

Bcps
0.94

Generated by Doxygen 1.8.9.1

Thu Oct 8 2015 22:50:48

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	9
2.1	Class List	9
3	File Index	11
3.1	File List	11
4	Class Documentation	13
4.1	BcpsBranchObject Class Reference	13
4.2	BcpsBranchStrategy Class Reference	13
4.2.1	Detailed Description	14
4.2.2	Constructor & Destructor Documentation	14
4.2.2.1	BcpsBranchStrategy	14
4.2.2.2	BcpsBranchStrategy	15
4.2.2.3	~BcpsBranchStrategy	15
4.2.3	Member Function Documentation	15
4.2.3.1	clone	15
4.2.3.2	getType	15
4.2.3.3	setType	15
4.2.3.4	setModel	15
4.2.3.5	getNumBranchObjects	15
4.2.3.6	clearBest	15
4.2.3.7	createCandBranchObjects	15
4.2.3.8	betterBranchObject	16
4.2.3.9	bestBranchObject	16
4.2.4	Member Data Documentation	16
4.2.4.1	type_	16

4.2.4.2	model_	16
4.2.4.3	numBranchObjects_	16
4.2.4.4	branchObjects_	16
4.2.4.5	bestBranchObject_	16
4.2.4.6	bestChangeUp_	17
4.2.4.7	bestNumberUp_	17
4.2.4.8	bestChangeDown_	17
4.2.4.9	bestNumberDown_	17
4.3	BcpsConstraint Class Reference	17
4.3.1	Detailed Description	17
4.3.2	Constructor & Destructor Documentation	18
4.3.2.1	BcpsConstraint	18
4.3.2.2	BcpsConstraint	18
4.3.2.3	~BcpsConstraint	18
4.3.2.4	BcpsConstraint	18
4.4	BcpsConstraintPool Class Reference	18
4.4.1	Detailed Description	19
4.4.2	Member Function Documentation	19
4.4.2.1	getNumConstraints	19
4.4.2.2	getConstraints	19
4.4.2.3	getConstraint	19
4.5	BcpsFieldListMod< T > Struct Template Reference	19
4.5.1	Detailed Description	20
4.5.2	Member Data Documentation	20
4.5.2.1	numModify	20
4.5.2.2	posModify	20
4.6	BcpsMessage Class Reference	20
4.6.1	Detailed Description	20
4.7	BcpsModel Class Reference	20
4.7.1	Detailed Description	21
4.7.2	Member Function Documentation	21
4.7.2.1	getVariables	21
4.7.2.2	getConstrints	22
4.7.2.3	bcpsMessageHandler	22
4.7.2.4	bcpsMessages	22
4.7.2.5	encodeBcps	22
4.7.2.6	decodeBcps	22

4.7.3	Member Data Documentation	22
4.7.3.1	constraints_	22
4.7.3.2	variables_	22
4.7.3.3	numCoreConstraints_	22
4.7.3.4	numCoreVariables_	22
4.7.3.5	bcpsMessageHandler_	23
4.7.3.6	bcpsMessages_	23
4.8	BcpsNodeDesc Class Reference	23
4.8.1	Detailed Description	25
4.8.2	Constructor & Destructor Documentation	26
4.8.2.1	BcpsNodeDesc	26
4.8.2.2	BcpsNodeDesc	26
4.8.2.3	~BcpsNodeDesc	26
4.8.3	Member Function Documentation	26
4.8.3.1	initToNull	26
4.8.3.2	setVars	26
4.8.3.3	assignVars	26
4.8.3.4	setCons	26
4.8.3.5	assignCons	26
4.8.3.6	getVars	27
4.8.3.7	getCons	27
4.8.3.8	vars	27
4.8.3.9	cons	27
4.8.3.10	assignVarSoftBound	27
4.8.3.11	setVarSoftBound	27
4.8.3.12	assignVarHardBound	27
4.8.3.13	setConSoftBound	27
4.8.3.14	setVarHardBound	28
4.8.3.15	setConHardBound	28
4.8.3.16	appendAddedConstraints	28
4.8.3.17	setAddedConstraints	28
4.8.3.18	delConstraints	28
4.8.3.19	addVariables	28
4.8.3.20	delVariables	28
4.8.3.21	encodeDbiFieldMods	29
4.8.3.22	encodeIntFieldMods	29
4.8.3.23	encodeObjectMods	29

4.8.3.24	decodeDbfFieldMods	29
4.8.3.25	decodeIntFieldMods	29
4.8.3.26	decodeObjectMods	29
4.8.3.27	encodeBcps	29
4.8.3.28	decodeBcps	29
4.8.4	Member Data Documentation	29
4.8.4.1	vars_	29
4.8.4.2	cons_	30
4.9	BcpsObject Class Reference	30
4.9.1	Detailed Description	32
4.9.2	Constructor & Destructor Documentation	32
4.9.2.1	BcpsObject	32
4.9.3	Member Function Documentation	32
4.9.3.1	operator=	32
4.9.3.2	clone	32
4.9.3.3	infeasibility	32
4.9.3.4	feasibleRegion	32
4.9.3.5	createBranchObject	32
4.9.3.6	preferredNewFeasible	33
4.9.3.7	notPreferredNewFeasible	33
4.9.3.8	resetBounds	33
4.9.3.9	boundBranch	33
4.9.3.10	floorCeiling	33
4.9.3.11	upEstimate	33
4.9.3.12	downEstimate	34
4.9.3.13	encodeBcpsObject	34
4.9.3.14	decodeBcpsObject	34
4.9.3.15	encode	34
4.9.3.16	decode	34
4.9.4	Member Data Documentation	34
4.9.4.1	intType_	34
4.9.4.2	validRegion_	34
4.9.4.3	hashValue_	35
4.9.4.4	numInactive_	35
4.9.4.5	effectiveness_	35
4.10	BcpsObjectListMod Struct Reference	35
4.10.1	Detailed Description	35

4.10.2	Member Data Documentation	36
4.10.2.1	numRemove	36
4.10.2.2	posRemove	36
4.10.2.3	numAdd	36
4.10.2.4	objects	36
4.10.2.5	lbHard	36
4.11	BcpsObjectPool Class Reference	36
4.11.1	Detailed Description	37
4.11.2	Constructor & Destructor Documentation	37
4.11.2.1	BcpsObjectPool	37
4.11.3	Member Function Documentation	37
4.11.3.1	freeGuts	37
4.11.3.2	clear	37
4.11.3.3	getNumKnowledges	38
4.11.3.4	hasKnowledge	38
4.11.3.5	getObjects	38
4.11.3.6	getObject	38
4.12	BcpsSolution Class Reference	38
4.12.1	Detailed Description	39
4.12.2	Constructor & Destructor Documentation	39
4.12.2.1	BcpsSolution	39
4.12.2.2	BcpsSolution	39
4.12.2.3	BcpsSolution	40
4.12.2.4	~BcpsSolution	40
4.12.3	Member Function Documentation	40
4.12.3.1	getSize	40
4.12.3.2	setSize	40
4.12.3.3	selectNonzeros	40
4.12.3.4	print	40
4.12.3.5	encodeBcps	40
4.12.3.6	decodeBcps	40
4.12.4	Member Data Documentation	40
4.12.4.1	size_	41
4.12.4.2	objects_	41
4.12.4.3	values_	41
4.12.4.4	quality_	41
4.13	BcpsSubTree Class Reference	41

4.13.1 Detailed Description	41
4.14 BcpsTreeNode Class Reference	42
4.14.1 Detailed Description	43
4.14.2 Constructor & Destructor Documentation	43
4.14.2.1 BcpsTreeNode	43
4.14.2.2 ~BcpsTreeNode	43
4.14.3 Member Function Documentation	43
4.14.3.1 generateConstraints	43
4.14.3.2 generateVariables	43
4.14.3.3 chooseBranchingObject	43
4.14.3.4 installSubProblem	43
4.14.3.5 handleBoundingStatus	44
4.14.3.6 process	44
4.14.3.7 bound	44
4.14.3.8 branch	44
4.14.3.9 branchObject	44
4.14.3.10 setBranchObject	44
4.14.3.11 encodeBcps	45
4.14.4 Member Data Documentation	45
4.14.4.1 branchObject_	45
4.15 BcpsVariable Class Reference	45
4.15.1 Detailed Description	45
4.15.2 Constructor & Destructor Documentation	45
4.15.2.1 BcpsVariable	45
4.15.2.2 BcpsVariable	46
4.15.2.3 ~BcpsVariable	46
4.15.2.4 BcpsVariable	46
4.16 BcpsVariablePool Class Reference	46
4.16.1 Detailed Description	46
4.16.2 Member Function Documentation	46
4.16.2.1 getNumVariables	46
4.16.2.2 getVariables	47
4.16.2.3 getVariable	47
Index	49

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>doubleton_action</code>	<code>::action</code>	<code>[external]</code>
<code>forcing_constraint_action</code>	<code>::action</code>	<code>[external]</code>
<code>tripleton_action</code>	<code>::action</code>	<code>[external]</code>
<code>remove_fixed_action</code>	<code>::action</code>	<code>[external]</code>
<code>std::allocator</code>	<code>< T ></code>	
<code>ALPS_PS_STATS</code>	<code>[external]</code>	
<code>AlpsEncoded</code>	<code>[external]</code>	
<code>AlpsKnowledge</code>	<code>[external]</code>	
<code>AlpsModel</code>	<code>[external]</code>	
<code>BcpsModel</code>		20
<code>AlpsSolution</code>	<code>[external]</code>	
<code>BcpsSolution</code>		38
<code>AlpsSubTree</code>	<code>[external]</code>	
<code>BcpsSubTree</code>		41
<code>AlpsTreeNode</code>	<code>[external]</code>	
<code>BcpsTreeNode</code>		42
<code>BcpsObject</code>		30
<code>BcpsConstraint</code>		17
<code>BcpsVariable</code>		45
<code>AlpsKnowledgeBroker</code>	<code>[external]</code>	
<code>AlpsKnowledgeBrokerMPI</code>	<code>[external]</code>	
<code>AlpsKnowledgeBrokerSerial</code>	<code>[external]</code>	
<code>AlpsKnowledgePool</code>	<code>[external]</code>	
<code>AlpsNodePool</code>	<code>[external]</code>	
<code>AlpsSolutionPool</code>	<code>[external]</code>	
<code>AlpsSubTreePool</code>	<code>[external]</code>	
<code>BcpsObjectPool</code>		36
<code>BcpsConstraintPool</code>		18
<code>BcpsVariablePool</code>		46
<code>AlpsNodeDesc</code>	<code>[external]</code>	
<code>BcpsNodeDesc</code>		23
<code>AlpsNodeSelection</code>	<code>[external]</code>	
<code>AlpsNodeSelectionBest</code>	<code>[external]</code>	

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]	
std::array< T >	
std::auto_ptr< T >	
std::basic_string< Char >	
std::string	
std::wstring	
std::basic_string< char >	
std::basic_string< wchar_t >	
BcpsBranchObject	13
BcpsBranchStrategy	13
BcpsFieldListMod< T >	19
BcpsFieldListMod< double >	19
BcpsFieldListMod< int >	19
BcpsObjectListMod	35
std::bitset< Bits >	
BitVector128 [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]	
CoinExternalVectorFirstGreater_2< class, class, class > [external]	
CoinExternalVectorFirstGreater_3< class, class, class, class > [external]	
CoinExternalVectorFirstLess_2< class, class, class > [external]	
CoinExternalVectorFirstLess_3< class, class, class, class > [external]	
CoinFactorization [external]	
CoinFileIOBase [external]	

```

CoinFileInput [external]
CoinFileOutput [external]
CoinFirstAbsGreater_2< class, class > [external]
CoinFirstAbsGreater_3< class, class, class > [external]
CoinFirstAbsLess_2< class, class > [external]
CoinFirstAbsLess_3< class, class, class > [external]
CoinFirstGreater_2< class, class > [external]
CoinFirstGreater_3< class, class, class > [external]
CoinFirstLess_2< class, class > [external]
CoinFirstLess_3< class, class, class > [external]
CoinLpIO::CoinHashLink [external]
CoinMpsIO::CoinHashLink [external]
CoinIndexedVector [external]
    CoinPartitionedVector [external]
CoinLpIO [external]
CoinMessageHandler [external]
CoinMessages [external]
    AlpsMessage [external]

    BcpsMessage ..... 20
    CoinMessage [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]
CoinPrePostsolveMatrix [external]
    CoinPostsolveMatrix [external]
    CoinPresolveMatrix [external]
CoinPresolveAction [external]
    do_tighten_action [external]
    doubleton_action [external]
    drop_empty_cols_action [external]
    drop_empty_rows_action [external]
    drop_zero_coefficients_action [external]
    dupcol_action [external]
    duprow3_action [external]
    duprow_action [external]
    forcing_constraint_action [external]
    gubrow_action [external]

```

```

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

```

```

std::string::const_iterator
std::wstring::const_iterator
std::deque< T >::const_iterator
std::list< T >::const_iterator
std::forward_list< T >::const_iterator
std::map< K, T >::const_iterator
std::unordered_map< K, T >::const_iterator
std::multimap< K, T >::const_iterator
std::multimap< K, T >::const_reverse_iterator
std::unordered_multimap< K, T >::const_reverse_iterator
std::set< K >::const_reverse_iterator
std::multiset< K >::const_reverse_iterator
std::unordered_multiset< K >::const_reverse_iterator
std::vector< T >::const_reverse_iterator
std::unordered_set< K >::const_reverse_iterator
std::basic_string< Char >::const_reverse_iterator
std::string::const_reverse_iterator
std::wstring::const_reverse_iterator
std::deque< T >::const_reverse_iterator
std::list< T >::const_reverse_iterator
std::forward_list< T >::const_reverse_iterator
std::map< K, T >::const_reverse_iterator
std::unordered_map< K, T >::const_reverse_iterator
DeletePtrObject[external]
std::deque< T >
dropped_zero[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 >
std::ios_base
    basic_ios< char >
    basic_ios< wchar_t >
    std::basic_ios
        basic_istream< char >
        basic_istream< wchar_t >
        basic_ostream< char >

```

```

basic_ostream< wchar_t >
std::basic_istream
    basic_ifstream< char >
    basic_ifstream< wchar_t >
    basic_iostream< char >
    basic_iostream< wchar_t >
    basic_istreamstream< char >
    basic_istreamstream< wchar_t >
std::basic_ifstream
    std::ifstream
    std::wifstream
std::basic_iostream
    basic_fstream< char >
    basic_fstream< wchar_t >
    basic_stringstream< char >
    basic_stringstream< wchar_t >
std::basic_fstream
    std::fstream
    std::wfstream
std::basic_stringstream
    std::stringstream
    std::wstringstream
std::basic_istreamstream
    std::istreamstream
    std::wistreamstream
std::istream
std::wistream
std::basic_ostream
    basic_iostream< char >
    basic_iostream< wchar_t >
    basic_ofstream< char >
    basic_ofstream< wchar_t >
    basic_ostringstream< char >
    basic_ostringstream< wchar_t >
std::basic_iostream
std::basic_ofstream
    std::ofstream
    std::wofstream
std::basic_ostringstream
    std::ostringstream
    std::wostringstream
std::ostream
std::wostream
std::ios
std::wios
std::unordered_multimap< K, T >::iterator
std::deque< T >::iterator
std::unordered_multiset< K >::iterator
std::set< K >::iterator
std::multimap< K, T >::iterator
std::map< K, T >::iterator
std::multiset< K >::iterator
std::forward_list< T >::iterator
std::wstring::iterator

```

```

std::unordered_set< K >::iterator
std::string::iterator
std::vector< T >::iterator
std::basic_string< Char >::iterator
std::list< T >::iterator
std::unordered_map< K, T >::iterator
std::list< T >
std::map< K, T >
std::map< AlpsKnowledgeType, AlpsKnowledgePool * >
std::multimap< K, T >
std::multiset< K >
presolvehlink[external]
std::priority_queue< T >
std::queue< T >
Coin::ReferencedObject[external]
std::string::reverse_iterator
std::multimap< K, T >::reverse_iterator
std::map< K, T >::reverse_iterator
std::vector< T >::reverse_iterator
std::wstring::reverse_iterator
std::basic_string< Char >::reverse_iterator
std::unordered_map< K, T >::reverse_iterator
std::list< T >::reverse_iterator
std::unordered_multimap< K, T >::reverse_iterator
std::multiset< K >::reverse_iterator
std::deque< T >::reverse_iterator
std::set< K >::reverse_iterator
std::forward_list< T >::reverse_iterator
std::unordered_set< K >::reverse_iterator
std::unordered_multiset< K >::reverse_iterator
std::set< K >
std::smart_ptr< T >
Coin::SmartPtr< T >[external]
std::stack< T >
symrec[external]
std::system_error
std::thread
TotalWorkload[external]
std::unique_ptr< T >
std::unordered_map< K, T >
std::unordered_multimap< K, T >
std::unordered_multiset< K >
std::unordered_set< K >
std::valarray< T >
std::vector< T >
std::vector< AlpsKnowledge * >
std::vector< BcpsConstraint * >
std::vector< BcpsVariable * >
std::vector< double >
std::vector< std::pair< std::string, AlpsParameter > >
std::vector< std::string >
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:

BcpsBranchObject	
BcpsBranchObject contains the member data required when choosing branching entities and executing actual branching	13
BcpsBranchStrategy	
Branching strategy specifies: (1) how to select a candidate set of branching objects (2) how to compare two branching objects	13
BcpsConstraint	17
BcpsConstraintPool	18
BcpsFieldListMod< T >	
This class contains modifications for a single <code>std::vector<T></code> object	19
BcpsMessage	20
BcpsModel	20
BcpsNodeDesc	
For a given type, the <code>objectVecStorage_</code> structure holds the description	23
BcpsObject	
A class for describing the objects that comprise a BCPS subproblem	30
BcpsObjectListMod	
Here is the set of <code>vectorMod_</code> objects that represent the list of objects of a particular type (either in relative or explicit form)	35
BcpsObjectPool	
Object pool is used to store objects	36
BcpsSolution	
This class holds the solution objects	38
BcpsSubTree	
This class is the data structure for storing a subtree within BCPS	41
BcpsTreeNode	
This class contain the data for a BCPS search tree node	42
BcpsVariable	45
BcpsVariablePool	46

Chapter 3

File Index

3.1 File List

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

Bcps.h	??
BcpsBranchObject.h	??
BcpsBranchStrategy.h	??
BcpsConfig.h	??
BcpsInterface.h	??
BcpsLicense.h	??
BcpsMessage.h	??
BcpsModel.h	??
BcpsNodeDesc.h	??
BcpsObject.h	??
BcpsObjectPool.h	??
BcpsSolution.h	??
BcpsSubTree.h	??
BcpsTreeNode.h	??
config_bcps_default.h	??
config_default.h	??

Chapter 4

Class Documentation

4.1 BcpsBranchObject Class Reference

[BcpsBranchObject](#) contains the member data required when choosing branching entities and excuting actual branching.

```
#include <BcpsBranchObject.h>
```

Collaboration diagram for BcpsBranchObject:

4.2 BcpsBranchStrategy Class Reference

Branching strategy specifies: (1) how to select a candidate set of branching objects (2) how to compare two branching objects.

```
#include <BcpsBranchStrategy.h>
```

Collaboration diagram for BcpsBranchStrategy:

Public Member Functions

- [BcpsBranchStrategy](#) ()
Default Constructor.
- [BcpsBranchStrategy](#) (BcpsModel *m)
Useful Constructor.
- virtual [~BcpsBranchStrategy](#) ()
Destructor.
- virtual [BcpsBranchStrategy](#) * [clone](#) () const =0
Clone a branch strategy.
- int [getType](#) ()
Get type.
- void [setType](#) (int t)
Set type.
- void [setModel](#) (BcpsModel *m)
Set model.
- virtual void [clearBest](#) (BcpsModel *model)

- Clear branching strategy environment before starting a new round of selecting the best branch object.*

 - virtual int [createCandBranchObjects](#) (int numPassesLeft, double ub)

Create a set of candidate branching objects.
- virtual int [betterBranchObject](#) ([BcpsBranchObject](#) *b, [BcpsBranchObject](#) *bestSoFar)=0

Compare branching object thisOne to bestSoFar.
- virtual [BcpsBranchObject](#) * [bestBranchObject](#) ()

Compare branching objects in branchObjects_.
- int [getNumBranchObjects](#) ()

Set/get branching objects.

Protected Attributes

- int [type_](#)

Type of branching strategy.
- [BcpsModel](#) * [model_](#)

Pointer to model.
- int [numBranchObjects_](#)

Following members are used to store candidate branching objects.
- [BcpsBranchObject](#) ** [branchObjects_](#)

The set of candidate branching objects.
- [BcpsBranchObject](#) * [bestBranchObject_](#)

Following members are used to store information about best branching object found so far.
- double [bestChangeUp_](#)

Change up for best.
- int [bestNumberUp_](#)

Number of infeasibilities for up.
- double [bestChangeDown_](#)

Change down for best.
- int [bestNumberDown_](#)

Number of infeasibilities for down.

4.2.1 Detailed Description

Branching strategy specifies: (1) how to select a candidate set of branching objects (2) how to compare two branching objects.

Definition at line 39 of file BcpsBranchStrategy.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 [BcpsBranchStrategy::BcpsBranchStrategy](#) () [inline]

Default Constructor.

Definition at line 83 of file BcpsBranchStrategy.h.

4.2.2.2 BcpsBranchStrategy::BcpsBranchStrategy (BcpsModel * *m*) [inline]

Useful Constructor.

Definition at line 95 of file BcpsBranchStrategy.h.

4.2.2.3 virtual BcpsBranchStrategy::~BcpsBranchStrategy () [inline],[virtual]

Destructor.

Definition at line 107 of file BcpsBranchStrategy.h.

4.2.3 Member Function Documentation

4.2.3.1 virtual BcpsBranchStrategy* BcpsBranchStrategy::clone () const [pure virtual]

Clone a branch strategy.

4.2.3.2 int BcpsBranchStrategy::getType () [inline]

Get type.

Definition at line 118 of file BcpsBranchStrategy.h.

4.2.3.3 void BcpsBranchStrategy::setType (int *t*) [inline]

Set type.

Definition at line 121 of file BcpsBranchStrategy.h.

4.2.3.4 void BcpsBranchStrategy::setModel (BcpsModel * *m*) [inline]

Set model.

Definition at line 124 of file BcpsBranchStrategy.h.

4.2.3.5 int BcpsBranchStrategy::getNumBranchObjects () [inline]

Set/get branching objects.

Definition at line 128 of file BcpsBranchStrategy.h.

4.2.3.6 virtual void BcpsBranchStrategy::clearBest (BcpsModel * *model*) [inline],[virtual]

Clear branching strategy environment before starting a new round of selecting the best branch object.

Definition at line 138 of file BcpsBranchStrategy.h.

4.2.3.7 virtual int BcpsBranchStrategy::createCandBranchObjects (int *numPassesLeft*, double *ub*) [inline],[virtual]

Create a set of candidate branching objects.

Definition at line 147 of file BcpsBranchStrategy.h.

4.2.3.8 `virtual int BcpsBranchStrategy::betterBranchObject (BcpsBranchObject * b, BcpsBranchObject * bestSoFar)`
`[pure virtual]`

Compare branching object thisOne to bestSoFar.

If thisOne is better than bestObject, return branching direction(1 or -1), otherwise return 0. If bestSoFar is NULL, then always return branching direction(1 or -1).

4.2.3.9 `virtual BcpsBranchObject* BcpsBranchStrategy::bestBranchObject ()` `[virtual]`

Compare branching objects in branchObjects_.

Return the index of the best branching object. Also, set branch direction in the best object.

4.2.4 Member Data Documentation

4.2.4.1 `int BcpsBranchStrategy::type_` `[protected]`

Type of branching strategy.

Definition at line 48 of file BcpsBranchStrategy.h.

4.2.4.2 `BcpsModel* BcpsBranchStrategy::model_` `[protected]`

Pointer to model.

Definition at line 51 of file BcpsBranchStrategy.h.

4.2.4.3 `int BcpsBranchStrategy::numBranchObjects_` `[protected]`

Following members are used to store candidate branching objects.

NOTE: They are required to be cleared before starting another round of selecting. Number of candidate branching objects.

Definition at line 58 of file BcpsBranchStrategy.h.

4.2.4.4 `BcpsBranchObject** BcpsBranchStrategy::branchObjects_` `[protected]`

The set of candidate branching objects.

Definition at line 60 of file BcpsBranchStrategy.h.

4.2.4.5 `BcpsBranchObject* BcpsBranchStrategy::bestBranchObject_` `[protected]`

Following members are used to store information about best branching object found so far.

NOTE: They are required to be cleared before starting another round of selecting. Best branching object found so far.

Definition at line 69 of file BcpsBranchStrategy.h.

4.2.4.6 `double BcpsBranchStrategy::bestChangeUp_` [protected]

Change up for best.

Definition at line 71 of file `BcpsBranchStrategy.h`.

4.2.4.7 `int BcpsBranchStrategy::bestNumberUp_` [protected]

Number of infeasibilities for up.

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

4.2.4.8 `double BcpsBranchStrategy::bestChangeDown_` [protected]

Change down for best.

Definition at line 75 of file `BcpsBranchStrategy.h`.

4.2.4.9 `int BcpsBranchStrategy::bestNumberDown_` [protected]

Number of infeasibilities for down.

Definition at line 77 of file `BcpsBranchStrategy.h`.

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

- `BcpsBranchStrategy.h`

4.3 BcpsConstraint Class Reference

Inheritance diagram for `BcpsConstraint`:

Collaboration diagram for `BcpsConstraint`:

Public Member Functions

- [BcpsConstraint](#) ()
Default constructor.
- [BcpsConstraint](#) (double lbh, double ubh, double lbs, double ubs)
Useful constructor.
- virtual [~BcpsConstraint](#) ()
Destructor constructor.
- [BcpsConstraint](#) (const [BcpsConstraint](#) &rhs)
Copy constructor.

Additional Inherited Members

4.3.1 Detailed Description

Definition at line 355 of file `BcpsObject.h`.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 `BcpsConstraint::BcpsConstraint ()` `[inline]`

Default constructor.

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

4.3.2.2 `BcpsConstraint::BcpsConstraint (double lbh, double ubh, double lbs, double ubs)` `[inline]`

Useful constructor.

Definition at line 362 of file `BcpsObject.h`.

4.3.2.3 `virtual BcpsConstraint::~BcpsConstraint ()` `[inline]`, `[virtual]`

Destructor constructor.

Definition at line 368 of file `BcpsObject.h`.

4.3.2.4 `BcpsConstraint::BcpsConstraint (const BcpsConstraint & rhs)` `[inline]`

Copy constructor.

Definition at line 371 of file `BcpsObject.h`.

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

- `BcpsObject.h`

4.4 BcpsConstraintPool Class Reference

Inheritance diagram for `BcpsConstraintPool`:

Collaboration diagram for `BcpsConstraintPool`:

Public Member Functions

- void `addConstraint` (`BcpsConstraint` *con)
Add a constraint to pool.
- void `deleteConstraint` (int k)
Delete constraint k from pool.
- int `getNumConstraints` () const
Query how many constraints are in the pool.
- const std::vector< `AlpsKnowledge` * > & `getConstraints` () const
Get the vector of constraints.
- `AlpsKnowledge` * `getConstraint` (int k) const
Get a constraints.

4.4.1 Detailed Description

Definition at line 104 of file BcpsObjectPool.h.

4.4.2 Member Function Documentation

4.4.2.1 int BcpsConstraintPool::getNumConstraints () const [inline]

Query how many constraints are in the pool.

Definition at line 116 of file BcpsObjectPool.h.

4.4.2.2 const std::vector<AlpsKnowledge*>& BcpsConstraintPool::getConstraints () const [inline]

Get the vector of constraints.

Definition at line 119 of file BcpsObjectPool.h.

4.4.2.3 AlpsKnowledge* BcpsConstraintPool::getConstraint (int k) const [inline]

Get a constraints.

Definition at line 122 of file BcpsObjectPool.h.

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

- BcpsObjectPool.h

4.5 BcpsFieldListMod< T > Struct Template Reference

This class contains modifications for a single `std::vector<T>` object.

```
#include <BcpsNodeDesc.h>
```

Inheritance diagram for BcpsFieldListMod< T >:

Public Attributes

- bool [relative](#)

How the modification is stored, explicit means complete replacement, relative means relative to some other explicit object vector ("explicit" is a key word so we can't name the field that.)

- int [numModify](#)

The number of entries to be modified.

- int * [posModify](#)

The positions to be modified.

- T * [entries](#)

Values.

4.5.1 Detailed Description

```
template<class T>struct BcpsFieldListMod< T >
```

This class contains modifications for a single `std::vector<T>` object.

Definition at line 39 of file `BcpsNodeDesc.h`.

4.5.2 Member Data Documentation

4.5.2.1 `template<class T> int BcpsFieldListMod< T >::numModify`

The number of entries to be modified.

Definition at line 47 of file `BcpsNodeDesc.h`.

4.5.2.2 `template<class T> int* BcpsFieldListMod< T >::posModify`

The positions to be modified.

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

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

- `BcpsNodeDesc.h`

4.6 BcpsMessage Class Reference

Inheritance diagram for `BcpsMessage`:

Collaboration diagram for `BcpsMessage`:

Public Member Functions

Constructors etc

- [BcpsMessage](#) (**Language** `language=us_en`)
Constructor.

4.6.1 Detailed Description

Definition at line 43 of file `BcpsMessage.h`.

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

- `BcpsMessage.h`

4.7 BcpsModel Class Reference

Inheritance diagram for `BcpsModel`:

Collaboration diagram for `BcpsModel`:

Public Member Functions

- `std::vector< BcpsVariable * > getVariables () const`
Return list of variables.
- `std::vector< BcpsConstraint * > getConstraints () const`
Return list of constraints.
- `CoinMessageHandler * bcpsMessageHandler () const`
Get the message handler.
- `CoinMessages bcpsMessages ()`
Return messages.
- `AlpsReturnStatus encodeBcps (AlpsEncoded *encoded) const`
Pack Bcps portion of model into an encoded object.
- `AlpsReturnStatus decodeBcps (AlpsEncoded &encoded)`
Unpack Bcps portion of model from an encoded object.
- `std::vector< BcpsConstraint * > & getConstraints ()`
Get variables and constraints.
- `void setConstraints (BcpsConstraint **con, int size)`
Set variables and constraints.

Protected Attributes

- `std::vector< BcpsConstraint * > constraints_`
Constraints input by users (before preprocessing).
- `std::vector< BcpsVariable * > variables_`
Variables input by users (before preprocessing).
- `int numCoreConstraints_`
Number of core constraints.
- `int numCoreVariables_`
Number of core variables.
- `CoinMessageHandler * bcpsMessageHandler_`
Message handler.
- `CoinMessages bcpsMessages_`
Bcps messages.

4.7.1 Detailed Description

Definition at line 40 of file BcpsModel.h.

4.7.2 Member Function Documentation

4.7.2.1 `std::vector<BcpsVariable*> BcpsModel::getVariables () const` `[inline]`

Return list of variables.

Definition at line 117 of file BcpsModel.h.

4.7.2.2 `std::vector<BcpsConstraint*> BcpsModel::getConstrints () const` [inline]

Return list of constraints.

Definition at line 120 of file BcpsModel.h.

4.7.2.3 `CoinMessageHandler* BcpsModel::bcpsMessageHandler () const` [inline]

Get the message handler.

Definition at line 123 of file BcpsModel.h.

4.7.2.4 `CoinMessages BcpsModel::bcpsMessages ()` [inline]

Return messages.

Definition at line 127 of file BcpsModel.h.

4.7.2.5 `AlpsReturnStatus BcpsModel::encodeBcps (AlpsEncoded * encoded) const`

Pack Bcps portion of model into an encoded object.

4.7.2.6 `AlpsReturnStatus BcpsModel::decodeBcps (AlpsEncoded & encoded)`

Unpack Bcps portion of model from an encoded object.

4.7.3 Member Data Documentation

4.7.3.1 `std::vector<BcpsConstraint*> BcpsModel::constraints_` [protected]

Constraints input by users (before preprocessing).

Definition at line 45 of file BcpsModel.h.

4.7.3.2 `std::vector<BcpsVariable*> BcpsModel::variables_` [protected]

Variables input by users (before preprocessing).

Definition at line 48 of file BcpsModel.h.

4.7.3.3 `int BcpsModel::numCoreConstraints_` [protected]

Number of core constraints.

By default, all input constraints are core.

Definition at line 54 of file BcpsModel.h.

4.7.3.4 `int BcpsModel::numCoreVariables_` [protected]

Number of core variables.

By default, all input variables are core.

Definition at line 57 of file BcpsModel.h.

4.7.3.5 CoinMessageHandler* BcpsModel::bcpsMessageHandler_ [protected]

Message handler.

Definition at line 60 of file BcpsModel.h.

4.7.3.6 CoinMessages BcpsModel::bcpsMessages_ [protected]

Bcps messages.

Definition at line 63 of file BcpsModel.h.

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

- BcpsModel.h

4.8 BcpsNodeDesc Class Reference

For a given type, the objectVecStorage_ structure holds the description.

```
#include <BcpsNodeDesc.h>
```

Inheritance diagram for BcpsNodeDesc:

Collaboration diagram for BcpsNodeDesc:

Public Member Functions

- [BcpsNodeDesc](#) ()
Default constructor.
- [BcpsNodeDesc](#) ([BcpsModel](#) *m)
Useful constructor.
- virtual [~BcpsNodeDesc](#) ()
Destructor.
- void [initToNull](#) ()
Initialize member data.
- void [setVars](#) (int numRem, const int *posRem, int numAdd, const [BcpsObject](#) **objects, bool relvlh, int numvlh, const int *vlhp, const double *vlhe, bool relvuh, int numvuh, const int *vuhp, const double *vuhe, bool relvls, int numvls, const int *vlsp, const double *vlse, bool relvus, int numvus, const int *vusp, const double *vuse)
Set variable objects.
- void [assignVars](#) (int numRem, int *&posRem, int numAdd, [BcpsObject](#) **&objects, bool relvlh, int numvlh, int *&vlhp, double *&vlhe, bool relvuh, int numvuh, int *&vuhp, double *&vuhe, bool relvls, int numvls, int *&vlsp, double *&vlse, bool relvus, int numvus, int *&vusp, double *&vuse)
Assign variable objects.
- void [setCons](#) (int numRem, const int *posRem, int numAdd, const [BcpsObject](#) **objects, bool relclh, int numclh, const int *clhp, const double *clhe, bool relcu, int numcu, const int *cuhp, const double *cuhe, bool relcls, int numcls, const int *clsp, const double *clse, bool relcus, int numcus, const int *cusp, const double *cuse)
Set constraint objects.

- void [assignCons](#) (int numRem, int *&posRem, int numAdd, [BcpsObject](#) **&objects, bool relclh, int numclh, int *&clhp, double *&clhe, bool relcu, int numcu, int *&cuhp, double *&cuhe, bool relcls, int numcls, int *&clsp, double *&clse, bool relcus, int numcus, int *&cus, double *&cuse)

Assign constraint objects.
- [BcpsObjectListMod](#) * [getVars](#) () const

Get variable objects.
- [BcpsObjectListMod](#) * [getCons](#) () const

Get constraint objects.
- [BcpsObjectListMod](#) * [vars](#) ()

Accesss variables.
- [BcpsObjectListMod](#) * [cons](#) ()

Accesss constraints.
- void [assignVarSoftBound](#) (int numModSoftVarLB, int *&varLBi, double *&varLBv, int numModSoftVarUB, int *&varUBi, double *&varUBv)

Set variable soft bounds.
- void [setVarSoftBound](#) (int numModSoftVarLB, const int *varLBi, const double *varLBv, int numModSoftVarUB, const int *varUBi, const double *varUBv)

Set variable soft bounds.
- void [assignVarHardBound](#) (int numModHardVarLB, int *&varLBi, double *&varLBv, int numModHardVarUB, int *&varUBi, double *&varUBv)

Set variable hard bounds.
- void [setConSoftBound](#) (int numModSoftConLB, const int *conLBi, const double *conLBv, int numModSoftConUB, const int *conUBi, const double *conUBv)

Set constraint soft bounds.
- void [setVarHardBound](#) (int numModHardVarLB, const int *varLBi, const double *varLBv, int numModHardVarUB, const int *varUBi, const double *varUBv)

Set variable hard bounds.
- void [setConHardBound](#) (int numModHardConLB, const int *conLBi, const double *conLBv, int numModHardConUB, const int *conUBi, const double *conUBv)

Set constraint hard bounds.
- void [appendAddedConstraints](#) (int numAdd, [BcpsObject](#) **addCons)

Recode the added constraints.
- void [setAddedConstraints](#) (int numAdd, [BcpsObject](#) **addCons)

Recode the added constraints.
- void [delConstraints](#) (int numDel, int *indices)

Record the constraints are deleted.
- void [addVariables](#) (int numAdd, [BcpsObject](#) **addVars)

Record added variables.
- void [delVariables](#) (int numDel, int *indices)

Record deleted variables.
- [AlpsReturnStatus](#) [encodeBcps](#) ([AlpsEncoded](#) *encoded) const

Pack bcps node description into an encoded.
- [AlpsReturnStatus](#) [decodeBcps](#) ([AlpsEncoded](#) &encoded)

Unpack bcps node description into an encoded.

Protected Member Functions

- AlpsReturnStatus [encodeDbfFieldMods](#) (**AlpsEncoded** *encoded, [BcpsFieldListMod](#)< double > *field) const
Pack a double field into an encoded object.
- AlpsReturnStatus [encodeIntFieldMods](#) (**AlpsEncoded** *encoded, [BcpsFieldListMod](#)< int > *field) const
Pack a integer field into an encoded object.
- AlpsReturnStatus [encodeObjectMods](#) (**AlpsEncoded** *encoded, [BcpsObjectListMod](#) *objMod) const
Pack object modifications to an encoded object.
- AlpsReturnStatus [decodeDbfFieldMods](#) (**AlpsEncoded** &encoded, [BcpsFieldListMod](#)< double > *field)
Unpack a double field from an encoded object.
- AlpsReturnStatus [decodeIntFieldMods](#) (**AlpsEncoded** &encoded, [BcpsFieldListMod](#)< int > *field)
Unpack a integer field from an encoded object.
- AlpsReturnStatus [decodeObjectMods](#) (**AlpsEncoded** &encoded, [BcpsObjectListMod](#) *objMod)
Unpack object modifications to an encoded object.

Protected Attributes

- [BcpsObjectListMod](#) * vars_
Variable objects.
- [BcpsObjectListMod](#) * cons_
Constraint objects.

4.8.1 Detailed Description

For a given type, the `objectVecStorage_` structure holds the description.

This description is explicit if the `numRemoved` member is -1. In this case, the `objects` member holds the full list of objects of the given type. If an explicit description of the parent is given together with a relative description of the current node then the process of reconstructing the explicit description of the current node is as follows:

1. initialize the explicit list **L** to be the parent's list;
2. remove the objects on the indicated positions from **L**;
3. modify the appropriate members of the objects in **L** according to the `vectorMod_` members of the `objectVecStorage_` structure;
4. append the objects to be added to **L**.

If the `numRemoved` field is -1, that means that the current description is explicit. In this case, the contents of the `vectorMod_` members are largely irrelevant, except for the `relative` fields which indicate whether the appropriate `vectorMod_` member can ever be expressed as a relative description or not.

If the `numRemoved` field is ≥ 0 , then the current description is considered relative even if every `vectorMod_` member contains explicit data. The description of a node can be either explicit or relative to its parent. In the node there are a number of object types and for each object type the explicit/relative description is considered separately.

If the information on an object type is relative, it means that at least one of the fields (`lbHard`, `ubHard`, etc.) has a relative description.

Definition at line 128 of file `BcpsNodeDesc.h`.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 BcpsNodeDesc::BcpsNodeDesc () [inline]

Default constructor.

Definition at line 141 of file BcpsNodeDesc.h.

4.8.2.2 BcpsNodeDesc::BcpsNodeDesc (BcpsModel * m) [inline]

Useful constructor.

Definition at line 144 of file BcpsNodeDesc.h.

4.8.2.3 virtual BcpsNodeDesc::~BcpsNodeDesc () [virtual]

Destructor.

4.8.3 Member Function Documentation

4.8.3.1 void BcpsNodeDesc::initToNull ()

Initialize member data.

4.8.3.2 void BcpsNodeDesc::setVars (int *numRem*, const int * *posRem*, int *numAdd*, const BcpsObject ** *objects*, bool *relvlh*, int *numvlh*, const int * *vlhp*, const double * *vlhe*, bool *relvuh*, int *numvuh*, const int * *vuhp*, const double * *vuhe*, bool *relvls*, int *numvls*, const int * *vlsp*, const double * *vlse*, bool *relvus*, int *numvus*, const int * *vusp*, const double * *vuse*)

Set variable objects.

4.8.3.3 void BcpsNodeDesc::assignVars (int *numRem*, int * & *posRem*, int *numAdd*, BcpsObject ** & *objects*, bool *relvlh*, int *numvlh*, int * & *vlhp*, double * & *vlhe*, bool *relvuh*, int *numvuh*, int * & *vuhp*, double * & *vuhe*, bool *relvls*, int *numvls*, int * & *vlsp*, double * & *vlse*, bool *relvus*, int *numvus*, int * & *vusp*, double * & *vuse*)

Assign variable objects.

Take over memory ownership.

4.8.3.4 void BcpsNodeDesc::setCons (int *numRem*, const int * *posRem*, int *numAdd*, const BcpsObject ** *objects*, bool *relclh*, int *numclh*, const int * *clhp*, const double * *clhe*, bool *relcu*, int *numcu*, const int * *cuhp*, const double * *cuhe*, bool *relcls*, int *numcls*, const int * *clsp*, const double * *clse*, bool *relcus*, int *numcus*, const int * *cus*, const double * *cuse*)

Set constraint objects.

4.8.3.5 void BcpsNodeDesc::assignCons (int *numRem*, int * & *posRem*, int *numAdd*, BcpsObject ** & *objects*, bool *relclh*, int *numclh*, int * & *clhp*, double * & *clhe*, bool *relcu*, int *numcu*, int * & *cuhp*, double * & *cuhe*, bool *relcls*, int *numcls*, int * & *clsp*, double * & *clse*, bool *relcus*, int *numcus*, int * & *cus*, double * & *cuse*)

Assign constraint objects.

Take over memory ownership.

4.8.3.6 BcpsObjectListMod* BcpsNodeDesc::getVars () const [inline]

Get variable objects.

Definition at line 243 of file BcpsNodeDesc.h.

4.8.3.7 BcpsObjectListMod* BcpsNodeDesc::getCons () const [inline]

Get constraint objects.

Definition at line 246 of file BcpsNodeDesc.h.

4.8.3.8 BcpsObjectListMod* BcpsNodeDesc::vars () [inline]

Accesss variables.

Definition at line 249 of file BcpsNodeDesc.h.

4.8.3.9 BcpsObjectListMod* BcpsNodeDesc::cons () [inline]

Accesss constraints.

Definition at line 252 of file BcpsNodeDesc.h.

4.8.3.10 void BcpsNodeDesc::assignVarSoftBound (int numModSoftVarLB, int *& varLBi, double *& varLBv, int numModSoftVarUB, int *& varUBi, double *& varUBv)

Set variable soft bounds.

Take ownerships of arraies.

4.8.3.11 void BcpsNodeDesc::setVarSoftBound (int numModSoftVarLB, const int * varLBi, const double * varLBv, int numModSoftVarUB, const int * varUBi, const double * varUBv)

Set variable soft bounds.

Don't take ownerships of arraies.

4.8.3.12 void BcpsNodeDesc::assignVarHardBound (int numModHardVarLB, int *& varLBi, double *& varLBv, int numModHardVarUB, int *& varUBi, double *& varUBv)

Set variable hard bounds.

Take ownerships of arraies.

4.8.3.13 void BcpsNodeDesc::setConSoftBound (int numModSoftConLB, const int * conLBi, const double * conLBv, int numModSoftConUB, const int * conUBi, const double * conUBv)

Set constraint soft bounds.

Don't take ownerships of arraies.

4.8.3.14 `void BcpsNodeDesc::setVarHardBound (int numModHardVarLB, const int * varLBi, const double * varLBv, int numModHardVarUB, const int * varUBi, const double * varUBv)`

Set variable hard bounds.

Don't take ownerships of arrays.

4.8.3.15 `void BcpsNodeDesc::setConHardBound (int numModHardConLB, const int * conLBi, const double * conLBv, int numModHardConUB, const int * conUBi, const double * conUBv)`

Set constraint hard bounds.

Don't take ownerships of arrays.

4.8.3.16 `void BcpsNodeDesc::appendAddedConstraints (int numAdd, BcpsObject ** addCons) [inline]`

Record the added constraints.

Take over the memory ownership of arguments. Append to previous constraints.

Definition at line 304 of file BcpsNodeDesc.h.

4.8.3.17 `void BcpsNodeDesc::setAddedConstraints (int numAdd, BcpsObject ** addCons) [inline]`

Record the added constraints.

Take over the memory ownership of arguments. Delete already added constraints.

Definition at line 323 of file BcpsNodeDesc.h.

4.8.3.18 `void BcpsNodeDesc::delConstraints (int numDel, int * indices) [inline]`

Record the constraints are deleted.

Take over the memory ownership of arguments.

Definition at line 337 of file BcpsNodeDesc.h.

4.8.3.19 `void BcpsNodeDesc::addVariables (int numAdd, BcpsObject ** addVars) [inline]`

Record added variables.

Take over the memory ownership of arguments.

Definition at line 344 of file BcpsNodeDesc.h.

4.8.3.20 `void BcpsNodeDesc::delVariables (int numDel, int * indices) [inline]`

Record deleted variables.

Take over the memory ownership of arguments.

Definition at line 357 of file BcpsNodeDesc.h.

4.8.3.21 `AlpsReturnStatus BcpsNodeDesc::encodeDbfFieldMods (AlpsEncoded * encoded, BcpsFieldListMod< double > * field) const [protected]`

Pack a double field into an encoded object.

4.8.3.22 `AlpsReturnStatus BcpsNodeDesc::encodeIntFieldMods (AlpsEncoded * encoded, BcpsFieldListMod< int > * field) const [protected]`

Pack a integer field into an encoded object.

4.8.3.23 `AlpsReturnStatus BcpsNodeDesc::encodeObjectMods (AlpsEncoded * encoded, BcpsObjectListMod * objMod) const [protected]`

Pack object modifications to an encoded object.

4.8.3.24 `AlpsReturnStatus BcpsNodeDesc::decodeDbfFieldMods (AlpsEncoded & encoded, BcpsFieldListMod< double > * field) [protected]`

Unpack a double field from an encoded object.

4.8.3.25 `AlpsReturnStatus BcpsNodeDesc::decodeIntFieldMods (AlpsEncoded & encoded, BcpsFieldListMod< int > * field) [protected]`

Unpack a integer field from an encoded object.

4.8.3.26 `AlpsReturnStatus BcpsNodeDesc::decodeObjectMods (AlpsEncoded & encoded, BcpsObjectListMod * objMod) [protected]`

Unpack object modifications to an encoded object.

4.8.3.27 `AlpsReturnStatus BcpsNodeDesc::encodeBcps (AlpsEncoded * encoded) const`

Pack bcps node description into an encoded.

4.8.3.28 `AlpsReturnStatus BcpsNodeDesc::decodeBcps (AlpsEncoded & encoded)`

Unpack bcps node description into an encoded.

4.8.4 Member Data Documentation

4.8.4.1 `BcpsObjectListMod* BcpsNodeDesc::vars_ [protected]`

Variable objects.

Definition at line 133 of file BcpsNodeDesc.h.

4.8.4.2 BcpsObjectListMod* BcpsNodeDesc::cons_ [protected]

Constraint objects.

Definition at line 136 of file BcpsNodeDesc.h.

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

- BcpsNodeDesc.h

4.9 BcpsObject Class Reference

A class for describing the objects that comprise a BCPS subproblem.

```
#include <BcpsObject.h>
```

Inheritance diagram for BcpsObject:

Collaboration diagram for BcpsObject:

Public Member Functions

- [BcpsObject](#) (const [BcpsObject](#) &rhs)
Copy constructor.
- [BcpsObject](#) & [operator=](#) (const [BcpsObject](#) &rhs)
Assignment operator.
- virtual [BcpsObject](#) * [clone](#) () const
Clone an entity.
- virtual double [infeasibility](#) ([BcpsModel](#) *m, int &preferredWay) const
Infeasibility of the object This is some measure of the infeasibility of the object.
- virtual void [feasibleRegion](#) ([BcpsModel](#) *m)
Look at the current solution and set bounds to match the solution.
- virtual [BcpsBranchObject](#) * [createBranchObject](#) ([BcpsModel](#) *m, int way) const
Create a branching object and indicate which way to branch first.
- virtual [BcpsBranchObject](#) * [preferredNewFeasible](#) ([BcpsModel](#) *m) const
Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a good direction.
- virtual [BcpsBranchObject](#) * [notPreferredNewFeasible](#) ([BcpsModel](#) *m) const
Given a valid solution (with reduced costs, etc.), return a branching object which would give a new feasible point in a bad direction.
- virtual void [resetBounds](#) ([BcpsModel](#) *m)
Reset variable bounds to their original values.
- virtual bool [boundBranch](#) ([BcpsModel](#) *m) const
Return true if branches created by object will modify variable bounds.
- virtual void [floorCeiling](#) (double &floorValue, double &ceilingValue, double value, double tolerance) const
Returns floor and ceiling i.e.
- virtual double [upEstimate](#) () const
Return "up" estimate.
- virtual double [downEstimate](#) () const
Return "down" estimate.

- virtual AlpsResponseStatus [encode](#) (**AlpsEncoded** *encoded)
Pack into a encode object.
- virtual **AlpsKnowledge** * [decode](#) (**AlpsEncoded** &encoded) const
Decode a constraint from an encoded object.
- int [getObjectIndex](#) () const
Return the value of the appropriate field.
- void [setObjectIndex](#) (int ind)
Set the appropriate property.
- virtual void [hashing](#) (**BcpsModel** *model=NULL)
Hashing.

Protected Member Functions

- AlpsResponseStatus [encodeBcpsObject](#) (**AlpsEncoded** *encoded) const
Pack Bcps part to a encode object.
- AlpsResponseStatus [decodeBcpsObject](#) (**AlpsEncoded** &encoded)
Unpack Bcps part from a encode object.

Protected Attributes

- int [objectIndex_](#)
Global index of this object.
- BcpsObjRep_t [repType_](#)
Core, indexed, or algorithmic.
- BcpsIntegral_t [intType_](#)
The integrality type of the object, i.e., what values it can take up between the specified bounds.
- BcpsValidRegion [validRegion_](#)
Valid in the whole tree or only the subtree rooted at the node that generate this object.
- int [status_](#)
The status of the object.
- double [lbHard_](#)
The lower bound of the object when it was first created.
- double [ubHard_](#)
The upper bound of the object when it was first created.
- double [lbSoft_](#)
The current lower bound of the object.
- double [ubSoft_](#)
The current upper bound of the object.
- double [hashValue_](#)
The hash value of this object.
- int [numInactive_](#)
Number of inactive when in formulation.
- double [effectiveness_](#)
Effectiveness: nonnegative value.

4.9.1 Detailed Description

A class for describing the objects that comprise a BCPS subproblem.

At the BCPS level, all that is assumed about an object is that it has bounds and that it has an integrality type. The concept that an object

Definition at line 76 of file BcpsObject.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 `BcpsObject::BcpsObject (const BcpsObject & rhs) [inline]`

Copy constructor.

Definition at line 159 of file BcpsObject.h.

4.9.3 Member Function Documentation

4.9.3.1 `BcpsObject& BcpsObject::operator= (const BcpsObject & rhs)`

Assignment operator.

4.9.3.2 `virtual BcpsObject* BcpsObject::clone () const [inline],[virtual]`

Clone an entity.

Definition at line 178 of file BcpsObject.h.

4.9.3.3 `virtual double BcpsObject::infeasibility (BcpsModel * m, int & preferredWay) const [inline],[virtual]`

Infeasibility of the object This is some measure of the infeasibility of the object.

It should be scaled to be in the range [0.0, 0.5], with 0.0 indicating the object is satisfied.

The preferred branching direction is returned in preferredWay,

This is used to prepare for strong branching but should also think of case when no strong branching

The object may also compute an estimate of cost of going "up" or "down". This will probably be based on pseudo-cost ideas.

Definition at line 231 of file BcpsObject.h.

4.9.3.4 `virtual void BcpsObject::feasibleRegion (BcpsModel * m) [inline],[virtual]`

Look at the current solution and set bounds to match the solution.

Definition at line 236 of file BcpsObject.h.

4.9.3.5 `virtual BcpsBranchObject* BcpsObject::createBranchObject (BcpsModel * m, int way) const [inline],[virtual]`

Create a branching object and indicate which way to branch first.

The branching object has to know how to create branches (fix variables, etc.)

Definition at line 241 of file BcpsObject.h.

4.9.3.6 `virtual BcpsBranchObject* BcpsObject::preferredNewFeasible (BcpsModel * m) const` `[inline],`
`[virtual]`

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

If the method cannot generate a feasible point (because there aren't any, or because it isn't bright enough to find one), it should return null.

Definition at line 252 of file BcpsObject.h.

4.9.3.7 `virtual BcpsBranchObject* BcpsObject::notPreferredNewFeasible (BcpsModel * m) const` `[inline],`
`[virtual]`

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

If the method cannot generate a feasible point (because there aren't any, or because it isn't bright enough to find one), it should return null.

Definition at line 263 of file BcpsObject.h.

4.9.3.8 `virtual void BcpsObject::resetBounds (BcpsModel * m)` `[inline], [virtual]`

Reset variable bounds to their original values.

Bounds may be tightened, so it may be good to be able to reset them to their original values.

Definition at line 271 of file BcpsObject.h.

4.9.3.9 `virtual bool BcpsObject::boundBranch (BcpsModel * m) const` `[inline], [virtual]`

Return true if branches created by object will modify variable bounds.

Definition at line 275 of file BcpsObject.h.

4.9.3.10 `virtual void BcpsObject::floorCeiling (double & floorValue, double & ceilingValue, double value, double tolerance) const`
`[virtual]`

Returns floor and ceiling i.e.

closest valid points.

4.9.3.11 `virtual double BcpsObject::upEstimate () const` `[inline], [virtual]`

Return "up" estimate.

Default: 1.0e-5.

Definition at line 284 of file BcpsObject.h.

4.9.3.12 `virtual double BcpsObject::downEstimate () const [inline],[virtual]`

Return "down" estimate.

Default: 1.0e-5.

Definition at line 287 of file BcpsObject.h.

4.9.3.13 `AlpsReturnStatus BcpsObject::encodeBcpsObject (AlpsEncoded * encoded) const [inline],[protected]`

Pack Bcps part to a encode object.

Definition at line 294 of file BcpsObject.h.

4.9.3.14 `AlpsReturnStatus BcpsObject::decodeBcpsObject (AlpsEncoded & encoded) [inline],[protected]`

Unpack Bcps part from a encode object.

Definition at line 310 of file BcpsObject.h.

4.9.3.15 `virtual AlpsReturnStatus BcpsObject::encode (AlpsEncoded * encoded) [inline],[virtual]`

Pack into a encode object.

Reimplemented from **AlpsKnowledge**.

Definition at line 331 of file BcpsObject.h.

4.9.3.16 `virtual AlpsKnowledge* BcpsObject::decode (AlpsEncoded & encoded) const [inline],[virtual]`

Decode a constraint from an encoded object.

Reimplemented from **AlpsKnowledge**.

Definition at line 338 of file BcpsObject.h.

4.9.4 Member Data Documentation

4.9.4.1 `BcpsIntegral_t BcpsObject::intType_ [protected]`

The integrality type of the object, i.e., what values it can take up between the specified bounds.

(Possible options: 'C' for continuous, 'I' for general integer, 'B' for binary and 'S' for semicontinuous)

Definition at line 90 of file BcpsObject.h.

4.9.4.2 `BcpsValidRegion BcpsObject::validRegion_ [protected]`

Valid in the whole tree or only the subtree rooted at the node that generate this object.

Definition at line 94 of file BcpsObject.h.

4.9.4.3 double BcpsObject::hashValue_ [protected]

The hash value of this object.

Definition at line 114 of file BcpsObject.h.

4.9.4.4 int BcpsObject::numInactive_ [protected]

Number of inactive when in formulation.

Definition at line 117 of file BcpsObject.h.

4.9.4.5 double BcpsObject::effectiveness_ [protected]

Effectiveness: nonnegative value.

Definition at line 120 of file BcpsObject.h.

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

- BcpsObject.h

4.10 BcpsObjectListMod Struct Reference

Here is the set of `vectorMod_` objects that represent the list of objects of a particular type (either in relative or explicit form).

```
#include <BcpsNodeDesc.h>
```

Collaboration diagram for BcpsObjectListMod:

Public Attributes

- int `numRemove`
The number of entries to be deleted.
- int * `posRemove`
The positions of the entries to be deleted.
- int `numAdd`
The number of objects that are to be added.
- `BcpsObject` ** `objects`
The objects to be added.
- `BcpsFieldListMod`< double > `lbHard`
These are the data structures that store the changes in the individual fields.

4.10.1 Detailed Description

Here is the set of `vectorMod_` objects that represent the list of objects of a particular type (either in relative or explicit form).

If `numRemove` is positive, then `posRemove` contains the positions of the objects that are to be deleted from all the lists (that are stored relatively).

Definition at line 66 of file `BcpsNodeDesc.h`.

4.10.2 Member Data Documentation

4.10.2.1 `int BcpsObjectListMod::numRemove`

The number of entries to be deleted.

If this is -1, then all objects should be deleted, i.e., we have an explicit list.

Definition at line 70 of file `BcpsNodeDesc.h`.

4.10.2.2 `int* BcpsObjectListMod::posRemove`

The positions of the entries to be deleted.

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

4.10.2.3 `int BcpsObjectListMod::numAdd`

The number of objects that are to added.

Definition at line 76 of file `BcpsNodeDesc.h`.

4.10.2.4 `BcpsObject** BcpsObjectListMod::objects`

The objects to be added.

Definition at line 78 of file `BcpsNodeDesc.h`.

4.10.2.5 `BcpsFieldListMod<double> BcpsObjectListMod::lbHard`

These are the data structures that store the changes in the individual fields.

Definition at line 83 of file `BcpsNodeDesc.h`.

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

- `BcpsNodeDesc.h`

4.11 BcpsObjectPool Class Reference

Object pool is used to store objects.

```
#include <BcpsObjectPool.h>
```

Inheritance diagram for `BcpsObjectPool`:

Collaboration diagram for `BcpsObjectPool`:

Public Member Functions

- [BcpsObjectPool](#) ()
Default construct.
- void [freeGuts](#) ()
Free object pointers.
- void [clear](#) ()
Reset to empty.
- virtual void [addKnowledge](#) (**AlpsKnowledge** *nk, double priority)
Add a knowledge to pool.
- virtual int [getNumKnowledges](#) () const
Query how many knowledges are in the pool.
- virtual std::pair< **AlpsKnowledge** *, double > [getKnowledge](#) () const
Query a knowledge, but doesn't remove it from the pool.
- virtual bool [hasKnowledge](#) () const
Check whether the pool has knowledge.
- void [deleteObject](#) (int k)
Delete object k from pool.
- const std::vector< **AlpsKnowledge** * > & [getObjects](#) () const
Get all objects.
- **AlpsKnowledge** * [getObject](#) (int k) const
Get a object.

4.11.1 Detailed Description

Object pool is used to store objects.

Definition at line 36 of file BcpsObjectPool.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 `BcpsObjectPool::BcpsObjectPool ()` `[inline]`

Default construct.

Definition at line 45 of file BcpsObjectPool.h.

4.11.3 Member Function Documentation

4.11.3.1 `void BcpsObjectPool::freeGuts ()` `[inline]`

Free object pointers.

Definition at line 53 of file BcpsObjectPool.h.

4.11.3.2 `void BcpsObjectPool::clear ()` `[inline]`

Reset to empty.

Don't free memory.

Definition at line 61 of file BcpsObjectPool.h.

4.11.3.3 `virtual int BcpsObjectPool::getNumKnowledges () const [inline],[virtual]`

Query how many knowledges are in the pool.

Implements **AlpsKnowledgePool**.

Definition at line 69 of file BcpsObjectPool.h.

4.11.3.4 `virtual bool BcpsObjectPool::hasKnowledge () const [inline],[virtual]`

Check whether the pool has knowledge.

Reimplemented from **AlpsKnowledgePool**.

Definition at line 79 of file BcpsObjectPool.h.

4.11.3.5 `const std::vector<AlpsKnowledge*>& BcpsObjectPool::getObjects () const [inline]`

Get all objects.

Definition at line 96 of file BcpsObjectPool.h.

4.11.3.6 `AlpsKnowledge* BcpsObjectPool::getObject (int k) const [inline]`

Get a object.

Definition at line 99 of file BcpsObjectPool.h.

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

- BcpsObjectPool.h

4.12 BcpsSolution Class Reference

This class holds the solution objects.

```
#include <BcpsSolution.h>
```

Inheritance diagram for BcpsSolution:

Collaboration diagram for BcpsSolution:

Public Member Functions

- [BcpsSolution](#) ()
Default constructor.
- [BcpsSolution](#) (int size, const double *values, double q)
Useful constructor.
- [BcpsSolution](#) (int size, [BcpsObject_p](#) *&objects, double *&values, double q)
Construct an object using the given arrays.
- virtual [~BcpsSolution](#) ()
Distructor.
- virtual void [print](#) (std::ostream &os) const

Print out the solution.

- AlpsReturnStatus [encodeBcps](#) (**AlpsEncoded** *encoded) const

Pack Bcps part of solution into an encoded objects.

- AlpsReturnStatus [decodeBcps](#) (**AlpsEncoded** &encoded)

Unpack Bcps part of solution from an encoded objects.

- int [getSize](#) () const

Get the appropriate data member.

- void [setSize](#) (int s)

Set/assign the appropriate data member.

- virtual **BcpsSolution** * [selectNonzeros](#) (const double etol=1e-5) const

Select the fractional/nonzero elements from the solution array and return a new object in compacted form.

Protected Attributes

- int [size_](#)

Size of values_.

- **BcpsObject_p** * [objects_](#)

List of objects associated with values.

- double * [values_](#)

Solution values.

- double [quality_](#)

Quality/Objective value associated with this solution.

4.12.1 Detailed Description

This class holds the solution objects.

At this level, a solution is just considered to be a list of objects with associated values.

Definition at line 34 of file BcpsSolution.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 **BcpsSolution::BcpsSolution** () `[inline]`

Default constructor.

Definition at line 58 of file BcpsSolution.h.

4.12.2.2 **BcpsSolution::BcpsSolution** (int *size*, const double * *values*, double *q*) `[inline]`

Useful constructor.

Definition at line 64 of file BcpsSolution.h.

4.12.2.3 `BcpsSolution::BcpsSolution (int size, BcpsObject_p *& objects, double *& values, double q)` [inline]

Construct an object using the given arrays.

Note that the new objects takes over the pointers and NULLs them out in the calling method.

Definition at line 80 of file BcpsSolution.h.

4.12.2.4 `virtual BcpsSolution::~~BcpsSolution ()` [inline],[virtual]

Distructor.

Definition at line 88 of file BcpsSolution.h.

4.12.3 Member Function Documentation

4.12.3.1 `int BcpsSolution::getSize () const` [inline]

Get the appropriate data member.

Definition at line 100 of file BcpsSolution.h.

4.12.3.2 `void BcpsSolution::setSize (int s)` [inline]

Set/assign the appropriate data member.

Definition at line 108 of file BcpsSolution.h.

4.12.3.3 `virtual BcpsSolution* BcpsSolution::selectNonzeros (const double etol = 1e-5) const` [virtual]

Select the fractional/nonzero elements from the solution array and return a new object in compacted form.

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

Print out the solution.

Reimplemented from **AlpsSolution**.

Definition at line 130 of file BcpsSolution.h.

4.12.3.5 `AlpsReturnStatus BcpsSolution::encodeBcps (AlpsEncoded * encoded) const`

Pack Bcps part of solution into an encoded objects.

4.12.3.6 `AlpsReturnStatus BcpsSolution::decodeBcps (AlpsEncoded & encoded)`

Unpack Bcps part of solution from an encoded objects.

4.12.4 Member Data Documentation

4.12.4.1 `int BcpsSolution::size_` [protected]

Size of values_.

Definition at line 44 of file BcpsSolution.h.

4.12.4.2 `BcpsObject_p* BcpsSolution::objects_` [protected]

List of objects associated with values.

Can be NULL.

Definition at line 47 of file BcpsSolution.h.

4.12.4.3 `double* BcpsSolution::values_` [protected]

Solution values.

Definition at line 50 of file BcpsSolution.h.

4.12.4.4 `double BcpsSolution::quality_` [protected]

Quality/Objective value associated with this solution.

Definition at line 53 of file BcpsSolution.h.

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

- BcpsSolution.h

4.13 BcpsSubTree Class Reference

This class is the data structure for storing a subtree within BCPS.

```
#include <BcpsSubTree.h>
```

Inheritance diagram for BcpsSubTree:

Collaboration diagram for BcpsSubTree:

4.13.1 Detailed Description

This class is the data structure for storing a subtree within BCPS.

The biggest addition to the fields that already exist withink ALPS is the storage for the global list of objects that are active within that subtree. Initially, this will be implemeted as a `std::set`, but later on should be changed to something more efficient such as a hash table or something like that.

Definition at line 42 of file BcpsSubTree.h.

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

- BcpsSubTree.h

4.14 BcpsTreeNode Class Reference

This class contain the data for a BCPS search tree node.

```
#include <BcpsTreeNode.h>
```

Inheritance diagram for BcpsTreeNode:

Collaboration diagram for BcpsTreeNode:

Public Member Functions

- [BcpsTreeNode](#) ()
Default constructor.
- virtual [~BcpsTreeNode](#) ()
Destructor.
- virtual int [process](#) (bool isRoot=false, bool rampUp=false)
This methods performs the processing of the node.
- virtual int [bound](#) ([BcpsModel](#) *model)=0
Bounding procedure to estimate quality of this node.
- virtual std::vector< [CoinTriple](#)< [AlpsNodeDesc](#) *, [AlpsNodeStatus](#), double > > [branch](#) ()=0
*This method must be invoked on a *pregnant* node (which has all the information needed to create the children) and should create the children's descriptions.*
- const [BcpsBranchObject](#) * [branchObject](#) () const
Return the branching object.
- void [setBranchObject](#) ([BcpsBranchObject](#) *b)
Set the branching object.

Protected Member Functions

- virtual int [generateConstraints](#) ([BcpsModel](#) *model, [BcpsConstraintPool](#) *conPool)
Generate constraints.
- virtual int [generateVariables](#) ([BcpsModel](#) *model, [BcpsVariablePool](#) *varPool)
Generate variables.
- virtual int [chooseBranchingObject](#) ([BcpsModel](#) *model)=0
Choose a branching object.
- virtual int [installSubProblem](#) ([BcpsModel](#) *model)=0
Extract node information (bounds, constraints, variables) from this node and load the information into the relaxation solver, such as linear programming solver.
- virtual int [handleBoundingStatus](#) (int status, bool &keepOn, bool &fathomed)
Handle bounding status:
- [AlpsReturnStatus](#) [encodeBcps](#) ([AlpsEncoded](#) *encoded) const
Pack Bcps portion of node into an encoded object.

Protected Attributes

- [BcpsBranchObject](#) * [branchObject_](#)
Branching object for this node, which has information of how to execute branching.

4.14.1 Detailed Description

This class contain the data for a BCPS search tree node.

At this level, we consider a tree node to be simply a list of objects. The objects are organized by type. A differencing scheme is implemented here by looking for differences between the lists of each type in the current node and its parent.

Definition at line 46 of file BcpsTreeNode.h.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 BcpsTreeNode::BcpsTreeNode () [inline]

Default constructor.

Definition at line 99 of file BcpsTreeNode.h.

4.14.2.2 virtual BcpsTreeNode::~BcpsTreeNode () [inline],[virtual]

Destructor.

Definition at line 102 of file BcpsTreeNode.h.

4.14.3 Member Function Documentation

4.14.3.1 virtual int BcpsTreeNode::generateConstraints (BcpsModel * *model*, BcpsConstraintPool * *conPool*) [inline],[protected],[virtual]

Generate constraints.

The generated constraints are stored in constraint pool. The default implementation does nothing.

Definition at line 58 of file BcpsTreeNode.h.

4.14.3.2 virtual int BcpsTreeNode::generateVariables (BcpsModel * *model*, BcpsVariablePool * *varPool*) [inline],[protected],[virtual]

Generate variables.

The generated variables are stored in variable pool. The default implementation does nothing.

Definition at line 66 of file BcpsTreeNode.h.

4.14.3.3 virtual int BcpsTreeNode::chooseBranchingObject (BcpsModel * *model*) [protected],[pure virtual]

Choose a branching object.

4.14.3.4 virtual int BcpsTreeNode::installSubProblem (BcpsModel * *model*) [protected],[pure virtual]

Extract node information (bounds, constraints, variables) from this node and load the information into the relaxation solver, such as linear programming solver.

4.14.3.5 `virtual int BcpsTreeNode::handleBoundingStatus (int status, bool & keepOn, bool & fathomed) [inline],
[protected],[virtual]`

Handle bounding status:

- relaxed feasible but not integer feasible,
- integer feasible,
- infeasible,
- unbounded,
- fathomed (for instance, reaching objective limit for LP). Set node status accordingly.

Parameters

<i>staus</i>	Input The solution status of bounding.
<i>keepOn</i>	Output Whether to keep on bounding.
<i>fathomed</i>	Output Whether this node is fathomed.

Definition at line 91 of file BcpsTreeNode.h.

4.14.3.6 `virtual int BcpsTreeNode::process (bool isRoot = false, bool rampUp = false) [virtual]`

This methods performs the processing of the node.

For branch and bound, this would mean performing the bounding operation. The minimum requirement for this method is that it change the status to either internal or fathomed so the tree manager can deal with it afterwards. The status of the node when it begins processing will be active.

4.14.3.7 `virtual int BcpsTreeNode::bound (BcpsModel * model) [pure virtual]`

Bounding procedure to estimate quality of this node.

4.14.3.8 `virtual std::vector< CoinTriple<AlpsNodeDesc*, AlpsNodeStatus, double> > BcpsTreeNode::branch () [pure
virtual]`

This method must be invoked on a `pregnant` node (which has all the information needed to create the children) and should create the children's decriptions.

The stati of the children can be any of the ones `process()` can return.

4.14.3.9 `const BcpsBranchObject* BcpsTreeNode::branchObject () const [inline]`

Return the branching object.

Definition at line 123 of file BcpsTreeNode.h.

4.14.3.10 `void BcpsTreeNode::setBranchObject (BcpsBranchObject * b) [inline]`

Set the branching object.

Definition at line 126 of file BcpsTreeNode.h.

4.14.3.11 `AlpsReturnStatus BcpsTreeNode::encodeBcps (AlpsEncoded * encoded) const` `[inline]`, `[protected]`

Pack Bcps portion of node into an encoded object.

Definition at line 131 of file BcpsTreeNode.h.

4.14.4 Member Data Documentation

4.14.4.1 `BcpsBranchObject* BcpsTreeNode::branchObject_` `[protected]`

Branching object for this node, which has information of how to execute branching.

Definition at line 52 of file BcpsTreeNode.h.

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

- BcpsTreeNode.h

4.15 BcpsVariable Class Reference

Inheritance diagram for BcpsVariable:

Collaboration diagram for BcpsVariable:

Public Member Functions

- [BcpsVariable](#) ()
Default constructor.
- [BcpsVariable](#) (double lbh, double ubh, double lbs, double ubs)
Useful constructor.
- virtual [~BcpsVariable](#) ()
Destructor.
- [BcpsVariable](#) (const [BcpsVariable](#) &rhs)
Copy constructor.

Additional Inherited Members

4.15.1 Detailed Description

Definition at line 383 of file BcpsObject.h.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 `BcpsVariable::BcpsVariable ()` `[inline]`

Default constructor.

Definition at line 386 of file BcpsObject.h.

4.15.2.2 `BcpsVariable::BcpsVariable (double lbh, double ubh, double lbs, double ubs) [inline]`

Useful constructor.

Definition at line 389 of file `BcpsObject.h`.

4.15.2.3 `virtual BcpsVariable::~~BcpsVariable () [inline],[virtual]`

Destructor.

Definition at line 395 of file `BcpsObject.h`.

4.15.2.4 `BcpsVariable::BcpsVariable (const BcpsVariable & rhs) [inline]`

Copy constructor.

Definition at line 398 of file `BcpsObject.h`.

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

- `BcpsObject.h`

4.16 BcpsVariablePool Class Reference

Inheritance diagram for `BcpsVariablePool`:

Collaboration diagram for `BcpsVariablePool`:

Public Member Functions

- void `addVariable (BcpsVariable *var)`
Add a variable to pool.
- void `deleteVariable (int k)`
Delete variable k from pool.
- int `getNumVariables () const`
Query how many variables are in the pool.
- const std::vector< **AlpsKnowledge** * > & `getVariables () const`
Get the vector of variables.
- **AlpsKnowledge** * `getVariable (int k) const`
Get the vector of variables.

4.16.1 Detailed Description

Definition at line 127 of file `BcpsObjectPool.h`.

4.16.2 Member Function Documentation

4.16.2.1 `int BcpsVariablePool::getNumVariables () const [inline]`

Query how many variables are in the pool.

Definition at line 139 of file BcpsObjectPool.h.

4.16.2.2 `const std::vector<AlpsKnowledge*>& BcpsVariablePool::getVariables () const` `[inline]`

Get the vector of variables.

Definition at line 142 of file BcpsObjectPool.h.

4.16.2.3 `AlpsKnowledge* BcpsVariablePool::getVariable (int k) const` `[inline]`

Get the vector of variables.

Definition at line 145 of file BcpsObjectPool.h.

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

- BcpsObjectPool.h

File Documentation

Index

- ~BcpsBranchStrategy
 - BcpsBranchStrategy, [15](#)
- ~BcpsConstraint
 - BcpsConstraint, [18](#)
- ~BcpsNodeDesc
 - BcpsNodeDesc, [26](#)
- ~BcpsSolution
 - BcpsSolution, [40](#)
- ~BcpsTreeNode
 - BcpsTreeNode, [43](#)
- ~BcpsVariable
 - BcpsVariable, [46](#)
- addVariables
 - BcpsNodeDesc, [28](#)
- appendAddedConstraints
 - BcpsNodeDesc, [28](#)
- assignCons
 - BcpsNodeDesc, [26](#)
- assignVarHardBound
 - BcpsNodeDesc, [27](#)
- assignVarSoftBound
 - BcpsNodeDesc, [27](#)
- assignVars
 - BcpsNodeDesc, [26](#)
- BcpsBranchObject, [13](#)
- BcpsBranchStrategy, [13](#)
 - ~BcpsBranchStrategy, [15](#)
 - BcpsBranchStrategy, [14](#)
 - bestBranchObject, [16](#)
 - bestBranchObject_, [16](#)
 - bestChangeDown_, [17](#)
 - bestChangeUp_, [16](#)
 - bestNumberDown_, [17](#)
 - bestNumberUp_, [17](#)
 - betterBranchObject, [16](#)
 - branchObjects_, [16](#)
 - clearBest, [15](#)
 - clone, [15](#)
 - createCandBranchObjects, [15](#)
 - getNumBranchObjects, [15](#)
 - getType, [15](#)
 - model_, [16](#)
 - numBranchObjects_, [16](#)
 - setModel, [15](#)
 - setType, [15](#)
 - type_, [16](#)
- BcpsConstraint, [17](#)
 - ~BcpsConstraint, [18](#)
 - BcpsConstraint, [18](#)
- BcpsConstraintPool, [18](#)
 - getConstraint, [19](#)
 - getConstraints, [19](#)
 - getNumConstraints, [19](#)
- BcpsFieldListMod
 - numModify, [20](#)
 - posModify, [20](#)
- BcpsFieldListMod< T >, [19](#)
- BcpsMessage, [20](#)
- bcpsMessageHandler
 - BcpsModel, [22](#)
- bcpsMessageHandler_
 - BcpsModel, [23](#)
- bcpsMessages
 - BcpsModel, [22](#)
- bcpsMessages_
 - BcpsModel, [23](#)
- BcpsModel, [20](#)
 - bcpsMessageHandler, [22](#)
 - bcpsMessageHandler_, [23](#)
 - bcpsMessages, [22](#)
 - bcpsMessages_, [23](#)
 - constraints_, [22](#)
 - decodeBcps, [22](#)
 - encodeBcps, [22](#)
 - getConstrints, [21](#)
 - getVariables, [21](#)
 - numCoreConstraints_, [22](#)
 - numCoreVariables_, [22](#)
 - variables_, [22](#)
- BcpsNodeDesc, [23](#)
 - ~BcpsNodeDesc, [26](#)
 - addVariables, [28](#)
 - appendAddedConstraints, [28](#)
 - assignCons, [26](#)
 - assignVarHardBound, [27](#)
 - assignVarSoftBound, [27](#)
 - assignVars, [26](#)
 - BcpsNodeDesc, [26](#)
 - cons, [27](#)

- cons_, 29
- decodeBcps, 29
- decodeDbfFieldMods, 29
- decodeIntFieldMods, 29
- decodeObjectMods, 29
- delConstraints, 28
- delVariables, 28
- encodeBcps, 29
- encodeDbfFieldMods, 28
- encodeIntFieldMods, 29
- encodeObjectMods, 29
- getCons, 27
- getVars, 26
- initToNull, 26
- setAddedConstraints, 28
- setConHardBound, 28
- setConSoftBound, 27
- setCons, 26
- setVarHardBound, 27
- setVarSoftBound, 27
- setVars, 26
- vars, 27
- vars_, 29
- BcpsObject, 30
 - BcpsObject, 32
 - boundBranch, 33
 - clone, 32
 - createBranchObject, 32
 - decode, 34
 - decodeBcpsObject, 34
 - downEstimate, 33
 - effectiveness_, 35
 - encode, 34
 - encodeBcpsObject, 34
 - feasibleRegion, 32
 - floorCeiling, 33
 - hashValue_, 34
 - infeasibility, 32
 - intType_, 34
 - notPreferredNewFeasible, 33
 - numInactive_, 35
 - operator=, 32
 - preferredNewFeasible, 33
 - resetBounds, 33
 - upEstimate, 33
 - validRegion_, 34
- BcpsObjectListMod, 35
 - lbHard, 36
 - numAdd, 36
 - numRemove, 36
 - objects, 36
 - posRemove, 36
- BcpsObjectPool, 36
 - BcpsObjectPool, 37
- clear, 37
- freeGuts, 37
- getNumKnowledges, 37
- getObject, 38
- getObjects, 38
- hasKnowledge, 38
- BcpsSolution, 38
 - ~BcpsSolution, 40
 - BcpsSolution, 39
 - decodeBcps, 40
 - encodeBcps, 40
 - getSize, 40
 - objects_, 41
 - print, 40
 - quality_, 41
 - selectNonzeros, 40
 - setSize, 40
 - size_, 40
 - values_, 41
- BcpsSubTree, 41
- BcpsTreeNode, 42
 - ~BcpsTreeNode, 43
 - BcpsTreeNode, 43
 - bound, 44
 - branch, 44
 - branchObject, 44
 - branchObject_, 45
 - chooseBranchingObject, 43
 - encodeBcps, 44
 - generateConstraints, 43
 - generateVariables, 43
 - handleBoundingStatus, 43
 - installSubProblem, 43
 - process, 44
 - setBranchObject, 44
- BcpsVariable, 45
 - ~BcpsVariable, 46
 - BcpsVariable, 45, 46
- BcpsVariablePool, 46
 - getNumVariables, 46
 - getVariable, 47
 - getVariables, 47
- bestBranchObject
 - BcpsBranchStrategy, 16
- bestBranchObject_
 - BcpsBranchStrategy, 16
- bestChangeDown_
 - BcpsBranchStrategy, 17
- bestChangeUp_
 - BcpsBranchStrategy, 16
- bestNumberDown_
 - BcpsBranchStrategy, 17
- bestNumberUp_
 - BcpsBranchStrategy, 17

- betterBranchObject
 - BcpsBranchStrategy, 16
- bound
 - BcpsTreeNode, 44
- boundBranch
 - BcpsObject, 33
- branch
 - BcpsTreeNode, 44
- branchObject
 - BcpsTreeNode, 44
- branchObject_
 - BcpsTreeNode, 45
- branchObjects_
 - BcpsBranchStrategy, 16
- chooseBranchingObject
 - BcpsTreeNode, 43
- clear
 - BcpsObjectPool, 37
- clearBest
 - BcpsBranchStrategy, 15
- clone
 - BcpsBranchStrategy, 15
 - BcpsObject, 32
- cons
 - BcpsNodeDesc, 27
- cons_
 - BcpsNodeDesc, 29
- constraints_
 - BcpsModel, 22
- createBranchObject
 - BcpsObject, 32
- createCandBranchObjects
 - BcpsBranchStrategy, 15
- decode
 - BcpsObject, 34
- decodeBcps
 - BcpsModel, 22
 - BcpsNodeDesc, 29
 - BcpsSolution, 40
- decodeBcpsObject
 - BcpsObject, 34
- decodeDbfFieldMods
 - BcpsNodeDesc, 29
- decodeIntFieldMods
 - BcpsNodeDesc, 29
- decodeObjectMods
 - BcpsNodeDesc, 29
- delConstraints
 - BcpsNodeDesc, 28
- delVariables
 - BcpsNodeDesc, 28
- downEstimate
 - BcpsObject, 33
- effectiveness_
 - BcpsObject, 35
- encode
 - BcpsObject, 34
- encodeBcps
 - BcpsModel, 22
 - BcpsNodeDesc, 29
 - BcpsSolution, 40
 - BcpsTreeNode, 44
- encodeBcpsObject
 - BcpsObject, 34
- encodeDbfFieldMods
 - BcpsNodeDesc, 28
- encodeIntFieldMods
 - BcpsNodeDesc, 29
- encodeObjectMods
 - BcpsNodeDesc, 29
- feasibleRegion
 - BcpsObject, 32
- floorCeiling
 - BcpsObject, 33
- freeGuts
 - BcpsObjectPool, 37
- generateConstraints
 - BcpsTreeNode, 43
- generateVariables
 - BcpsTreeNode, 43
- getCons
 - BcpsNodeDesc, 27
- getConstraint
 - BcpsConstraintPool, 19
- getConstraints
 - BcpsConstraintPool, 19
- getConstraints
 - BcpsModel, 21
- getNumBranchObjects
 - BcpsBranchStrategy, 15
- getNumConstraints
 - BcpsConstraintPool, 19
- getNumKnowledges
 - BcpsObjectPool, 37
- getNumVariables
 - BcpsVariablePool, 46
- getObject
 - BcpsObjectPool, 38
- getObjects
 - BcpsObjectPool, 38
- getSize
 - BcpsSolution, 40
- getType
 - BcpsBranchStrategy, 15
- getVariable
 - BcpsVariablePool, 47

- getVariables
 - BcpsModel, 21
 - BcpsVariablePool, 47
- getVars
 - BcpsNodeDesc, 26
- handleBoundingStatus
 - BcpsTreeNode, 43
- hasKnowledge
 - BcpsObjectPool, 38
- hashValue_
 - BcpsObject, 34
- infeasibility
 - BcpsObject, 32
- initToNull
 - BcpsNodeDesc, 26
- installSubProblem
 - BcpsTreeNode, 43
- intType_
 - BcpsObject, 34
- lbHard
 - BcpsObjectListMod, 36
- model_
 - BcpsBranchStrategy, 16
- notPreferredNewFeasible
 - BcpsObject, 33
- numAdd
 - BcpsObjectListMod, 36
- numBranchObjects_
 - BcpsBranchStrategy, 16
- numCoreConstraints_
 - BcpsModel, 22
- numCoreVariables_
 - BcpsModel, 22
- numInactive_
 - BcpsObject, 35
- numModify
 - BcpsFieldListMod, 20
- numRemove
 - BcpsObjectListMod, 36
- objects
 - BcpsObjectListMod, 36
- objects_
 - BcpsSolution, 41
- operator=
 - BcpsObject, 32
- posModify
 - BcpsFieldListMod, 20
- posRemove
 - BcpsObjectListMod, 36
- preferredNewFeasible
 - BcpsObject, 33
- print
 - BcpsSolution, 40
- process
 - BcpsTreeNode, 44
- quality_
 - BcpsSolution, 41
- resetBounds
 - BcpsObject, 33
- selectNonzeros
 - BcpsSolution, 40
- setAddedConstraints
 - BcpsNodeDesc, 28
- setBranchObject
 - BcpsTreeNode, 44
- setConHardBound
 - BcpsNodeDesc, 28
- setConSoftBound
 - BcpsNodeDesc, 27
- setCons
 - BcpsNodeDesc, 26
- setModel
 - BcpsBranchStrategy, 15
- setSize
 - BcpsSolution, 40
- setType
 - BcpsBranchStrategy, 15
- setVarHardBound
 - BcpsNodeDesc, 27
- setVarSoftBound
 - BcpsNodeDesc, 27
- setVars
 - BcpsNodeDesc, 26
- size_
 - BcpsSolution, 40
- type_
 - BcpsBranchStrategy, 16
- upEstimate
 - BcpsObject, 33
- validRegion_
 - BcpsObject, 34
- values_
 - BcpsSolution, 41
- variables_
 - BcpsModel, 22
- vars
 - BcpsNodeDesc, 27
- vars_
 - BcpsNodeDesc, 29