on the most negative reduced cost (RC_LB) extreme point (ray) from the subproblem polytope (for the associated master duals). $LB = zPC_LB + RC_LB \leq zPC^* \leq zPC_UB = UB$

Definition at line 115 of file DecompStats.h.

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

- DecompStats.h

4.26 DecompObjBound Class Reference

Public Member Functions

- `bool operator< (const DecompObjBound &objBound) const`
Comparison operator for sorting on time.

Public Attributes

- `int phase`
The phase when bound was recorded.
- `int cutPass`
The cut pass when bound was recorded.
- `int pricePass`
The price pass when bound was recorded.
- `double timeStamp`
The time stamp (from start) when bound was recorded.
- `double thisBound`
The recorded continuous lower bound.
- `double thisBoundUB`
The recorded continuous upper bound.
- `double bestBound`
The best recorded continuous lower bound.
- `double thisBoundIP`
The recorded integer upper bound.
- `double bestBoundIP`
The best recorded integer upper bound.

4.26.1 Detailed Description

Definition at line 26 of file DecompStats.h.

4.26.2 Member Data Documentation

4.26.2.1 `double DecompObjBound::bestBound`

The best recorded continuous lower bound.

global LB = max{active node lower bounds}

Definition at line 56 of file DecompStats.h.

4.26.2.2 `double DecompObjBound::bestBoundIP`

The best recorded integer upper bound.

global UB = min{node integer upper bounds}

Definition at line 65 of file DecompStats.h.

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

- DecompStats.h

4.27 DecompParam Class Reference

Collaboration diagram for DecompParam:

Public Member Functions

Helper functions.

- void [getSettingsImpl](#) ([UtilParameters](#) ¶m, const char *sec)
- void [getSettings](#) ([UtilParameters](#) ¶m)
- void [getSettings](#) ([UtilParameters](#) ¶m, const std::string &sec)
- void [dumpSettings](#) (const std::string &sec, std::ostream *os=&std::cout)
- void [setDefault](#) ()
- void [dumpSettings](#) (std::ostream *os=&std::cout)

Constructors and destructor.

- [DecompParam](#) ()
Default constructors.
- [~DecompParam](#) ()
Destructor.

Public Attributes

Data.

- int **LogLevel**
- int **LogDebugLevel**
- int **LogLpLevel**
- int **LogIpLevel**
- int **LogDumpModel**
- int **LogObjHistory**
0: print nothing 1: print the node objective history
- int **InitVarsLimit**
- int **DebugLevel**
- double **TolZero**
- int **TotalCutltersLimit**
- int **TotalPriceltersLimit**
- int **RoundCutltersLimit**
- int **RoundPriceltersLimit**
- double **TimeLimit**
- int **NodeLimit**
Max number of nodes (copied from Alps parameters)
- int **TailoffLength**
- double **TailoffPercent**
- double **MasterGapLimit**
- int **PCStrategy**
- int **CompressColumns**
- int **CompressColumnsIterFreq**
- double **CompressColumnsSizeMultLimit**
- double **CompressColumnsMasterGapStart**
- int **CutDC**
- int **CutCGL**
- int **CutCglKnapC**
- int **CutCglFlowC**
- int **CutCglMir**
- int **CutCglClique**
- int **CutCglOddHole**
- int **CutCglGomory**
- int **SubProbUseCutoff**
- double **SubProbGapLimitExact**
- double **SubProbGapLimitInexact**
- double **SubProbTimeLimitExact**
- double **SubProbTimeLimitInexact**
- int **NumConcurrentThreadsSubProb**
- int **NumThreadsIPSolver**
- int **SubProbNumSolLimit**
- int **SubProbSolverStartAlgo**
- int **RoundRobinInterval**
- int **RoundRobinStrategy**
- int **SolveMasterAsMip**
- int **SolveMasterAsMipFreqNode**
- int **SolveMasterAsMipFreqPass**
- double **SolveMasterAsMipTimeLimit**
- double **SolveMasterAsMipLimitGap**
- int **SolveMasterUpdateAlgo**
- int **SolveRelaxAslp**
- int **InitVarsWithCutDC**
- int **InitVarsWithIP**
- int **InitVarsWithIPTimeLimit**
- int **InitCompactSolve**
- bool **DualStab**
- double **DualStabAlpha**
- double **DualStabAlphaOrig**
- bool **BreakOutPartial**

- bool **BranchEnforceInSubProb**
- bool **BranchEnforceInMaster**
- int **MasterConvexityLessThan**
- double **ParallelColsLimit**
- int **BranchStrongIter**
Number of iterations to process in estimating bounds during strong branching.
- int **DebugCheckBlocksColumns**
Number of threads to use in DIP.
- std::string **DataDir**
- std::string **Instance**
- std::string **InstanceFormat**
- std::string **BlockFile**
- std::string **BlockFileFormat**
The format of BlockFile.
- std::string **PermuteFile**
- std::string **InitSolutionFile**
- int **UseNames**
- int **UseSparse**
- int **FullModel**
- double **BestKnownLB**
- double **BestKnownUB**
- double **ColumnUB**
- double **ColumnLB**
- int **ObjectiveSense**
- bool **Concurrent**
- int **NumBlocksCand**
- double **ConcurrentCutOffTime**
- std::string **CurrentWorkingDir**
- bool **SubProbParallel**
- int **SubProbParallelType**
- int **SubProbParallelChunksize**
- int **ConcurrentThreadsNum**
- int **BlockNumInput**
- bool **BlockFileOutput**
- double **RedCostEpsilon**
- double **PhaseObjTol**
- bool **CheckSpecialStructure**
- int **BlockFileOutputFormat**
- bool **SolutionOutputToFile**
- std::string **SolutionOutputFileName**
- bool **WarmStart**
- std::string **DecompLPSolver**
- std::string **DecompIPSolver**
- bool **UseMultiRay**
- bool **DoInteriorPoint**

4.27.1 Detailed Description

Definition at line 28 of file DecompParam.h.

4.27.2 Member Function Documentation

4.27.2.1 void DecompParam::getSettingsImpl (UtilParameters & param, const char * sec) [inline]

Definition at line 345 of file DecompParam.h.

4.27.2.2 `void DecompParam::dumpSettings (const std::string & sec, std::ostream * os = &std::cout) [inline]`

Definition at line 474 of file DecompParam.h.

4.27.3 Member Data Documentation

4.27.3.1 `int DecompParam::BranchStrongIter`

Number of iterations to process in estimating bounds during strong branching.

CPM: this is simplex iterations of master PC : this is outer price and cut iterations sets TotalCutItersLimit=TotalPrice↔ItersLimit=BranchStrongIter THINK: or CPM could be cut passes... and solve master fully? which is expensive and clearly not standard strong branching

Definition at line 207 of file DecompParam.h.

4.27.3.2 `int DecompParam::DebugCheckBlocksColumns`

Number of threads to use in DIP.

Currently, only used for solving the pricing problem for block angular models. The subproblems (each block) are independent and can be solved in parallel.

Definition at line 222 of file DecompParam.h.

4.27.3.3 `std::string DecompParam::BlockFileFormat`

The format of BlockFile.

(1) "List" or "LIST" The block file defines those rows in each block. [block id] [num rows in block] [row ids...] [block id] [num rows in block] [row ids...]

(2) "ZIBList" or "ZIBLIST" The block file defines those rows in each block. NBLOCKS [numBlocks] BLOCK [id of block] [row names...] BLOCK [id of block] [row names...]

(3) "Pair" or "PAIR" Each line is a block id to row id pair. [id of block] [row id]

(4) "PairName" or "PAIRNAME" Each line is a block id to row name (matching mps) pair. [id of block] [row name]

Definition at line 266 of file DecompParam.h.

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

- DecompParam.h

4.28 DecompSolution Class Reference

Public Member Functions

Helper functions (public).

- `const int getSize () const`
Get length of solution.
- `const double * getValues () const`
Get solution values.

- const double [getQuality](#) () const
Get quality of solution.
- virtual void [print](#) (int precision=4, std::ostream &os=std::cout) const
Print solution.
- virtual void [print](#) (const std::vector< std::string > &colNames, int precision=2, std::ostream &os=std::cout) const
Print solution in MIPLIB2010 solution checker format.
- const int [getSize](#) () const
Get length of solution.
- const double * [getValues](#) () const
Get solution values.
- const double [getQuality](#) () const
Get quality of solution.
- virtual void [print](#) (ostream &os=cout) const

Copy Constructors

- **DecompSolution** (const [DecompSolution](#) &source)
- [DecompSolution](#) & **operator=** (const [DecompSolution](#) &rhs)
- **DecompSolution** (const [DecompSolution](#) &source)
- [DecompSolution](#) & **operator=** (const [DecompSolution](#) &rhs)

Constructor and Destructor

- [DecompSolution](#) ()
Default constructor.
- [DecompSolution](#) (const int size, const double *values, const double quality)
Constructor.
- **DecompSolution** (const int size, const double *values, const double *cost)
- virtual ~**DecompSolution** ()
- [DecompSolution](#) ()
Default constructor.
- [DecompSolution](#) (const int size, const double *values, const double quality)
Constructor.
- virtual ~**DecompSolution** ()

Protected Attributes

- int [m_size](#)
Length of solution (number of columns).
- double * [m_values](#)
Solution values.
- double [m_quality](#)
Quality of solution (bound wrt to objective).

4.28.1 Detailed Description

Definition at line 20 of file DecompSolution.h.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 `DecompSolution::DecompSolution ()` `[inline]`

Default constructor.

Takes size of solution.

Definition at line 119 of file `DecompSolution.h`.

4.28.2.2 `DecompSolution::DecompSolution (const int size, const double * values, const double quality)` `[inline]`

Constructor.

Definition at line 126 of file `DecompSolution.h`.

4.28.2.3 `DecompSolution::DecompSolution ()` `[inline]`

Default constructor.

Takes size of solution.

Definition at line 88 of file `old/DecompSolution.h`.

4.28.2.4 `DecompSolution::DecompSolution (const int size, const double * values, const double quality)` `[inline]`

Constructor.

Definition at line 95 of file `old/DecompSolution.h`.

4.28.3 Member Function Documentation

4.28.3.1 `const int DecompSolution::getSize () const` `[inline]`

Get length of solution.

Definition at line 35 of file `DecompSolution.h`.

4.28.3.2 `const double* DecompSolution::getValues () const` `[inline]`

Get solution values.

Definition at line 40 of file `DecompSolution.h`.

4.28.3.3 `const double DecompSolution::getQuality () const` `[inline]`

Get quality of solution.

Definition at line 45 of file `DecompSolution.h`.

4.28.3.4 `virtual void DecompSolution::print (int precision = 4, std::ostream & os = std::cout) const` `[inline]`,
`[virtual]`

Print solution.

Definition at line 51 of file DecompSolution.h.

4.28.3.5 `virtual void DecompSolution::print (const std::vector< std::string > & colNames, int precision = 2, std::ostream & os = std::cout) const` `[inline]`,`[virtual]`

Print solution in MIPLIB2010 solution checker format.

Definition at line 73 of file DecompSolution.h.

4.28.3.6 `const int DecompSolution::getSize () const` `[inline]`

Get length of solution.

Definition at line 31 of file old/DecompSolution.h.

4.28.3.7 `const double* DecompSolution::getValues () const` `[inline]`

Get solution values.

Definition at line 36 of file old/DecompSolution.h.

4.28.3.8 `const double DecompSolution::getQuality () const` `[inline]`

Get quality of solution.

Definition at line 41 of file old/DecompSolution.h.

4.28.4 Member Data Documentation

4.28.4.1 `int DecompSolution::m_size` `[protected]`

Length of solution (number of columns).

Definition at line 23 of file DecompSolution.h.

4.28.4.2 `double * DecompSolution::m_values` `[protected]`

Solution values.

Definition at line 26 of file DecompSolution.h.

4.28.4.3 `double DecompSolution::m_quality` `[protected]`

Quality of solution (bound wrt to objective).

Definition at line 29 of file DecompSolution.h.

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

- [DecompSolution.h](#)

4.29 DecompSolverResult Class Reference

Storage of solver result.

```
#include <DecompSolverResult.h>
```

Collaboration diagram for DecompSolverResult:

Public Member Functions

- [DecompSolverResult \(\)](#)
Default constructors.
- [~DecompSolverResult \(\)](#)
Destructor.

Public Attributes

Data.

- int **m_solStatus**
- int **m_solStatus2**
- double **m_objLB**
- double **m_objUB**
- bool **m_isOptimal**
- bool **m_isUnbounded**
- bool **m_isCutoff**
- int **m_nSolutions**
- `std::vector< std::vector< double > >` **m_solution**

4.29.1 Detailed Description

Storage of solver result.

Definition at line 34 of file [DecompSolverResult.h](#).

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

- [DecompSolverResult.h](#)

4.30 DecompStats Class Reference

Collaboration diagram for DecompStats:

4.30.1 Detailed Description

Definition at line 226 of file [DecompStats.h](#).

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

- [DecompStats.h](#)

4.31 DecompSubModel Class Reference

Inheritance diagram for DecompSubModel:

Collaboration diagram for DecompSubModel:

Additional Inherited Members

4.31.1 Detailed Description

Definition at line 92 of file DecompModel.h.

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

- DecompModel.h

4.32 DecompVar Class Reference

Collaboration diagram for DecompVar:

Public Member Functions

- void [increaseEffCnt](#) ()
Increase the effectiveness count by 1 (or to 1 if it was negative).
- void [decreaseEffCnt](#) ()
Decrease the effectiveness count by 1 (or to -1 if it was positive).

Copy Constructors

- **DecompVar** (const [DecompVar](#) &source)
- [DecompVar](#) & **operator=** (const [DecompVar](#) &rhs)
- **DecompVar** ()
- **DecompVar** (const std::vector< int > &ind, const double els, const double redCost, const double origCost, const DecompVarType varType)
- **DecompVar** (const std::vector< int > &ind, const std::vector< double > &els, const double redCost, const double origCost)
- **DecompVar** (const std::vector< int > &ind, const std::vector< double > &els, const double redCost, const double origCost, const DecompVarType varType)
- **DecompVar** (const int len, const int *ind, const double *els, const double origCost)
- **DecompVar** (const int len, const int *ind, const double *els, const double origCost, const DecompVarType varType)
- **DecompVar** (const int len, const int *ind, const double *els, const double origCost)
- **DecompVar** (const int len, const int *ind, const double *els, const double origCost, const DecompVarType varType)
- **DecompVar** (const int len, const int *ind, const double *els, const double redCost, const double origCost)
- **DecompVar** (const int len, const int *ind, const double *els, const double redCost, const double origCost, const DecompVarType varType)
- **DecompVar** (const int denseLen, const double *denseArray, const double redCost, const double origCost, const DecompVarType varType)
- virtual ~**DecompVar** ()

4.32.1 Detailed Description

Definition at line 30 of file DecompVar.h.

4.32.2 Member Function Documentation

4.32.2.1 void DecompVar::increaseEffCnt () [inline]

Increase the effectiveness count by 1 (or to 1 if it was negative).

Return the new effectiveness count.

Definition at line 101 of file DecompVar.h.

4.32.2.2 void DecompVar::decreaseEffCnt () [inline]

Decrease the effectiveness count by 1 (or to -1 if it was positive).

Return the new effectiveness count.

Definition at line 107 of file DecompVar.h.

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

- DecompVar.h

4.33 DecompVarPool Class Reference

Inheritance diagram for DecompVarPool:

Collaboration diagram for DecompVarPool:

4.33.1 Detailed Description

Definition at line 35 of file DecompVarPool.h.

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

- DecompVarPool.h

4.34 DecompWaitingCol Class Reference

4.34.1 Detailed Description

Definition at line 24 of file DecompWaitingCol.h.

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

- DecompWaitingCol.h

4.35 DecompWaitingRow Class Reference

4.35.1 Detailed Description

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

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

- `DecompWaitingRow.h`

4.36 DippyAlgoC Class Reference

Python-enabled [DecompAlgoC](#).

```
#include <DippyDecompAlgo.h>
```

Inheritance diagram for `DippyAlgoC`:

Collaboration diagram for `DippyAlgoC`:

Public Member Functions

- virtual void [postProcessBranch](#) (`DecompStatus decompStatus`)
Do some information sending after the current node has been branched.
- virtual void [postProcessNode](#) (`DecompStatus decompStatus`)
Do some information sending after the current node has been processed.

Additional Inherited Members

4.36.1 Detailed Description

Python-enabled [DecompAlgoC](#).

Definition at line 73 of file `DippyDecompAlgo.h`.

4.36.2 Member Function Documentation

4.36.2.1 virtual void `DippyAlgoC::postProcessBranch (DecompStatus decompStatus)` `[inline]`, `[virtual]`

Do some information sending after the current node has been branched.

Does nothing by default.

Reimplemented from [DecompAlgo](#).

Definition at line 89 of file `DippyDecompAlgo.h`.

4.36.2.2 virtual void `DippyAlgoC::postProcessNode (DecompStatus decompStatus)` `[inline]`, `[virtual]`

Do some information sending after the current node has been processed.

Does nothing by default.

Reimplemented from [DecompAlgo](#).

Definition at line 93 of file DippyDecompAlgo.h.

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

- DippyDecompAlgo.h

4.37 DippyAlgoMixin Class Reference

Mixin class for Dip Algorithms.

```
#include <DippyDecompAlgo.h>
```

Inheritance diagram for DippyAlgoMixin:

Collaboration diagram for DippyAlgoMixin:

Public Member Functions

- [DippyAlgoMixin](#) ([UtilParameters](#) &utilParam, PyObject *pProb)
Constructor.

4.37.1 Detailed Description

Mixin class for Dip Algorithms.

This is a helper class for interfacing Dip Algo classes with Python. To add Python support to a standard [DecompAlgo](#), create a subclass which also inherits from [DippyAlgoMixin](#) and override the virtual methods to call those provided by the Mixin class. See [DippyAlgoC](#) for an example.

Definition at line 22 of file DippyDecompAlgo.h.

4.37.2 Constructor & Destructor Documentation

4.37.2.1 [DippyAlgoMixin::DippyAlgoMixin](#) ([UtilParameters](#) & *utilParam*, PyObject * *pProb*) `[inline]`

Constructor.

Parameters

<i>utilParam</i>	parameter class
<i>pProb</i>	a DipProblem python object

Definition at line 39 of file DippyDecompAlgo.h.

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

- DippyDecompAlgo.h

4.38 DippyAlgoPC Class Reference

Python-enabled [DecompAlgoPC](#).

```
#include <DippyDecompAlgo.h>
```

Inheritance diagram for DippyAlgoPC:

Collaboration diagram for DippyAlgoPC:

Public Member Functions

- virtual void [postProcessBranch](#) (DecompStatus decompStatus)
Do some information sending after the current node has been branched.
- virtual void [postProcessNode](#) (DecompStatus decompStatus)
Do some information sending after the current node has been processed.

Additional Inherited Members

4.38.1 Detailed Description

Python-enabled [DecompAlgoPC](#).

Definition at line 103 of file DippyDecompAlgo.h.

4.38.2 Member Function Documentation

4.38.2.1 virtual void DippyAlgoPC::postProcessBranch (DecompStatus *decompStatus*) [inline],[virtual]

Do some information sending after the current node has been branched.

Does nothing by default.

Reimplemented from [DecompAlgo](#).

Definition at line 117 of file DippyDecompAlgo.h.

4.38.2.2 virtual void DippyAlgoPC::postProcessNode (DecompStatus *decompStatus*) [inline],[virtual]

Do some information sending after the current node has been processed.

Does nothing by default.

Reimplemented from [DecompAlgo](#).

Definition at line 121 of file DippyDecompAlgo.h.

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

- DippyDecompAlgo.h

4.39 DippyAlgoRC Class Reference

Python-enabled [DecompAlgoRC](#).

```
#include <DippyDecompAlgo.h>
```

Inheritance diagram for DippyAlgoRC:

Collaboration diagram for DippyAlgoRC:

Public Member Functions

- virtual void [postProcessBranch](#) (DecompStatus decompStatus)
Do some information sending after the current node has been branched.
- virtual void [postProcessNode](#) (DecompStatus decompStatus)
Do some information sending after the current node has been processed.

Additional Inherited Members

4.39.1 Detailed Description

Python-enabled [DecompAlgoRC](#).

Definition at line 131 of file DippyDecompAlgo.h.

4.39.2 Member Function Documentation

4.39.2.1 virtual void DippyAlgoRC::postProcessBranch (DecompStatus *decompStatus*) [inline],[virtual]

Do some information sending after the current node has been branched.

Does nothing by default.

Reimplemented from [DecompAlgo](#).

Definition at line 145 of file DippyDecompAlgo.h.

4.39.2.2 virtual void DippyAlgoRC::postProcessNode (DecompStatus *decompStatus*) [inline],[virtual]

Do some information sending after the current node has been processed.

Does nothing by default.

Reimplemented from [DecompAlgo](#).

Definition at line 149 of file DippyDecompAlgo.h.

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

- DippyDecompAlgo.h

4.40 DippyDecompApp Class Reference

A [DecompApp](#) that links Python to DIP.

```
#include <DippyDecompApp.h>
```

Inheritance diagram for DippyDecompApp:

Collaboration diagram for DippyDecompApp:

Helper functions (public).

- PyObject * [m_rowList](#)

- `map< PyObject *, int > m_rowIndices`
- `PyObject * m_colList`
- `map< PyObject *, int > m_colIndices`
- `PyObject * m_relaxedKeys`
- `map< PyObject *, int > m_relaxIndices`
- `void addPuLPProb (PyObject *p)`
- `void createModels ()`
- `virtual DecompSolverStatus solveRelaxed (const int whichBlock, const double *redCostX, const double convexDual, DecompVarList &varList)`
- `bool APPisUserFeasible (const double *x, const int n_cols, const double tolZero)`
Method to determine if the solution (x) is feasible to the original model.
- `virtual int generateCuts (const double *x, DecompCutList &newCuts)`
- `int APPheuristics (const double *xhat, const double *origCost, vector< DecompSolution * > &xhatIPFeas)`
- `int generateInitVars (DecompVarList &initVars)`
- `DippyDecompApp (UtilParameters &utilParam, PyObject *p)`
- `virtual ~DippyDecompApp ()`

Additional Inherited Members

4.40.1 Detailed Description

A [DecompApp](#) that links Python to DIP.

See also

[DecompApp](#)

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

4.40.2 Member Function Documentation

4.40.2.1 `bool DippyDecompApp::APPisUserFeasible (const double * x, const int numCols, const double tolZero) [virtual]`

Method to determine if the solution (x) is feasible to the original model.

For explicitly defined model components, like the model core constraints (A"), the feasibility of the solution is automatically checked against the constraints. In the case when the relaxed problem constraints (A') are explicitly defined - these are also checked automatically.

However, for some applications, a valid feasible constraint system cannot be explicitly defined (even for the core set of constraints). For example, think of the case of TSP, where A" is defined as the subtour elimination constraints. These constraints are implicitly defined by deriving the method `DecompApp::generateCuts`. Therefore, the framework cannot automatically tell if a solution is feasible by checking against the constraint system. In this case, the user must provide this method.

Parameters

<code>in</code>	<code>x</code>	The solution point to check.
-----------------	----------------	------------------------------

<code>in</code>	<code>numCols</code>	The number of variables.
<code>in</code>	<code>tolZero</code>	The integrality tolerance (currently ignored).

Returns

True, if x is feasible; otherwise, false.

Reimplemented from [DecompApp](#).

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

- DippyDecompApp.h

4.41 DippyDecompCut Class Reference

Inheritance diagram for DippyDecompCut:

Collaboration diagram for DippyDecompCut:

Additional Inherited Members

4.41.1 Detailed Description

Definition at line 11 of file DippyDecompCut.h.

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

- DippyDecompCut.h

4.42 is_greater_thanD Class Reference

4.42.1 Detailed Description

Definition at line 28 of file DecompCutPool.h.

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

- DecompCutPool.h

4.43 is_less_thanD Class Reference

4.43.1 Detailed Description

Definition at line 26 of file DecompVarPool.h.

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

- DecompVarPool.h

4.44 OsiData Class Reference

Class collecting pointers on data for OsiEmpty.

```
#include <OsiData.hpp>
```

Collaboration diagram for OsiData:

Public Member Functions

- void [convertBoundToSense](#) (const double lower, const double upper, char &sense, double &right, double &range) const
A quick inlined function to convert from the lb/ub style of constraint definition to the sense/rhs/range style.
- void [convertBoundToSense](#) (const double lower, const double upper, char &sense, double &right, double &range) const
A quick inlined function to convert from the lb/ub style of constraint definition to the sense/rhs/range style.

Public Set/get methods

- virtual void [setInfinity](#) (const double givenInfinity)
Set infinity.
- double [getInfinity](#) () const
Get infinity.
- virtual void [setNrow](#) (const int givenNrow)
Set nrow to the number of rows.
- int [getNrow](#) () const
Get nrow.
- virtual void [setNcol](#) (const int givenNcol)
Set ncol to the number of variables.
- int [getNcol](#) () const
Get ncol.
- virtual void [setMatrixByCol](#) (const **CoinPackedMatrix** *givenMatrixByCol)
Set matrixByCol to point on the coefficient matrix ordered by columns.
- const **CoinPackedMatrix** * [getMatrixByCol](#) () const
Get matrixByCol.
- virtual void [setMatrixByRow](#) (const **CoinPackedMatrix** *givenMatrixByRow)
Set matrixByRow to point on the coefficient matrix ordered by rows.
- const **CoinPackedMatrix** * [getMatrixByRow](#) () const
Get matrixByRow.
- virtual void [setObj](#) (const double *givenObj)
Set obj to point on a vector holding the objective coefficient values.
- const double * [getObj](#) () const
Get obj.
- virtual void [setColLower](#) (const double *givenColLower)
Set colLower to point on a vector holding the lower bounds on the variables.
- const double * [getColLower](#) () const
Get colLower.
- virtual void [setColUpper](#) (const double *givenColUpper)
Set colUpper to point on a vector holding the upper bounds on the variables.
- const double * [getColUpper](#) () const
Get colUpper.
- virtual void [setRowLower](#) (const double *givenRowLower)

- Set rowLower to point on a vector holding the lower bounds on the constraints.*

 - const double * [getRowLower](#) () const

Get rowLower.
- virtual void [setRowUpper](#) (const double *givenRowUpper)

Set rowUpper to point on a vector holding the upper bounds on the constraints.

 - const double * [getRowUpper](#) () const

Get rowUpper.
- const double * [getRowRhs](#) () const

Set rowRhs to point on a vector holding the right hand side of the constraints (for a ranged constraint, it contains the upper bound).

 - const double * [getRowRange](#) () const

Get rowRange.
- const char * [getRowSense](#) () const

Get rowSense.
- const double * [getRowActivity](#) () const

Set rowActivity to point on a vector holding the activity of the constraints (i.e.
- virtual void [setColType](#) (const char *givenColType)

Set colType to point on a vector holding the type of the variables ('B', 'I', or 'C' for Binary, Integer and Continuous)

 - const char * [getColType](#) () const

Get colType.
- virtual void [setPrimalSol](#) (const double *givenPrimalSol)

Set primal solution.

 - const double * [getPrimalSol](#) () const

Get primal solution.
- void [initializeOtherData](#) ()

initialize the non-const data
- virtual void [setInfinity](#) (const double givenInfinity)

Set infinity.

 - double [getInfinity](#) () const

Get infinity.
- virtual void [setNrow](#) (const int givenNrow)

Set nrow to the number of rows.

 - int [getNrow](#) () const

Get nrow.
- virtual void [setNcol](#) (const int givenNcol)

Set ncol to the number of variables.

 - int [getNcol](#) () const

Get ncol.
- virtual void [setMatrixByCol](#) (const **CoinPackedMatrix** *givenMatrixByCol)

Set matrixByCol to point on the coefficient matrix ordered by columns.

 - const **CoinPackedMatrix** * [getMatrixByCol](#) () const

Get matrixByCol.
- virtual void [setMatrixByRow](#) (const **CoinPackedMatrix** *givenMatrixByRow)

Set matrixByRow to point on the coefficient matrix ordered by rows.

 - const **CoinPackedMatrix** * [getMatrixByRow](#) () const

Get matrixByRow.
- virtual void [setObj](#) (const double *givenObj)

Set obj to point on a vector holding the objective coefficient values.

 - const double * [getObj](#) () const

Get obj.
- virtual void [setColLower](#) (const double *givenColLower)

Set colLower to point on a vector holding the lower bounds on the variables.

- const double * [getColLower](#) () const
Get colLower.
- virtual void [setColUpper](#) (const double *givenColUpper)
Set colUpper to point on a vector holding the upper bounds on the variables.
- const double * [getColUpper](#) () const
Get colUpper.
- virtual void [setRowLower](#) (const double *givenRowLower)
Set rowLower to point on a vector holding the lower bounds on the constraints.
- const double * [getRowLower](#) () const
Get rowLower.
- virtual void [setRowUpper](#) (const double *givenRowUpper)
Set rowUpper to point on a vector holding the upper bounds on the constraints.
- const double * [getRowUpper](#) () const
Get rowUpper.
- const double * [getRowRhs](#) () const
Set rowRhs to point on a vector holding the right hand side of the constraints (for a ranged constraint, it contains the upper bound).
- const double * [getRowRange](#) () const
Get rowRange.
- const char * [getRowSense](#) () const
Get rowSense.
- const double * [getRowActivity](#) () const
Set rowActivity to point on a vector holding the activity of the constraints (i.e.
- virtual void [setColType](#) (const char *givenColType)
Set colType to point on a vector holding the type of the variables ('B', 'I', or 'C' for Binary, Integer and Continuous)
- const char * [getColType](#) () const
Get colType.
- virtual void [setPrimalSol](#) (const double *givenPrimalSol)
Set primal solution.
- const double * [getPrimalSol](#) () const
Get primal solution.
- void [initializeOtherData](#) ()
initialize the non-const data

Constructors and destructors

- [OsiData](#) (const double givenInfinity=DBL_MAX, const int &givenNrow=0, const int &givenNcol=0, const **Coin**↔**PackedMatrix** *givenMatrixByCol=NULL, const **CoinPackedMatrix** *givenMatrixByRow=NULL, const double *givenObj=NULL, const double *givenColLower=NULL, const double *givenColUpper=NULL, const double *givenRowLower=NULL, const double *givenRowUpper=NULL, const char *givenColType=NULL, const double *givenPrimalSol=NULL)
Default constructor.
- virtual [~OsiData](#) ()
Destructor.
- [OsiData](#) (const double givenInfinity=DBL_MAX, const int &givenNrow=0, const int &givenNcol=0, const **Coin**↔**PackedMatrix** *givenMatrixByCol=NULL, const **CoinPackedMatrix** *givenMatrixByRow=NULL, const double *givenObj=NULL, const double *givenColLower=NULL, const double *givenColUpper=NULL, const double *givenRowLower=NULL, const double *givenRowUpper=NULL, const char *givenColType=NULL, const double *givenPrimalSol=NULL)
Default constructor.
- virtual [~OsiData](#) ()
Destructor.

Protected Attributes

Private member data

- double **infinity**
- int **nrow**
- int **ncol**
- **CoinPackedMatrix** const * **matrixByCol**
- **CoinPackedMatrix** const * **matrixByRow**
- const double * **obj**
- const double * **colLower**
- const double * **colUpper**
- const double * **rowLower**
- const double * **rowUpper**
- const char * **colType**
Pointer on vector of characters for columns types.
- const double * **primalSol**
- double * **rowRhs**
- double * **rowRange**
- char * **rowSense**
- double * **rowActivity**

4.44.1 Detailed Description

Class collecting pointers on data for OsiEmpty.

Each generator may have a derived class to add additional pointers on data. If a data member is not used by a generator, the data member need not be defined (or may be NULL). Ownership of the data remains with the calling method.

Definition at line 22 of file OsiData.hpp.

4.44.2 Member Function Documentation

4.44.2.1 virtual void OsiData::setInfinity (const double *givenInfinity*) [inline], [virtual]

Set infinity.

Definition at line 30 of file OsiData.hpp.

4.44.2.2 const double* OsiData::getRowRhs () const [inline]

Set rowRhs to point on a vector holding the right hand side of the constraints (for a ranged constraint, it contains the upper bound).

Get rowRhs

Definition at line 119 of file OsiData.hpp.

4.44.2.3 const double* OsiData::getRowActivity () const [inline]

Set rowActivity to point on a vector holding the activity of the constraints (i.e.

coefficient matrix times separateThis). Get rowActivity

Definition at line 129 of file OsiData.hpp.

4.44.2.4 `virtual void OsiData::setPrimalSol (const double * givenPrimalSol) [inline],[virtual]`

Set primal solution.

Definition at line 141 of file OsiData.hpp.

4.44.2.5 `const double* OsiData::getPrimalSol () const [inline]`

Get primal solution.

Definition at line 146 of file OsiData.hpp.

4.44.2.6 `virtual void OsiData::setInfinity (const double givenInfinity) [inline],[virtual]`

Set infinity.

Definition at line 30 of file OsiData2.hpp.

4.44.2.7 `const double* OsiData::getRowRhs () const [inline]`

Set rowRhs to point on a vector holding the right hand side of the constraints (for a ranged constraint, it contains the upper bound).

Get rowRhs

Definition at line 119 of file OsiData2.hpp.

4.44.2.8 `const double* OsiData::getRowActivity () const [inline]`

Set rowActivity to point on a vector holding the activity of the constraints (i.e. coefficient matrix times separateThis). Get rowActivity

Definition at line 129 of file OsiData2.hpp.

4.44.2.9 `virtual void OsiData::setPrimalSol (const double * givenPrimalSol) [inline],[virtual]`

Set primal solution.

Definition at line 141 of file OsiData2.hpp.

4.44.2.10 `const double* OsiData::getPrimalSol () const [inline]`

Get primal solution.

Definition at line 146 of file OsiData2.hpp.

4.44.3 Member Data Documentation

4.44.3.1 `const char * OsiData::colType [protected]`

Pointer on vector of characters for columns types.

colType[i] can have values

- 'C' : continuous
- 'B' : binary
- 'I' : integer

Definition at line 289 of file OsiData.hpp.

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

- OsiData.hpp
- OsiData2.hpp

4.45 OsiNullSolverInterface Class Reference

Inheritance diagram for OsiNullSolverInterface:

Collaboration diagram for OsiNullSolverInterface:

Public Member Functions

Solve methods

- virtual void [initialSolve](#) ()
Solve initial LP relaxation.
- virtual void [resolve](#) ()
Resolve an LP relaxation after problem modification.
- virtual void [branchAndBound](#) ()
Invoke solver's built-in enumeration algorithm.
- void **copyParameters** ([OsiNullSolverInterface](#) &rhs)
- virtual void [initialSolve](#) ()
Solve initial LP relaxation.
- virtual void [resolve](#) ()
Resolve an LP relaxation after problem modification.
- virtual void [branchAndBound](#) ()
Invoke solver's built-in enumeration algorithm.
- void **copyParameters** ([OsiNullSolverInterface](#) &rhs)

Methods returning info on how the solution process terminated

- virtual bool [isAbandoned](#) () const
Are there numerical difficulties?
- virtual bool [isProvenOptimal](#) () const
Is optimality proven?
- virtual bool [isProvenPrimalInfeasible](#) () const
Is primal infeasibility proven?
- virtual bool [isProvenDualInfeasible](#) () const
Is dual infeasibility proven?
- virtual bool [isPrimalObjectiveLimitReached](#) () const
Is the given primal objective limit reached?
- virtual bool [isDualObjectiveLimitReached](#) () const
Is the given dual objective limit reached?

- virtual bool `isIterationLimitReached ()` const
Iteration limit reached?
- virtual bool `isAbandoned ()` const
Are there numerical difficulties?
- virtual bool `isProvenOptimal ()` const
Is optimality proven?
- virtual bool `isProvenPrimalInfeasible ()` const
Is primal infeasibility proven?
- virtual bool `isProvenDualInfeasible ()` const
Is dual infeasibility proven?
- virtual bool `isPrimalObjectiveLimitReached ()` const
Is the given primal objective limit reached?
- virtual bool `isDualObjectiveLimitReached ()` const
Is the given dual objective limit reached?
- virtual bool `isIterationLimitReached ()` const
Iteration limit reached?

Warm start methods

Note that the warm start methods return a generic **CoinWarmStart** object.

The precise characteristics of this object are solver-dependent. Clients who wish to maintain a maximum degree of solver independence should take care to avoid unnecessary assumptions about the properties of a warm start object.

- virtual **CoinWarmStart** * `getEmptyWarmStart ()` const
Get an empty warm start object.
- virtual **CoinWarmStart** * `getWarmStart ()` const
Get warm start information.
- virtual bool `setWarmStart (const CoinWarmStart *warmstart)`
Set warm start information.
- virtual **CoinWarmStart** * `getEmptyWarmStart ()` const
Get an empty warm start object.
- virtual **CoinWarmStart** * `getWarmStart ()` const
Get warm start information.
- virtual bool `setWarmStart (const CoinWarmStart *warmstart)`
Set warm start information.

Problem query methods

Querying a problem that has no data associated with it will result in zeros for the number of rows and columns, and NULL pointers from the methods that return vectors.

Const pointers returned from any data-query method are valid as long as the data is unchanged and the solver is not called.

- virtual int `getNumCols ()` const
Get number of columns.
- virtual int `getNumRows ()` const
Get number of rows.
- virtual int `getNumElements ()` const
Get number of nonzero elements.
- virtual int `getNumIntegers ()` const
Get number of integer variables.
- virtual const double * `getColLower ()` const
Get pointer to array[getNumCols()] of column lower bounds.

- virtual const double * [getColUpper](#) () const
Get pointer to array[getNumCols()] of column upper bounds.
- virtual const char * [getRowSense](#) () const
Get pointer to array[getNumRows()] of row constraint senses.
- virtual const double * [getRightHandSide](#) () const
- virtual const double * **getRowRange** () const
- virtual const double * [getRowLower](#) () const
Get pointer to array[getNumRows()] of row lower bounds.
- virtual const double * [getRowUpper](#) () const
Get pointer to array[getNumRows()] of row upper bounds.
- virtual const double * [getObjCoefficients](#) () const
Get pointer to array[getNumCols()] of objective function coefficients.
- virtual double [getObjSense](#) () const
Get objective function sense (1 for min (default), -1 for max)
- virtual bool [isContinuous](#) (int colIndex) const
Return true if variable is continuous.
- virtual bool [isBinary](#) (int colIndex) const
Return true if variable is binary.
- virtual bool [isInteger](#) (int colIndex) const
Return true if column is integer.
- virtual const **CoinPackedMatrix** * [getMatrixByRow](#) () const
Get pointer to row-wise copy of matrix.
- virtual const **CoinPackedMatrix** * [getMatrixByCol](#) () const
Get pointer to column-wise copy of matrix.
- virtual double [getInfinity](#) () const
Get solver's value for infinity.
- virtual int [getNumCols](#) () const
Get number of columns.
- virtual int [getNumRows](#) () const
Get number of rows.
- virtual int [getNumElements](#) () const
Get number of nonzero elements.
- virtual int [getNumIntegers](#) () const
Get number of integer variables.
- virtual const double * [getColLower](#) () const
Get pointer to array[getNumCols()] of column lower bounds.
- virtual const double * [getColUpper](#) () const
Get pointer to array[getNumCols()] of column upper bounds.
- virtual const char * [getRowSense](#) () const
Get pointer to array[getNumRows()] of row constraint senses.
- virtual const double * [getRightHandSide](#) () const
- virtual const double * **getRowRange** () const
- virtual const double * [getRowLower](#) () const
Get pointer to array[getNumRows()] of row lower bounds.
- virtual const double * [getRowUpper](#) () const
Get pointer to array[getNumRows()] of row upper bounds.
- virtual const double * [getObjCoefficients](#) () const
Get pointer to array[getNumCols()] of objective function coefficients.
- virtual double [getObjSense](#) () const
Get objective function sense (1 for min (default), -1 for max)
- virtual bool [isContinuous](#) (int colIndex) const
Return true if variable is continuous.

- virtual bool **isBinary** (int colIndex) const
Return true if variable is binary.
- virtual bool **isInteger** (int colIndex) const
Return true if column is integer.
- virtual const **CoinPackedMatrix** * **getMatrixByRow** () const
Get pointer to row-wise copy of matrix.
- virtual const **CoinPackedMatrix** * **getMatrixByCol** () const
Get pointer to column-wise copy of matrix.
- virtual double **getInfinity** () const
Get solver's value for infinity.

Solution query methods

- virtual const double * **getColSolution** () const
Get pointer to array[getNumCols()] of primal variable values.
- virtual const double * **getRowPrice** () const
Get pointer to array[getNumRows()] of dual variable values.
- virtual const double * **getReducedCost** () const
Get a pointer to array[getNumCols()] of reduced costs.
- virtual const double * **getRowActivity** () const
Get pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).
- virtual double **getObjValue** () const
Get objective function value.
- virtual int **getIterationCount** () const
Get the number of iterations it took to solve the problem (whatever "iteration" means to the solver).
- virtual std::vector< double * > **getDualRays** (int maxNumRays) const
Get as many dual rays as the solver can provide.
- virtual std::vector< double * > **getPrimalRays** (int maxNumRays) const
- virtual const double * **getColSolution** () const
Get pointer to array[getNumCols()] of primal variable values.
- virtual const double * **getRowPrice** () const
Get pointer to array[getNumRows()] of dual variable values.
- virtual const double * **getReducedCost** () const
Get a pointer to array[getNumCols()] of reduced costs.
- virtual const double * **getRowActivity** () const
Get pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).
- virtual double **getObjValue** () const
Get objective function value.
- virtual int **getIterationCount** () const
Get the number of iterations it took to solve the problem (whatever "iteration" means to the solver).
- virtual std::vector< double * > **getDualRays** (int maxNumRays) const
Get as many dual rays as the solver can provide.
- virtual std::vector< double * > **getPrimalRays** (int maxNumRays) const

Methods to modify the objective, bounds, and solution

For functions which take a set of indices as parameters (**setObjCoeffSet()**, **setColSetBounds()**, **setRowSet↵
Bounds()**, **setRowSetTypes()**), the parameters follow the C++ STL iterator convention: *indexFirst* points to the first index in the set, and *indexLast* points to a position one past the last index in the set.

- virtual void **setObjCoeff** (int elementIndex, double elementValue)
Set an objective function coefficient.
- virtual void **setColLower** (int elementIndex, double elementValue)
Set a single column lower bound.

- virtual void [setColUpper](#) (int elementIndex, double elementValue)
Set a single column upper bound.
- virtual void [setRowLower](#) (int elementIndex, double elementValue)
Set a single row lower bound.
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)
Set a single row upper bound.
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)
Set the type of a single row.
- virtual void [setObjSense](#) (double s)
Set the objective function sense.
- virtual void [setColType](#) (const char *colType)
Set characters for columns types.
- virtual void **setColSolution** (const double *colsol)
- virtual void [setRowPrice](#) (const double *rowprice)
Set dual solution variable values.
- virtual void [setObjCoeff](#) (int elementIndex, double elementValue)
Set an objective function coefficient.
- virtual void [setColLower](#) (int elementIndex, double elementValue)
Set a single column lower bound.
- virtual void [setColUpper](#) (int elementIndex, double elementValue)
Set a single column upper bound.
- virtual void [setRowLower](#) (int elementIndex, double elementValue)
Set a single row lower bound.
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)
Set a single row upper bound.
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)
Set the type of a single row.
- virtual void [setObjSense](#) (double s)
Set the objective function sense.
- virtual void [setColType](#) (const char *colType)
Set characters for columns types.
- virtual void **setColSolution** (const double *colsol)
- virtual void [setRowPrice](#) (const double *rowprice)
Set dual solution variable values.

Methods to set variable type

- virtual void [setContinuous](#) (int index)
Set the index-th variable to be a continuous variable.
- virtual void [setInteger](#) (int index)
Set the index-th variable to be an integer variable.
- virtual void [setContinuous](#) (int index)
Set the index-th variable to be a continuous variable.
- virtual void [setInteger](#) (int index)
Set the index-th variable to be an integer variable.

Methods to modify the constraint system.

Set the variables listed in indices (which is of length len) to be continuous variables

Note that new columns are added as continuous variables.

- virtual void [addCol](#) (const **CoinPackedVectorBase** &vec, const double collb, const double colub, const double obj)
Add a column (primal variable) to the problem.

- virtual void **deleteCols** (const int num, const int *colIndices)
Remove a set of columns (primal variables) from the problem.
- virtual void **addRow** (const **CoinPackedVectorBase** &vec, const double rowlb, const double rowub)
Add a row (constraint) to the problem.
- virtual void **addRow** (const **CoinPackedVectorBase** &vec, const char rowsen, const double rowrhs, const double rowrng)
- virtual void **deleteRows** (const int num, const int *rowIndices)
Delete a set of rows (constraints) from the problem.
- virtual void **addCol** (const **CoinPackedVectorBase** &vec, const double collb, const double colub, const double obj)
Add a column (primal variable) to the problem.
- virtual void **deleteCols** (const int num, const int *colIndices)
Remove a set of columns (primal variables) from the problem.
- virtual void **addRow** (const **CoinPackedVectorBase** &vec, const double rowlb, const double rowub)
Add a row (constraint) to the problem.
- virtual void **addRow** (const **CoinPackedVectorBase** &vec, const char rowsen, const double rowrhs, const double rowrng)
- virtual void **deleteRows** (const int num, const int *rowIndices)
Delete a set of rows (constraints) from the problem.

Methods to input a problem

- void **loadDataAndSolution** (const **CoinPackedMatrix** &rowMatrix, const **CoinPackedMatrix** &colMatrix, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub, const char *colType, const double *primalSol, const double infinity)
- virtual void **loadProblem** (const **CoinPackedMatrix** &matrix, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub)
Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).
- virtual void **assignProblem** (**CoinPackedMatrix** *&matrix, double *&collb, double *&colub, double *&obj, double *&rowlb, double *&rowub)
Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).
- virtual void **loadProblem** (const **CoinPackedMatrix** &matrix, const double *collb, const double *colub, const double *obj, const char *rowsen, const double *rowrhs, const double *rowrng)
Load in an problem by copying the arguments (the constraints on the rows are given by sense/rhs/range triplets).
- virtual void **assignProblem** (**CoinPackedMatrix** *&matrix, double *&collb, double *&colub, double *&obj, char *&rowsen, double *&rowrhs, double *&rowrng)
Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by sense/rhs/range triplets).
- virtual void **loadProblem** (const int numcols, const int numRows, const CoinBigIndex *start, const int *index, const double *value, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub)
*Just like the other **loadProblem()** methods except that the matrix is given in a standard column major ordered format (without gaps).*
- virtual void **loadProblem** (const int numcols, const int numRows, const CoinBigIndex *start, const int *index, const double *value, const double *collb, const double *colub, const double *obj, const char *rowsen, const double *rowrhs, const double *rowrng)
*Just like the other **loadProblem()** methods except that the matrix is given in a standard column major ordered format (without gaps).*
- virtual void **writeMps** (const char *filename, const char *extension="mps", double objSense=0.0) const
Write the problem in MPS format to the specified file.
- void **loadDataAndSolution** (const **CoinPackedMatrix** &rowMatrix, const **CoinPackedMatrix** &colMatrix, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub, const char *colType, const double *primalSol, const double infinity)

- virtual void [loadProblem](#) (const **CoinPackedMatrix** &matrix, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub)
Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).
- virtual void [assignProblem](#) (**CoinPackedMatrix** *&matrix, double *&collb, double *&colub, double *&obj, double *&rowlb, double *&rowub)
Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).
- virtual void [loadProblem](#) (const **CoinPackedMatrix** &matrix, const double *collb, const double *colub, const double *obj, const char *rowsen, const double *rowrhs, const double *rowrng)
Load in an problem by copying the arguments (the constraints on the rows are given by sense/rhs/range triplets).
- virtual void [assignProblem](#) (**CoinPackedMatrix** *&matrix, double *&collb, double *&colub, double *&obj, char *&rowsen, double *&rowrhs, double *&rowrng)
Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by sense/rhs/range triplets).
- virtual void [loadProblem](#) (const int numcols, const int numRows, const CoinBigIndex *start, const int *index, const double *value, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub)
Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).
- virtual void [loadProblem](#) (const int numcols, const int numRows, const CoinBigIndex *start, const int *index, const double *value, const double *collb, const double *colub, const double *obj, const char *rowsen, const double *rowrhs, const double *rowrng)
Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).
- virtual void [writeMps](#) (const char *filename, const char *extension="mps", double objSense=0.0) const
Write the problem in MPS format to the specified file.

Constructors and destructors

- [OsiNullSolverInterface](#) ()
Default Constructor.
- virtual [OsiNullSolverInterface](#) * [clone](#) (bool copyData=true) const
Clone.
- [OsiNullSolverInterface](#) (const [OsiNullSolverInterface](#) &)
Copy constructor (disabled)
- [OsiNullSolverInterface](#) & [operator=](#) (const [OsiNullSolverInterface](#) &rhs)
Assignment operator (disabled)
- virtual [~OsiNullSolverInterface](#) ()
Destructor.
- [OsiNullSolverInterface](#) ()
Default Constructor.
- virtual [OsiNullSolverInterface](#) * [clone](#) (bool copyData=true) const
Clone.
- [OsiNullSolverInterface](#) (const [OsiNullSolverInterface](#) &)
Copy constructor (disabled)
- [OsiNullSolverInterface](#) & [operator=](#) (const [OsiNullSolverInterface](#) &rhs)
Assignment operator (disabled)
- virtual [~OsiNullSolverInterface](#) ()
Destructor.

Protected Member Functions

Protected methods

- virtual void [applyRowCut](#) (const **OsiRowCut** &rc)
Apply a row cut (append to the constraint matrix).
- virtual void [applyColCut](#) (const **OsiColCut** &cc)
Apply a column cut (adjust the bounds of one or more variables).
- template<class T >
T [forceIntoRange](#) (const T value, const T lower, const T upper) const
A quick inlined function to force a value to be between a minimum and a maximum value.
- virtual void [applyRowCut](#) (const **OsiRowCut** &rc)
Apply a row cut (append to the constraint matrix).
- virtual void [applyColCut](#) (const **OsiColCut** &cc)
Apply a column cut (adjust the bounds of one or more variables).
- template<class T >
T [forceIntoRange](#) (const T value, const T lower, const T upper) const
A quick inlined function to force a value to be between a minimum and a maximum value.

4.45.1 Detailed Description

Definition at line 38 of file OsiNullSolverInterface.hpp.

4.45.2 Member Function Documentation

4.45.2.1 virtual **CoinWarmStart*** OsiNullSolverInterface::getEmptyWarmStart () const [inline],[virtual]

Get an empty warm start object.

This routine returns an empty warm start object. Its purpose is to provide a way for a client to acquire a warm start object of the appropriate type for the solver, which can then be resized and modified as desired.

Implements **OsiSolverInterface**.

Definition at line 121 of file OsiNullSolverInterface.hpp.

4.45.2.2 virtual **CoinWarmStart*** OsiNullSolverInterface::getWarmStart () const [inline],[virtual]

Get warm start information.

Return warm start information for the current state of the solver interface. If there is no valid warm start information, an empty warm start object will be returned.

Implements **OsiSolverInterface**.

Definition at line 132 of file OsiNullSolverInterface.hpp.

4.45.2.3 virtual bool OsiNullSolverInterface::setWarmStart (const **CoinWarmStart** * warmstart) [inline],[virtual]

Set warm start information.

Return true or false depending on whether the warm start information was accepted or not. By definition, a call to setWarmStart with an empty warm start object should remove the warm start information held in the solver interface.

Implements **OsiSolverInterface**.

Definition at line 144 of file OsiNullSolverInterface.hpp.

4.45.2.4 `virtual const char* OsiNullSolverInterface::getRowSense () const [inline],[virtual]`

Get pointer to array[[getNumRows\(\)](#)] of row constraint senses.

- 'L': \leq constraint
- 'E': = constraint
- 'G': \geq constraint
- 'R': ranged constraint
- 'N': free constraint

Get pointer to array[[getNumRows\(\)](#)] of row right-hand sides

- if [getRowSense\(\)](#)[i] == 'L' then [getRightHandSide\(\)](#)[i] == [getRowUpper\(\)](#)[i]
- if [getRowSense\(\)](#)[i] == 'G' then [getRightHandSide\(\)](#)[i] == [getRowLower\(\)](#)[i]
- if [getRowSense\(\)](#)[i] == 'R' then [getRightHandSide\(\)](#)[i] == [getRowUpper\(\)](#)[i]
- if [getRowSense\(\)](#)[i] == 'N' then [getRightHandSide\(\)](#)[i] == 0.0

Implements **OsiSolverInterface**.

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

4.45.2.5 `virtual const double* OsiNullSolverInterface::getRightHandSide () const [inline],[virtual]`

Get pointer to array[[getNumRows\(\)](#)] of row ranges.

- if [getRowSense\(\)](#)[i] == 'R' then [getRowRange\(\)](#)[i] == [getRowUpper\(\)](#)[i] - [getRowLower\(\)](#)[i]
- if [getRowSense\(\)](#)[i] != 'R' then [getRowRange\(\)](#)[i] is 0.0

Implements **OsiSolverInterface**.

Definition at line 230 of file `OsiNullSolverInterface.hpp`.

4.45.2.6 `virtual bool OsiNullSolverInterface::isInteger (int colIndex) const [inline],[virtual]`

Return true if column is integer.

Note: This function returns true if the the column is binary or a general integer.

Reimplemented from **OsiSolverInterface**.

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

4.45.2.7 `virtual const double* OsiNullSolverInterface::getRowActivity () const [inline],[virtual]`

Get pointer to array[[getNumRows\(\)](#)] of row activity levels (constraint matrix times the solution vector).

Implements **OsiSolverInterface**.

Definition at line 324 of file `OsiNullSolverInterface.hpp`.

4.45.2.8 `virtual int OsiNullSolverInterface::getIterationCount () const [inline],[virtual]`

Get the number of iterations it took to solve the problem (whatever "iteration" means to the solver).

Implements **OsiSolverInterface**.

Definition at line 336 of file OsiNullSolverInterface.hpp.

4.45.2.9 `virtual std::vector<double*> OsiNullSolverInterface::getDualRays (int maxNumRays) const [inline],[virtual]`

Get as many dual rays as the solver can provide.

In case of proven primal infeasibility there should be at least one.

Note

Implementors of solver interfaces note that the double pointers in the vector should point to arrays of length [get←NumRows\(\)](#) and they should be allocated via `new[]`.

Clients of solver interfaces note that it is the client's responsibility to free the double pointers in the vector using `delete[]`.

Get as many primal rays as the solver can provide. (In case of proven dual infeasibility there should be at least one.)

NOTE for implementers of solver interfaces:

The double pointers in the vector should point to arrays of length [getNumCols\(\)](#) and they should be allocated via `new[]`.

NOTE for users of solver interfaces:

It is the user's responsibility to free the double pointers in the vector using `delete[]`.

Definition at line 353 of file OsiNullSolverInterface.hpp.

4.45.2.10 `virtual void OsiNullSolverInterface::setColLower (int elementIndex, double elementValue) [inline],[virtual]`

Set a single column lower bound.

Use `-getInfinity()` for -infinity.

Implements **OsiSolverInterface**.

Definition at line 391 of file OsiNullSolverInterface.hpp.

4.45.2.11 `virtual void OsiNullSolverInterface::setColUpper (int elementIndex, double elementValue) [inline],[virtual]`

Set a single column upper bound.

Use [getInfinity\(\)](#) for infinity.

Implements **OsiSolverInterface**.

Definition at line 397 of file OsiNullSolverInterface.hpp.

4.45.2.12 `virtual void OsiNullSolverInterface::setRowLower (int elementIndex, double elementValue) [inline],
[virtual]`

Set a single row lower bound.

Use `-getInfinity()` for -infinity.

Implements **OsiSolverInterface**.

Definition at line 403 of file `OsiNullSolverInterface.hpp`.

4.45.2.13 `virtual void OsiNullSolverInterface::setRowUpper (int elementIndex, double elementValue) [inline],
[virtual]`

Set a single row upper bound.

Use `getInfinity()` for infinity.

Implements **OsiSolverInterface**.

Definition at line 409 of file `OsiNullSolverInterface.hpp`.

4.45.2.14 `virtual void OsiNullSolverInterface::setObjSense (double s) [inline],[virtual]`

Set the objective function sense.

(1 for min (default), -1 for max)

Implements **OsiSolverInterface**.

Definition at line 421 of file `OsiNullSolverInterface.hpp`.

4.45.2.15 `virtual void OsiNullSolverInterface::setColType (const char * colType) [inline],[virtual]`

Set characters for columns types.

`colType[i]` can have values

- 'C' : continuous
- 'B' : binary
- 'I' : integer

Definition at line 433 of file `OsiNullSolverInterface.hpp`.

4.45.2.16 `virtual void OsiNullSolverInterface::setRowPrice (const double * rowprice) [inline],[virtual]`

Set dual solution variable values.

`rowprice[getNumRows()]` is an array of values for the dual variables. These values are copied to memory owned by the solver interface object or the solver. They will be returned as the result of `getRowPrice()` until changed by another call to `setRowPrice()` or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

Implements **OsiSolverInterface**.

Definition at line 462 of file `OsiNullSolverInterface.hpp`.

4.45.2.17 `virtual void OsiNullSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [inline], [virtual]`

Add a column (primal variable) to the problem.

Implements **OsiSolverInterface**.

Definition at line 492 of file OsiNullSolverInterface.hpp.

4.45.2.18 `virtual void OsiNullSolverInterface::deleteCols (const int num, const int * colIndices) [inline], [virtual]`

Remove a set of columns (primal variables) from the problem.

The solver interface for a basis-oriented solver will maintain valid warm start information if all deleted variables are nonbasic.

Implements **OsiSolverInterface**.

Definition at line 504 of file OsiNullSolverInterface.hpp.

4.45.2.19 `virtual void OsiNullSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [inline], [virtual]`

Add a row (constraint) to the problem.

Implements **OsiSolverInterface**.

Definition at line 509 of file OsiNullSolverInterface.hpp.

4.45.2.20 `virtual void OsiNullSolverInterface::deleteRows (const int num, const int * rowIndices) [inline], [virtual]`

Delete a set of rows (constraints) from the problem.

The solver interface for a basis-oriented solver will maintain valid warm start information if all deleted rows are loose.

Implements **OsiSolverInterface**.

Definition at line 526 of file OsiNullSolverInterface.hpp.

4.45.2.21 `virtual void OsiNullSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub) [inline], [virtual]`

Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).

If a pointer is 0 then the following values are the default:

- `colub`: all columns have upper bound infinity
- `collb`: all columns have lower bound 0
- `rowub`: all rows have upper bound infinity
- `rowlb`: all rows have lower bound -infinity
- `obj`: all variables have 0 objective coefficient

Implements **OsiSolverInterface**.

Definition at line 574 of file OsiNullSolverInterface.hpp.

4.45.2.22 `virtual void OsiNullSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, double *& rowlb, double *& rowub) [inline],[virtual]`

Load in a problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).

For default values see the previous method.

Warning

The arguments passed to this method will be freed using the C++ `delete` and `delete[]` functions.

Implements **OsiSolverInterface**.

Definition at line 594 of file `OsiNullSolverInterface.hpp`.

4.45.2.23 `virtual void OsiNullSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const char * rowsen, const double * rowrhs, const double * rowrng) [inline],[virtual]`

Load in a problem by copying the arguments (the constraints on the rows are given by sense/rhs/range triplets).

If a pointer is 0 then the following values are the default:

- `colub`: all columns have upper bound infinity
- `collb`: all columns have lower bound 0
- `obj`: all variables have 0 objective coefficient
- `rowsen`: all rows are \geq
- `rowrhs`: all right hand sides are 0
- `rowrng`: 0 for the ranged rows

Implements **OsiSolverInterface**.

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

4.45.2.24 `virtual void OsiNullSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, char *& rowsen, double *& rowrhs, double *& rowrng) [inline],[virtual]`

Load in a problem by assuming ownership of the arguments (the constraints on the rows are given by sense/rhs/range triplets).

For default values see the previous method.

Warning

The arguments passed to this method will be freed using the C++ `delete` and `delete[]` functions.

Implements **OsiSolverInterface**.

Definition at line 629 of file `OsiNullSolverInterface.hpp`.

4.45.2.25 `virtual void OsiNullSolverInterface::loadProblem (const int numcols, const int numrows, const CoinBigIndex * start, const int * index, const double * value, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub) [inline], [virtual]`

Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).

Implements **OsiSolverInterface**.

Definition at line 638 of file `OsiNullSolverInterface.hpp`.

4.45.2.26 `virtual void OsiNullSolverInterface::loadProblem (const int numcols, const int numrows, const CoinBigIndex * start, const int * index, const double * value, const double * collb, const double * colub, const double * obj, const char * rowSEN, const double * rowrhs, const double * rowrng) [inline], [virtual]`

Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).

Implements **OsiSolverInterface**.

Definition at line 649 of file `OsiNullSolverInterface.hpp`.

4.45.2.27 `virtual void OsiNullSolverInterface::writeMps (const char * filename, const char * extension = "mps", double objSense = 0.0) const [inline], [virtual]`

Write the problem in MPS format to the specified file.

If *objSense* is non-zero, a value of -1.0 causes the problem to be written with a maximization objective; +1.0 forces a minimization objective. If *objSense* is zero, the choice is left to implementation.

Implements **OsiSolverInterface**.

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

4.45.2.28 `virtual OsiNullSolverInterface* OsiNullSolverInterface::clone (bool copyData = true) const [inline], [virtual]`

Clone.

The result of calling `clone(false)` is defined to be equivalent to calling the default constructor [OsiNullSolverInterface\(\)](#).

Implements **OsiSolverInterface**.

Definition at line 688 of file `OsiNullSolverInterface.hpp`.

4.45.2.29 `virtual void OsiNullSolverInterface::applyRowCut (const OsiRowCut & rc) [inline], [protected], [virtual]`

Apply a row cut (append to the constraint matrix).

Implements **OsiSolverInterface**.

Definition at line 712 of file `OsiNullSolverInterface.hpp`.

4.45.2.30 `virtual void OsiNullSolverInterface::applyColCut (const OsiColCut & cc) [inline],[protected],
[virtual]`

Apply a column cut (adjust the bounds of one or more variables).

Implements **OsiSolverInterface**.

Definition at line 717 of file OsiNullSolverInterface.hpp.

4.45.2.31 `virtual CoinWarmStart* OsiNullSolverInterface::getEmptyWarmStart () const [inline],[virtual]`

Get an empty warm start object.

This routine returns an empty warm start object. Its purpose is to provide a way for a client to acquire a warm start object of the appropriate type for the solver, which can then be resized and modified as desired.

Implements **OsiSolverInterface**.

Definition at line 121 of file OsiNullSolverInterface2.hpp.

4.45.2.32 `virtual CoinWarmStart* OsiNullSolverInterface::getWarmStart () const [inline],[virtual]`

Get warm start information.

Return warm start information for the current state of the solver interface. If there is no valid warm start information, an empty warm start object will be returned.

Implements **OsiSolverInterface**.

Definition at line 132 of file OsiNullSolverInterface2.hpp.

4.45.2.33 `virtual bool OsiNullSolverInterface::setWarmStart (const CoinWarmStart * warmstart) [inline],
[virtual]`

Set warm start information.

Return true or false depending on whether the warm start information was accepted or not. By definition, a call to setWarmStart with an empty warm start object should remove the warm start information held in the solver interface.

Implements **OsiSolverInterface**.

Definition at line 144 of file OsiNullSolverInterface2.hpp.

4.45.2.34 `virtual const char* OsiNullSolverInterface::getRowSense () const [inline],[virtual]`

Get pointer to array[getNumRows()] of row constraint senses.

- 'L': <= constraint
- 'E': = constraint
- 'G': >= constraint
- 'R': ranged constraint
- 'N': free constraint

Get pointer to array[getNumRows()] of row right-hand sides

- if `getRowSense()[i] == 'L'` then `getRightHandSide()[i] == getRowUpper()[i]`
- if `getRowSense()[i] == 'G'` then `getRightHandSide()[i] == getRowLower()[i]`
- if `getRowSense()[i] == 'R'` then `getRightHandSide()[i] == getRowUpper()[i]`
- if `getRowSense()[i] == 'N'` then `getRightHandSide()[i] == 0.0`

Implements **OsiSolverInterface**.

Definition at line 213 of file `OsiNullSolverInterface2.hpp`.

4.45.2.35 `virtual const double* OsiNullSolverInterface::getRightHandSide () const [inline],[virtual]`

Get pointer to array[`getNumRows()`] of row ranges.

- if `getRowSense()[i] == 'R'` then `getRowRange()[i] == getRowUpper()[i] - getRowLower()[i]`
- if `getRowSense()[i] != 'R'` then `getRowRange()[i]` is 0.0

Implements **OsiSolverInterface**.

Definition at line 230 of file `OsiNullSolverInterface2.hpp`.

4.45.2.36 `virtual bool OsiNullSolverInterface::isInteger (int colIndex) const [inline],[virtual]`

Return true if column is integer.

Note: This function returns true if the the column is binary or a general integer.

Reimplemented from **OsiSolverInterface**.

Definition at line 282 of file `OsiNullSolverInterface2.hpp`.

4.45.2.37 `virtual const double* OsiNullSolverInterface::getRowActivity () const [inline],[virtual]`

Get pointer to array[`getNumRows()`] of row activity levels (constraint matrix times the solution vector).

Implements **OsiSolverInterface**.

Definition at line 324 of file `OsiNullSolverInterface2.hpp`.

4.45.2.38 `virtual int OsiNullSolverInterface::getIterationCount () const [inline],[virtual]`

Get the number of iterations it took to solve the problem (whatever "iteration" means to the solver).

Implements **OsiSolverInterface**.

Definition at line 336 of file `OsiNullSolverInterface2.hpp`.

4.45.2.39 `virtual std::vector<double*> OsiNullSolverInterface::getDualRays (int maxNumRays) const [inline],[virtual]`

Get as many dual rays as the solver can provide.

In case of proven primal infeasibility there should be at least one.

Note

Implementors of solver interfaces note that the double pointers in the vector should point to arrays of length `getNumRows()` and they should be allocated via `new[]`.

Clients of solver interfaces note that it is the client's responsibility to free the double pointers in the vector using `delete[]`.

Get as many primal rays as the solver can provide. (In case of proven dual infeasibility there should be at least one.)

NOTE for implementers of solver interfaces:

The double pointers in the vector should point to arrays of length `getNumCols()` and they should be allocated via `new[]`.

NOTE for users of solver interfaces:

It is the user's responsibility to free the double pointers in the vector using `delete[]`.

Definition at line 353 of file `OsiNullSolverInterface2.hpp`.

```
4.45.2.40 virtual void OsiNullSolverInterface::setColLower ( int elementIndex, double elementValue ) [inline],
          [virtual]
```

Set a single column lower bound.

Use `-getInfinity()` for -infinity.

Implements **OsiSolverInterface**.

Definition at line 391 of file `OsiNullSolverInterface2.hpp`.

```
4.45.2.41 virtual void OsiNullSolverInterface::setColUpper ( int elementIndex, double elementValue ) [inline],
          [virtual]
```

Set a single column upper bound.

Use `getInfinity()` for infinity.

Implements **OsiSolverInterface**.

Definition at line 397 of file `OsiNullSolverInterface2.hpp`.

```
4.45.2.42 virtual void OsiNullSolverInterface::setRowLower ( int elementIndex, double elementValue ) [inline],
          [virtual]
```

Set a single row lower bound.

Use `-getInfinity()` for -infinity.

Implements **OsiSolverInterface**.

Definition at line 403 of file `OsiNullSolverInterface2.hpp`.

```
4.45.2.43 virtual void OsiNullSolverInterface::setRowUpper ( int elementIndex, double elementValue ) [inline],
          [virtual]
```

Set a single row upper bound.

Use `getInfinity()` for infinity.

Implements **OsiSolverInterface**.

Definition at line 409 of file OsiNullSolverInterface2.hpp.

4.45.2.44 `virtual void OsiNullSolverInterface::setObjSense (double s) [inline],[virtual]`

Set the objective function sense.

(1 for min (default), -1 for max)

Implements **OsiSolverInterface**.

Definition at line 421 of file OsiNullSolverInterface2.hpp.

4.45.2.45 `virtual void OsiNullSolverInterface::setColType (const char * colType) [inline],[virtual]`

Set characters for columns types.

colType[i] can have values

- 'C' : continuous
- 'B' : binary
- 'I' : integer

Definition at line 433 of file OsiNullSolverInterface2.hpp.

4.45.2.46 `virtual void OsiNullSolverInterface::setRowPrice (const double * rowprice) [inline],[virtual]`

Set dual solution variable values.

rowprice[getNumRows()] is an array of values for the dual variables. These values are copied to memory owned by the solver interface object or the solver. They will be returned as the result of [getRowPrice\(\)](#) until changed by another call to [setRowPrice\(\)](#) or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

Implements **OsiSolverInterface**.

Definition at line 462 of file OsiNullSolverInterface2.hpp.

4.45.2.47 `virtual void OsiNullSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [inline],[virtual]`

Add a column (primal variable) to the problem.

Implements **OsiSolverInterface**.

Definition at line 492 of file OsiNullSolverInterface2.hpp.

4.45.2.48 `virtual void OsiNullSolverInterface::deleteCols (const int num, const int * colIndices) [inline],[virtual]`

Remove a set of columns (primal variables) from the problem.

The solver interface for a basis-oriented solver will maintain valid warm start information if all deleted variables are nonbasic.

Implements **OsiSolverInterface**.

Definition at line 504 of file OsiNullSolverInterface2.hpp.

4.45.2.49 `virtual void OsiNullSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [inline],[virtual]`

Add a row (constraint) to the problem.

Implements **OsiSolverInterface**.

Definition at line 509 of file OsiNullSolverInterface2.hpp.

4.45.2.50 `virtual void OsiNullSolverInterface::deleteRows (const int num, const int * rowIndices) [inline],[virtual]`

Delete a set of rows (constraints) from the problem.

The solver interface for a basis-oriented solver will maintain valid warm start information if all deleted rows are loose.

Implements **OsiSolverInterface**.

Definition at line 526 of file OsiNullSolverInterface2.hpp.

4.45.2.51 `virtual void OsiNullSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub) [inline],[virtual]`

Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).

If a pointer is 0 then the following values are the default:

- `colub`: all columns have upper bound infinity
- `collb`: all columns have lower bound 0
- `rowub`: all rows have upper bound infinity
- `rowlb`: all rows have lower bound -infinity
- `obj`: all variables have 0 objective coefficient

Implements **OsiSolverInterface**.

Definition at line 574 of file OsiNullSolverInterface2.hpp.

4.45.2.52 `virtual void OsiNullSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, double *& rowlb, double *& rowub) [inline],[virtual]`

Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).

For default values see the previous method.

Warning

The arguments passed to this method will be freed using the C++ `delete` and `delete[]` functions.

Implements **OsiSolverInterface**.

Definition at line 594 of file OsiNullSolverInterface2.hpp.

4.45.2.53 `virtual void OsiNullSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const char * rowsen, const double * rowrhs, const double * rowrng) [inline], [virtual]`

Load in a problem by copying the arguments (the constraints on the rows are given by sense/rhs/range triplets).

If a pointer is 0 then the following values are the default:

- *colub*: all columns have upper bound infinity
- *collb*: all columns have lower bound 0
- *obj*: all variables have 0 objective coefficient
- *rowsen*: all rows are \geq
- *rowrhs*: all right hand sides are 0
- *rowrng*: 0 for the ranged rows

Implements **OsiSolverInterface**.

Definition at line 612 of file OsiNullSolverInterface2.hpp.

4.45.2.54 `virtual void OsiNullSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, char *& rowsen, double *& rowrhs, double *& rowrng) [inline], [virtual]`

Load in a problem by assuming ownership of the arguments (the constraints on the rows are given by sense/rhs/range triplets).

For default values see the previous method.

Warning

The arguments passed to this method will be freed using the C++ `delete` and `delete[]` functions.

Implements **OsiSolverInterface**.

Definition at line 629 of file OsiNullSolverInterface2.hpp.

4.45.2.55 `virtual void OsiNullSolverInterface::loadProblem (const int numcols, const int numrows, const CoinBigIndex * start, const int * index, const double * value, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub) [inline], [virtual]`

Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).

Implements **OsiSolverInterface**.

Definition at line 638 of file OsiNullSolverInterface2.hpp.

4.45.2.56 `virtual void OsiNullSolverInterface::loadProblem (const int numcols, const int numrows, const CoinBigIndex * start, const int * index, const double * value, const double * collb, const double * colub, const double * obj, const char * rowsen, const double * rowrhs, const double * rowrng) [inline], [virtual]`

Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).

Implements **OsiSolverInterface**.

Definition at line 649 of file OsiNullSolverInterface2.hpp.

```
4.45.2.57 virtual void OsiNullSolverInterface::writeMps ( const char * filename, const char * extension = "mps", double objSense
              = 0.0 ) const [inline], [virtual]
```

Write the problem in MPS format to the specified file.

If objSense is non-zero, a value of -1.0 causes the problem to be written with a maximization objective; +1.0 forces a minimization objective. If objSense is zero, the choice is left to implementation.

Implements **OsiSolverInterface**.

Definition at line 665 of file OsiNullSolverInterface2.hpp.

```
4.45.2.58 virtual OsiNullSolverInterface* OsiNullSolverInterface::clone ( bool copyData = true ) const [inline],
              [virtual]
```

Clone.

The result of calling clone(false) is defined to be equivalent to calling the default constructor [OsiNullSolverInterface\(\)](#).

Implements **OsiSolverInterface**.

Definition at line 688 of file OsiNullSolverInterface2.hpp.

```
4.45.2.59 virtual void OsiNullSolverInterface::applyRowCut ( const OsiRowCut & rc ) [inline], [protected],
              [virtual]
```

Apply a row cut (append to the constraint matrix).

Implements **OsiSolverInterface**.

Definition at line 712 of file OsiNullSolverInterface2.hpp.

```
4.45.2.60 virtual void OsiNullSolverInterface::applyColCut ( const OsiColCut & cc ) [inline], [protected],
              [virtual]
```

Apply a column cut (adjust the bounds of one or more variables).

Implements **OsiSolverInterface**.

Definition at line 717 of file OsiNullSolverInterface2.hpp.

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

- OsiNullSolverInterface.hpp
- OsiNullSolverInterface2.hpp

4.46 Perturb Struct Reference

4.46.1 Detailed Description

Definition at line 360 of file old/UtilMacros.h.

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

- old/UtilMacros.h

4.47 SOR_IntDbIArrT Struct Reference

Collaboration diagram for SOR_IntDbIArrT:

4.47.1 Detailed Description

Definition at line 29 of file UtilKnapsack.h.

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

- UtilKnapsack.h

4.48 SOR_IntDbIT Struct Reference

4.48.1 Detailed Description

Definition at line 24 of file UtilKnapsack.h.

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

- UtilKnapsack.h

4.49 UtilApp Class Reference

Collaboration diagram for UtilApp:

4.49.1 Detailed Description

Definition at line 21 of file UtilApp.h.

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

- UtilApp.h

4.50 UtilGraphLib Class Reference

Collaboration diagram for UtilGraphLib:

4.50.1 Detailed Description

Definition at line 33 of file old/UtilGraphLib.h.

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

- [old/UtilGraphLib.h](#)

4.51 `UtilsGreaterThan< S, T >` Class Template Reference

4.51.1 Detailed Description

`template<class S, class T>class UtilsGreaterThan< S, T >`

Definition at line 472 of file `old/UtilMacros.h`.

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

- [old/UtilMacros.h](#)

4.52 `UtilsLessThan< S, T >` Class Template Reference

4.52.1 Detailed Description

`template<class S, class T>class UtilsLessThan< S, T >`

Definition at line 481 of file `old/UtilMacros.h`.

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

- [old/UtilMacros.h](#)

4.53 `UtilParameters` Class Reference

4.53.1 Detailed Description

Definition at line 30 of file `old/UtilParameters.h`.

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

- [old/UtilParameters.h](#)

4.54 `UtilParamT` Struct Reference

Collaboration diagram for `UtilParamT`:

4.54.1 Detailed Description

Definition at line 23 of file `old/UtilParameters.h`.

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

- [old/UtilParameters.h](#)

4.55 UtilTimer Class Reference

Public Member Functions

- void `reset` ()
Reset.
- void `start` ()
Start to count times.
- void `stop` ()
Stop timer and computing times.
- double `getCpuTime` ()
Get cpu time.
- double `getRealTime` ()
Get wallClock time.
- bool `isPast` (double limit)
Return whether the given amount of real time has elapsed since the timer was started.

4.55.1 Detailed Description

Definition at line 24 of file UtilTimer.h.

4.55.2 Member Function Documentation

4.55.2.1 void UtilTimer::reset () [inline]

Reset.

Definition at line 45 of file UtilTimer.h.

4.55.2.2 void UtilTimer::start () [inline]

Start to count times.

Definition at line 54 of file UtilTimer.h.

4.55.2.3 void UtilTimer::stop () [inline]

Stop timer and computing times.

Definition at line 60 of file UtilTimer.h.

4.55.2.4 double UtilTimer::getCpuTime () [inline]

Get cpu time.

Definition at line 68 of file UtilTimer.h.

4.55.2.5 `double UtilTimer::getRealTime () [inline]`

Get wallClock time.

Definition at line 75 of file UtilTimer.h.

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

- UtilTimer.h

File Documentation

Index

- ~AlpsDecompNodeDesc
 - AlpsDecompNodeDesc, [24](#)
- ~BcpsDecompNodeDesc
 - BcpsDecompNodeDesc, [34](#)
- APPisUserFeasible
 - DecompApp, [53](#)
 - DippyDecompApp, [73](#)
- addCol
 - OsiNullSolverInterface, [90](#), [97](#)
- AddOffset < T >, [21](#)
- addRow
 - OsiNullSolverInterface, [91](#), [97](#)
- AlpsDecompModel, [21](#)
 - fathomAllNodes, [23](#)
- AlpsDecompNodeDesc, [23](#)
 - ~AlpsDecompNodeDesc, [24](#)
 - AlpsDecompNodeDesc, [24](#)
 - basis_, [26](#)
 - branched_, [26](#)
 - branchedDir_, [26](#)
 - decode, [26](#)
 - decodeAlpsDecomp, [25](#)
 - encode, [25](#)
 - encodeAlpsDecomp, [25](#)
 - getBasis, [25](#)
 - getBranched, [25](#)
 - getBranchedDir, [25](#)
 - setBasis, [25](#)
 - setBranched, [25](#)
 - setBranchedDir, [25](#)
- AlpsDecompParam, [26](#)
 - checkMemory, [27](#)
 - logFileLevel, [27](#)
 - msgLevel, [28](#)
 - nodeLimit, [28](#)
 - nodeLogInterval, [28](#)
 - printSolution, [27](#)
- AlpsDecompSolution, [28](#)
 - getQuality, [29](#)
 - getSize, [29](#)
 - getValues, [29](#)
 - m_app, [30](#)
 - m_quality, [30](#)
 - m_size, [30](#)
 - m_values, [30](#)
- print, [29](#)
- AlpsDecompTreeNode, [30](#)
 - AlpsDecompTreeNode, [31](#)
 - branch, [31](#)
 - chooseBranchingObject, [31](#)
 - createNewTreeNode, [31](#)
 - process, [31](#)
- applyColCut
 - OsiNullSolverInterface, [93](#), [100](#)
- applyRowCut
 - OsiNullSolverInterface, [93](#), [100](#)
- assignProblem
 - OsiNullSolverInterface, [91](#), [92](#), [98](#), [99](#)
- basis_
 - AlpsDecompNodeDesc, [26](#)
 - BcpsDecompNodeDesc, [36](#)
- BcpsDecompModel, [32](#)
 - BcpsDecompModel, [32](#)
- BcpsDecompNodeDesc, [33](#)
 - ~BcpsDecompNodeDesc, [34](#)
 - basis_, [36](#)
 - BcpsDecompNodeDesc, [34](#)
 - branchedDir_, [36](#)
 - branchedInd_, [36](#)
 - branchedVal_, [36](#)
 - decode, [35](#)
 - decodeBcpsDecomp, [35](#)
 - encode, [35](#)
 - encodeBcpsDecomp, [35](#)
 - getBasis, [34](#)
 - getBranchedDir, [34](#)
 - getBranchedInd, [35](#)
 - getBranchedVal, [35](#)
 - numberRows_, [36](#)
 - setBasis, [34](#)
 - setBranchedDir, [34](#)
 - setBranchedInd, [35](#)
 - setBranchedVal, [35](#)
- BcpsDecompSolution, [36](#)
 - print, [37](#)
- BcpsDecompTreeNode, [37](#)
 - BcpsDecompTreeNode, [38](#)
 - branch, [38](#)
 - chooseBranchingObject, [38](#)
 - createNewTreeNode, [38](#)

- process, 38
- bestBound
 - DecompObjBound, 59
- bestBoundIP
 - DecompObjBound, 59
- BlockFileFormat
 - DecompParam, 62
- branch
 - AlpsDecompTreeNode, 31
 - BcpsDecompTreeNode, 38
- BranchStrongIter
 - DecompParam, 62
- branched_
 - AlpsDecompNodeDesc, 26
- branchedDir_
 - AlpsDecompNodeDesc, 26
 - BcpsDecompNodeDesc, 36
- branchedInd_
 - BcpsDecompNodeDesc, 36
- branchedVal_
 - BcpsDecompNodeDesc, 36
- checkMemory
 - AlpsDecompParam, 27
- chooseBranchingObject
 - AlpsDecompTreeNode, 31
 - BcpsDecompTreeNode, 38
- clone
 - OsiNullSolverInterface, 93, 100
- colType
 - OsiData, 79
- createNewTreeNode
 - AlpsDecompTreeNode, 31
 - BcpsDecompTreeNode, 38
- DebugCheckBlocksColumns
 - DecompParam, 62
- decode
 - AlpsDecompNodeDesc, 26
 - BcpsDecompNodeDesc, 35
- decodeAlpsDecomp
 - AlpsDecompNodeDesc, 25
- decodeBcpsDecomp
 - BcpsDecompNodeDesc, 35
- DecompAlgo, 39
 - generateInitVars, 44
 - m_cutgenSI, 44
 - m_cutoffUB, 44
 - m_masterSI, 44
 - postProcessBranch, 43
 - postProcessNode, 43
 - recomposeSolution, 43
- DecompAlgoC, 44
 - generateInitVars, 45
 - recomposeSolution, 45
- DecompAlgoCGL, 46
- DecompAlgoD, 47
 - recomposeSolution, 47
- DecompAlgoPC, 48
 - recomposeSolution, 48
- DecompAlgoRC, 49
- DecompApp, 49
 - APPisUserFeasible, 53
 - DecompApp, 52
 - getDualForGenerateVars, 53
 - initDualVector, 52
 - m_decompAlgo, 53
 - m_threadIndex, 54
 - setModelCore, 52
 - setModelObjective, 52
 - setModelRelax, 52
- DecompConstraintSet, 54
- DecompCut, 54
 - decreaseEffCnt, 55
 - increaseEffCnt, 55
- DecompCutOsi, 55
- DecompCutPool, 56
- DecompMainParam, 56
- DecompMemPool, 56
- DecompModel, 56
 - vars, 57
- DecompNodeStats, 57
 - objHistoryBound, 58
- DecompObjBound, 58
 - bestBound, 59
 - bestBoundIP, 59
- DecompParam, 59
 - BlockFileFormat, 62
 - BranchStrongIter, 62
 - DebugCheckBlocksColumns, 62
 - dumpSettings, 61
 - getSettingsImpl, 61
- DecompSolution, 62
 - DecompSolution, 64
 - getQuality, 64, 65
 - getSize, 64, 65
 - getValues, 64, 65
 - m_quality, 65
 - m_size, 65
 - m_values, 65
 - print, 64, 65
- DecompSolverResult, 66
- DecompStats, 66
- DecompSubModel, 67
- DecompVar, 67
 - decreaseEffCnt, 68
 - increaseEffCnt, 68
- DecompVarPool, 68
- DecompWaitingCol, 68

- DecompWaitingRow, 69
- decreaseEffCnt
 - DecompCut, 55
 - DecompVar, 68
- deleteCols
 - OsiNullSolverInterface, 91, 97
- deleteRows
 - OsiNullSolverInterface, 91, 98
- DippyAlgoC, 69
 - postProcessBranch, 69
 - postProcessNode, 69
- DippyAlgoMixin, 70
 - DippyAlgoMixin, 70
- DippyAlgoPC, 70
 - postProcessBranch, 71
 - postProcessNode, 71
- DippyAlgoRC, 71
 - postProcessBranch, 72
 - postProcessNode, 72
- DippyDecompApp, 72
 - APPisUserFeasible, 73
- DippyDecompCut, 74
- dumpSettings
 - DecompParam, 61
- encode
 - AlpsDecompNodeDesc, 25
 - BcpsDecompNodeDesc, 35
- encodeAlpsDecomp
 - AlpsDecompNodeDesc, 25
- encodeBcpsDecomp
 - BcpsDecompNodeDesc, 35
- fathomAllNodes
 - AlpsDecompModel, 23
- generateInitVars
 - DecompAlgo, 44
 - DecompAlgoC, 45
- getBasis
 - AlpsDecompNodeDesc, 25
 - BcpsDecompNodeDesc, 34
- getBranched
 - AlpsDecompNodeDesc, 25
- getBranchedDir
 - AlpsDecompNodeDesc, 25
 - BcpsDecompNodeDesc, 34
- getBranchedInd
 - BcpsDecompNodeDesc, 35
- getBranchedVal
 - BcpsDecompNodeDesc, 35
- getCpuTime
 - UtilTimer, 103
- getDualForGenerateVars
 - DecompApp, 53
- getDualRays
 - OsiNullSolverInterface, 89, 95
- getEmptyWarmStart
 - OsiNullSolverInterface, 87, 94
- getIterationCount
 - OsiNullSolverInterface, 88, 95
- getPrimalSol
 - OsiData, 79
- getQuality
 - AlpsDecompSolution, 29
 - DecompSolution, 64, 65
- getRealTime
 - UtilTimer, 103
- getRightHandSide
 - OsiNullSolverInterface, 88, 95
- getRowActivity
 - OsiData, 78, 79
 - OsiNullSolverInterface, 88, 95
- getRowRhs
 - OsiData, 78, 79
- getRowSense
 - OsiNullSolverInterface, 87, 94
- getSettingsImpl
 - DecompParam, 61
- getSize
 - AlpsDecompSolution, 29
 - DecompSolution, 64, 65
- getValues
 - AlpsDecompSolution, 29
 - DecompSolution, 64, 65
- getWarmStart
 - OsiNullSolverInterface, 87, 94
- increaseEffCnt
 - DecompCut, 55
 - DecompVar, 68
- initDualVector
 - DecompApp, 52
- is_greater_thanD, 74
- is_less_thanD, 74
- isInteger
 - OsiNullSolverInterface, 88, 95
- loadProblem
 - OsiNullSolverInterface, 91–93, 98, 99
- logFileLevel
 - AlpsDecompParam, 27
- m_app
 - AlpsDecompSolution, 30
- m_cutgenSI
 - DecompAlgo, 44
- m_cutoffUB
 - DecompAlgo, 44
- m_decompAlgo

- DecompApp, 53
- m_masterSI
 - DecompAlgo, 44
- m_quality
 - AlpsDecompSolution, 30
 - DecompSolution, 65
- m_size
 - AlpsDecompSolution, 30
 - DecompSolution, 65
- m_threadIndex
 - DecompApp, 54
- m_values
 - AlpsDecompSolution, 30
 - DecompSolution, 65
- msgLevel
 - AlpsDecompParam, 28
- nodeLimit
 - AlpsDecompParam, 28
- nodeLogInterval
 - AlpsDecompParam, 28
- numberOfRows_
 - BcpsDecompNodeDesc, 36
- objHistoryBound
 - DecompNodeStats, 58
- OsiData, 75
 - colType, 79
 - getPrimalSol, 79
 - getRowActivity, 78, 79
 - getRowRhs, 78, 79
 - setInfinity, 78, 79
 - setPrimalSol, 78, 79
- OsiNullSolverInterface, 80
 - addCol, 90, 97
 - addRow, 91, 97
 - applyColCut, 93, 100
 - applyRowCut, 93, 100
 - assignProblem, 91, 92, 98, 99
 - clone, 93, 100
 - deleteCols, 91, 97
 - deleteRows, 91, 98
 - getDualRays, 89, 95
 - getEmptyWarmStart, 87, 94
 - getIterationCount, 88, 95
 - getRightHandSide, 88, 95
 - getRowActivity, 88, 95
 - getRowSense, 87, 94
 - getWarmStart, 87, 94
 - isInteger, 88, 95
 - loadProblem, 91–93, 98, 99
 - setColLower, 89, 96
 - setColType, 90, 97
 - setColUpper, 89, 96
 - setObjSense, 90, 97
 - setRowLower, 89, 96
 - setRowPrice, 90, 97
 - setRowUpper, 90, 96
 - setWarmStart, 87, 94
 - writeMps, 93, 100
- Perturb, 100
- postProcessBranch
 - DecompAlgo, 43
 - DippyAlgoC, 69
 - DippyAlgoPC, 71
 - DippyAlgoRC, 72
- postProcessNode
 - DecompAlgo, 43
 - DippyAlgoC, 69
 - DippyAlgoPC, 71
 - DippyAlgoRC, 72
- print
 - AlpsDecompSolution, 29
 - BcpsDecompSolution, 37
 - DecompSolution, 64, 65
- printSolution
 - AlpsDecompParam, 27
- process
 - AlpsDecompTreeNode, 31
 - BcpsDecompTreeNode, 38
- recomposeSolution
 - DecompAlgo, 43
 - DecompAlgoC, 45
 - DecompAlgoD, 47
 - DecompAlgoPC, 48
- reset
 - UtilTimer, 103
- SOR_IntDblArrT, 101
- SOR_IntDblIT, 101
- setBasis
 - AlpsDecompNodeDesc, 25
 - BcpsDecompNodeDesc, 34
- setBranched
 - AlpsDecompNodeDesc, 25
- setBranchedDir
 - AlpsDecompNodeDesc, 25
 - BcpsDecompNodeDesc, 34
- setBranchedInd
 - BcpsDecompNodeDesc, 35
- setBranchedVal
 - BcpsDecompNodeDesc, 35
- setColLower
 - OsiNullSolverInterface, 89, 96
- setColType
 - OsiNullSolverInterface, 90, 97
- setColUpper
 - OsiNullSolverInterface, 89, 96

- setInfinity
 - OsiData, [78](#), [79](#)
- setModelCore
 - DecompApp, [52](#)
- setModelObjective
 - DecompApp, [52](#)
- setModelRelax
 - DecompApp, [52](#)
- setObjSense
 - OsiNullSolverInterface, [90](#), [97](#)
- setPrimalSol
 - OsiData, [78](#), [79](#)
- setRowLower
 - OsiNullSolverInterface, [89](#), [96](#)
- setRowPrice
 - OsiNullSolverInterface, [90](#), [97](#)
- setRowUpper
 - OsiNullSolverInterface, [90](#), [96](#)
- setWarmStart
 - OsiNullSolverInterface, [87](#), [94](#)
- start
 - UtilTimer, [103](#)
- stop
 - UtilTimer, [103](#)
- UtilApp, [101](#)
- UtilGraphLib, [101](#)
- UtilsGreaterThan< S, T >, [102](#)
- UtilsLessThan< S, T >, [102](#)
- UtilParamT, [102](#)
- UtilParameters, [102](#)
- UtilTimer, [103](#)
 - getCpuTime, [103](#)
 - getRealTime, [103](#)
 - reset, [103](#)
 - start, [103](#)
 - stop, [103](#)
- vars
 - DecompModel, [57](#)
- writeMps
 - OsiNullSolverInterface, [93](#), [100](#)