

Osi
trunk

Generated by Doxygen 1.8.5

Mon Oct 21 2013 18:57:48

Contents

1	Todo List	1
2	Deprecated List	2
3	Namespace Index	2
3.1	Namespace List	2
4	Hierarchical Index	2
4.1	Class Hierarchy	2
5	Class Index	6
5.1	Class List	7
6	File Index	9
6.1	File List	9
7	Namespace Documentation	10
7.1	OsiUnitTest Namespace Reference	10
7.1.1	Detailed Description	11
7.1.2	Function Documentation	11
7.1.3	Variable Documentation	12
7.2	soplex Namespace Reference	12
8	Class Documentation	12
8.1	OsiSolverInterface::ApplyCutsReturnCode Class Reference	12
8.1.1	Detailed Description	14
8.1.2	Constructor & Destructor Documentation	14
8.1.3	Member Function Documentation	14
8.1.4	Friends And Related Function Documentation	15
8.1.5	Member Data Documentation	16
8.2	OsiCuts::const_iterator Class Reference	16
8.2.1	Detailed Description	17
8.2.2	Member Typedef Documentation	17
8.2.3	Constructor & Destructor Documentation	18
8.2.4	Member Function Documentation	18
8.2.5	Friends And Related Function Documentation	18
8.2.6	Member Data Documentation	18
8.3	glp_prob Struct Reference	19

8.3.1	Detailed Description	19
8.3.2	Member Data Documentation	19
8.4	OsiCuts::iterator Class Reference	19
8.4.1	Detailed Description	20
8.4.2	Constructor & Destructor Documentation	20
8.4.3	Member Function Documentation	20
8.4.4	Friends And Related Function Documentation	21
8.4.5	Member Data Documentation	21
8.5	OsiAuxInfo Class Reference	21
8.5.1	Detailed Description	22
8.5.2	Constructor & Destructor Documentation	22
8.5.3	Member Function Documentation	22
8.5.4	Member Data Documentation	23
8.6	OsiBabSolver Class Reference	23
8.6.1	Detailed Description	25
8.6.2	Constructor & Destructor Documentation	25
8.6.3	Member Function Documentation	25
8.6.4	Member Data Documentation	28
8.7	OsiBranchingInformation Class Reference	29
8.7.1	Detailed Description	31
8.7.2	Constructor & Destructor Documentation	31
8.7.3	Member Function Documentation	31
8.7.4	Member Data Documentation	32
8.8	OsiBranchingObject Class Reference	35
8.8.1	Detailed Description	36
8.8.2	Constructor & Destructor Documentation	36
8.8.3	Member Function Documentation	37
8.8.4	Member Data Documentation	39
8.9	OsiChooseStrong Class Reference	39
8.9.1	Detailed Description	41
8.9.2	Constructor & Destructor Documentation	41
8.9.3	Member Function Documentation	41
8.9.4	Member Data Documentation	42
8.10	OsiChooseVariable Class Reference	43
8.10.1	Detailed Description	46
8.10.2	Constructor & Destructor Documentation	46
8.10.3	Member Function Documentation	47

8.10.4	Member Data Documentation	50
8.11	OsiColCut Class Reference	52
8.11.1	Detailed Description	54
8.11.2	Constructor & Destructor Documentation	54
8.11.3	Member Function Documentation	55
8.11.4	Friends And Related Function Documentation	56
8.11.5	Member Data Documentation	57
8.12	OsiCpxSolverInterface Class Reference	57
8.12.1	Detailed Description	65
8.12.2	Member Enumeration Documentation	65
8.12.3	Constructor & Destructor Documentation	66
8.12.4	Member Function Documentation	66
8.12.5	Friends And Related Function Documentation	80
8.12.6	Member Data Documentation	80
8.13	OsiCut Class Reference	82
8.13.1	Detailed Description	84
8.13.2	Constructor & Destructor Documentation	84
8.13.3	Member Function Documentation	84
8.13.4	Member Data Documentation	86
8.14	OsiCuts::OsiCutCompare Class Reference	87
8.14.1	Detailed Description	87
8.14.2	Member Function Documentation	87
8.15	OsiCuts Class Reference	87
8.15.1	Detailed Description	90
8.15.2	Constructor & Destructor Documentation	90
8.15.3	Member Function Documentation	90
8.15.4	Friends And Related Function Documentation	94
8.15.5	Member Data Documentation	94
8.16	OsiGlpkSolverInterface Class Reference	94
8.16.1	Detailed Description	102
8.16.2	Member Enumeration Documentation	103
8.16.3	Constructor & Destructor Documentation	103
8.16.4	Member Function Documentation	103
8.16.5	Friends And Related Function Documentation	116
8.16.6	Member Data Documentation	116
8.17	OsiGrbSolverInterface Class Reference	121
8.17.1	Detailed Description	130

8.17.2	Member Enumeration Documentation	130
8.17.3	Constructor & Destructor Documentation	131
8.17.4	Member Function Documentation	131
8.17.5	Friends And Related Function Documentation	146
8.17.6	Member Data Documentation	146
8.18	OsiHotInfo Class Reference	150
8.18.1	Detailed Description	151
8.18.2	Constructor & Destructor Documentation	152
8.18.3	Member Function Documentation	152
8.18.4	Member Data Documentation	154
8.19	OsiIntegerBranchingObject Class Reference	154
8.19.1	Detailed Description	155
8.19.2	Constructor & Destructor Documentation	156
8.19.3	Member Function Documentation	156
8.19.4	Member Data Documentation	156
8.20	OsiLotsize Class Reference	157
8.20.1	Detailed Description	158
8.20.2	Constructor & Destructor Documentation	159
8.20.3	Member Function Documentation	159
8.20.4	Member Data Documentation	161
8.21	OsiLotsizeBranchingObject Class Reference	162
8.21.1	Detailed Description	162
8.21.2	Constructor & Destructor Documentation	163
8.21.3	Member Function Documentation	163
8.21.4	Member Data Documentation	163
8.22	OsiMskSolverInterface Class Reference	164
8.22.1	Detailed Description	172
8.22.2	Member Enumeration Documentation	172
8.22.3	Constructor & Destructor Documentation	172
8.22.4	Member Function Documentation	172
8.22.5	Friends And Related Function Documentation	185
8.22.6	Member Data Documentation	185
8.23	OsiObject Class Reference	188
8.23.1	Detailed Description	190
8.23.2	Constructor & Destructor Documentation	190
8.23.3	Member Function Documentation	191
8.23.4	Member Data Documentation	194

8.24	OsiObject2 Class Reference	194
8.24.1	Detailed Description	195
8.24.2	Constructor & Destructor Documentation	195
8.24.3	Member Function Documentation	195
8.24.4	Member Data Documentation	196
8.25	OsiPresolve Class Reference	196
8.25.1	Detailed Description	198
8.25.2	Constructor & Destructor Documentation	198
8.25.3	Member Function Documentation	198
8.25.4	Member Data Documentation	200
8.26	OsiPseudoCosts Class Reference	202
8.26.1	Detailed Description	203
8.26.2	Constructor & Destructor Documentation	203
8.26.3	Member Function Documentation	203
8.26.4	Member Data Documentation	205
8.27	OsiRowCut Class Reference	205
8.27.1	Detailed Description	208
8.27.2	Constructor & Destructor Documentation	208
8.27.3	Member Function Documentation	208
8.27.4	Friends And Related Function Documentation	211
8.27.5	Member Data Documentation	211
8.28	OsiRowCut2 Class Reference	211
8.28.1	Detailed Description	212
8.28.2	Constructor & Destructor Documentation	212
8.28.3	Member Function Documentation	212
8.28.4	Member Data Documentation	213
8.29	OsiRowCutDebugger Class Reference	213
8.29.1	Detailed Description	214
8.29.2	Constructor & Destructor Documentation	215
8.29.3	Member Function Documentation	215
8.29.4	Friends And Related Function Documentation	217
8.29.5	Member Data Documentation	217
8.30	OsiSimpleInteger Class Reference	217
8.30.1	Detailed Description	219
8.30.2	Constructor & Destructor Documentation	219
8.30.3	Member Function Documentation	219
8.30.4	Member Data Documentation	221

8.31	OsiSolverBranch Class Reference	221
8.31.1	Detailed Description	222
8.31.2	Constructor & Destructor Documentation	222
8.31.3	Member Function Documentation	223
8.31.4	Member Data Documentation	223
8.32	OsiSolverInterface Class Reference	224
8.32.1	Detailed Description	236
8.32.2	Member Typedef Documentation	237
8.32.3	Constructor & Destructor Documentation	237
8.32.4	Member Function Documentation	237
8.32.5	Friends And Related Function Documentation	266
8.32.6	Member Data Documentation	266
8.33	OsiSolverResult Class Reference	268
8.33.1	Detailed Description	269
8.33.2	Constructor & Destructor Documentation	270
8.33.3	Member Function Documentation	270
8.33.4	Member Data Documentation	271
8.34	OsiSOS Class Reference	271
8.34.1	Detailed Description	273
8.34.2	Constructor & Destructor Documentation	273
8.34.3	Member Function Documentation	273
8.34.4	Member Data Documentation	275
8.35	OsiSOSBranchingObject Class Reference	276
8.35.1	Detailed Description	277
8.35.2	Constructor & Destructor Documentation	277
8.35.3	Member Function Documentation	277
8.36	OsiSpxSolverInterface Class Reference	277
8.36.1	Detailed Description	283
8.36.2	Member Enumeration Documentation	284
8.36.3	Constructor & Destructor Documentation	284
8.36.4	Member Function Documentation	284
8.36.5	Friends And Related Function Documentation	294
8.36.6	Member Data Documentation	294
8.37	OsiTwoWayBranchingObject Class Reference	296
8.37.1	Detailed Description	297
8.37.2	Constructor & Destructor Documentation	297
8.37.3	Member Function Documentation	297

8.37.4	Member Data Documentation	298
8.38	OsiXprSolverInterface Class Reference	298
8.38.1	Detailed Description	305
8.38.2	Constructor & Destructor Documentation	305
8.38.3	Member Function Documentation	306
8.38.4	Friends And Related Function Documentation	318
8.38.5	Member Data Documentation	318
8.39	OsiUnitTest::TestOutcome Class Reference	321
8.39.1	Detailed Description	322
8.39.2	Member Enumeration Documentation	322
8.39.3	Constructor & Destructor Documentation	323
8.39.4	Member Function Documentation	323
8.39.5	Member Data Documentation	323
8.40	OsiUnitTest::TestOutcomes Class Reference	324
8.40.1	Detailed Description	324
8.40.2	Member Function Documentation	324
9	File Documentation	325
9.1	/home/ted/COIN/trunk/Osi/src/Osi/config_default.h File Reference	325
9.1.1	Macro Definition Documentation	325
9.2	/home/ted/COIN/trunk/Osi/src/Osi/config_osi_default.h File Reference	325
9.2.1	Macro Definition Documentation	326
9.3	/home/ted/COIN/trunk/Osi/src/Osi/OsiAuxInfo.hpp File Reference	326
9.4	/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp File Reference	326
9.5	/home/ted/COIN/trunk/Osi/src/Osi/OsiChooseVariable.hpp File Reference	327
9.6	/home/ted/COIN/trunk/Osi/src/Osi/OsiColCut.hpp File Reference	327
9.7	/home/ted/COIN/trunk/Osi/src/Osi/OsiCollections.hpp File Reference	328
9.7.1	Typedef Documentation	328
9.8	/home/ted/COIN/trunk/Osi/src/Osi/OsiConfig.h File Reference	328
9.9	/home/ted/COIN/trunk/Osi/src/Osi/OsiCut.hpp File Reference	329
9.9.1	Macro Definition Documentation	329
9.10	/home/ted/COIN/trunk/Osi/src/Osi/OsiCuts.hpp File Reference	329
9.11	/home/ted/COIN/trunk/Osi/src/Osi/OsiPresolve.hpp File Reference	330
9.12	/home/ted/COIN/trunk/Osi/src/Osi/OsiRowCut.hpp File Reference	330
9.12.1	Macro Definition Documentation	330
9.13	/home/ted/COIN/trunk/Osi/src/Osi/OsiRowCutDebugger.hpp File Reference	330
9.13.1	Detailed Description	331

9.14	/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverBranch.hpp File Reference	331
9.15	/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverInterface.hpp File Reference	331
9.16	/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverParameters.hpp File Reference	331
9.16.1	Enumeration Type Documentation	332
9.17	/home/ted/COIN/trunk/Osi/src/OsiCommonTest/OsiUnitTests.hpp File Reference	333
9.17.1	Detailed Description	335
9.17.2	Macro Definition Documentation	336
9.17.3	Function Documentation	337
9.18	/home/ted/COIN/trunk/Osi/src/OsiCpx/OsiCpxSolverInterface.hpp File Reference	338
9.18.1	Typedef Documentation	338
9.18.2	Function Documentation	339
9.19	/home/ted/COIN/trunk/Osi/src/OsiGlpk/OsiGlpkSolverInterface.hpp File Reference	339
9.19.1	Macro Definition Documentation	339
9.19.2	Function Documentation	339
9.20	/home/ted/COIN/trunk/Osi/src/OsiGrb/OsiGrbSolverInterface.hpp File Reference	340
9.20.1	Typedef Documentation	340
9.20.2	Function Documentation	340
9.21	/home/ted/COIN/trunk/Osi/src/OsiMsk/OsiMskSolverInterface.hpp File Reference	340
9.21.1	Typedef Documentation	341
9.21.2	Function Documentation	341
9.22	/home/ted/COIN/trunk/Osi/src/OsiSpx/OsiSpxSolverInterface.hpp File Reference	341
9.22.1	Function Documentation	342
9.23	/home/ted/COIN/trunk/Osi/src/OsiXpr/OsiXprSolverInterface.hpp File Reference	342
9.23.1	Typedef Documentation	342
9.23.2	Function Documentation	342

Index

343

1 Todo List

Member `OsiSolverInterface::getIntegerTolerance () const`

This method should be replaced; it's architecturally wrong. This should be an honest `dblParam` with a keyword. Underlying solvers that do not support integer variables should return false for set and get on this parameter. Underlying solvers that support integrality should add this to the parameters they support, using whatever tolerance is appropriate. -lh, 091021-

2 Deprecated List

Member [OsiSolverInterface::columnType](#) (bool refresh=false) const

See #getColType

3 Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

OsiUnitTest	
A namespace so we can define a few 'global' variables to use during tests	10
soplex	12

4 Hierarchical Index

4.1 Class Hierarchy

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

std::allocator< T >	
OsiSolverInterface::ApplyCutsReturnCode	12
std::array< T >	
std::auto_ptr< T >	
std::basic_string< Char >	
std::string	
std::wstring	
std::basic_string< char >	
std::basic_string< wchar_t >	
std::bitset< Bits >	
std::complex	
std::unordered_multimap< K, T >::const_iterator	
std::list< T >::const_iterator	
std::forward_list< T >::const_iterator	
std::map< K, T >::const_iterator	
OsiCuts::const_iterator	16
std::unordered_map< K, T >::const_iterator	
std::basic_string< Char >::const_iterator	
std::multimap< K, T >::const_iterator	
std::set< K >::const_iterator	
std::string::const_iterator	
std::unordered_set< K >::const_iterator	
std::wstring::const_iterator	
std::multiset< K >::const_iterator	
std::unordered_multiset< K >::const_iterator	
std::vector< T >::const_iterator	
std::deque< T >::const_iterator	

```

std::list< T >::const_reverse_iterator
std::forward_list< T >::const_reverse_iterator
std::map< K, T >::const_reverse_iterator
std::unordered_map< K, T >::const_reverse_iterator
std::multimap< K, T >::const_reverse_iterator
std::basic_string< Char >::const_reverse_iterator
std::unordered_multimap< K, T >::const_reverse_iterator
std::set< K >::const_reverse_iterator
std::string::const_reverse_iterator
std::unordered_set< K >::const_reverse_iterator
std::multiset< K >::const_reverse_iterator
std::wstring::const_reverse_iterator
std::unordered_multiset< K >::const_reverse_iterator
std::vector< T >::const_reverse_iterator
std::deque< T >::const_reverse_iterator
std::deque< T >
std::error_category
std::error_code
std::error_condition
std::exception
    std::bad_alloc
    std::bad_cast
    std::bad_exception
    std::bad_typeid
    std::ios_base::failure
    std::logic_error
        std::domain_error
        std::invalid_argument
        std::length_error
        std::out_of_range
    std::runtime_error
        std::overflow_error
        std::range_error
        std::underflow_error
std::forward_list< T >

```

glp_prob

19

```

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::wofstream
std::basic_istream
    basic_fstream< char >
    basic_fstream< wchar_t >
    basic_stringstream< char >
    basic_stringstream< wchar_t >
    std::basic_fstream
        std::fstream
        std::wfstream
    std::basic_stringstream
        std::stringstream
        std::wstringstream
std::basic_istream
    std::istream
    std::wistream
std::basic_ostream
    basic_ostream< char >
    basic_ostream< wchar_t >
    basic_ofstream< char >
    basic_ofstream< wchar_t >
    basic_ostringstream< char >
    basic_ostringstream< wchar_t >
    std::basic_ostream
        std::ofstream
        std::wofstream
    std::basic_ostringstream
        std::ostringstream
        std::wostringstream
    std::ostream
    std::wostream
std::ios
std::wios
std::list< T >::iterator
std::forward_list< T >::iterator
std::set< K >::iterator
std::map< K, T >::iterator

```

OsiCuts::iterator

19

```

std::unordered_map< K, T >::iterator
std::basic_string< Char >::iterator
std::multimap< K, T >::iterator
std::unordered_multimap< K, T >::iterator
std::string::iterator
std::unordered_set< K >::iterator
std::wstring::iterator
std::multiset< K >::iterator
std::unordered_multiset< K >::iterator
std::vector< T >::iterator
std::deque< T >::iterator
std::list< T >

```

OsiUnitTest::TestOutcomes	324
std::list< TestOutcome >	
std::map< K, T >	
std::multimap< K, T >	
std::multiset< K >	
OsiAuxInfo	21
OsiBabSolver	23
OsiBranchingInformation	29
OsiBranchingObject	35
OsiTwoWayBranchingObject	296
OsiIntegerBranchingObject	154
OsiLotsizeBranchingObject	162
OsiSOSBranchingObject	276
OsiChooseVariable	43
OsiChooseStrong	39
OsiCut	82
OsiColCut	52
OsiRowCut	205
OsiRowCut2	211
OsiCuts::OsiCutCompare	87
OsiCuts	87
OsiHotInfo	150
OsiObject	188
OsiObject2	194
OsiLotsize	157
OsiSimpleInteger	217
OsiSOS	271
OsiPresolve	196
OsiPseudoCosts	202
OsiRowCutDebugger	213
OsiSolverBranch	221
OsiSolverInterface	224

OsiCpxSolverInterface	57
OsiGlpkSolverInterface	94
OsiGrbSolverInterface	121
OsiMskSolverInterface	164
OsiSpxSolverInterface	277
OsiXprSolverInterface	298
OsiSolverResult	268
std::priority_queue< T >	
std::queue< T >	
std::deque< T >::reverse_iterator	
std::wstring::reverse_iterator	
std::forward_list< T >::reverse_iterator	
std::unordered_multimap< K, T >::reverse_iterator	
std::string::reverse_iterator	
std::unordered_multiset< K >::reverse_iterator	
std::map< K, T >::reverse_iterator	
std::vector< T >::reverse_iterator	
std::multimap< K, T >::reverse_iterator	
std::multiset< K >::reverse_iterator	
std::unordered_map< K, T >::reverse_iterator	
std::list< T >::reverse_iterator	
std::unordered_set< K >::reverse_iterator	
std::basic_string< Char >::reverse_iterator	
std::set< K >::reverse_iterator	
std::set< K >	
std::smart_ptr< T >	
std::stack< T >	
std::system_error	
OsiUnitTest::TestOutcome	321
std::thread	
std::unique_ptr< T >	
std::unordered_map< K, T >	
std::unordered_multimap< K, T >	
std::unordered_multiset< K >	
std::unordered_set< K >	
std::valarray< T >	
std::vector< T >	
std::vector< double >	
std::vector< OsiColCut * >	
std::vector< OsiRowCut * >	
std::vector< std::string >	
std::weak_ptr< T >	
K	
T	

5 Class Index

5.1 Class List

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

OsiSolverInterface::ApplyCutsReturnCode	
Internal class for obtaining status from the applyCuts method	12
OsiCuts::const_iterator	
Const Iterator	16
glp_prob	19
OsiCuts::iterator	
Iterator	19
OsiAuxInfo	
This class allows for a more structured use of algorithmic tweaking to an OsiSolverInterface	21
OsiBabSolver	
This class allows for the use of more exotic solvers e.g	23
OsiBranchingInformation	29
OsiBranchingObject	
Abstract branching object base class	35
OsiChooseStrong	
This class chooses a variable to branch on	39
OsiChooseVariable	
This class chooses a variable to branch on	43
OsiColCut	
Column Cut Class	52
OsiCpxSolverInterface	
CPLEX Solver Interface	57
OsiCut	82
OsiCuts::OsiCutCompare	87
OsiCuts	
Collections of row cuts and column cuts	87
OsiGlpkSolverInterface	94
OsiGrbSolverInterface	
Gurobi Solver Interface	121
OsiHotInfo	
This class contains the result of strong branching on a variable When created it stores enough information for strong branching	150
OsiIntegerBranchingObject	
Simple branching object for an integer variable	154

OsiLotsize	
Lotsize class	157
OsiLotsizeBranchingObject	
Lotsize branching object	162
OsiMskSolverInterface	164
OsiObject	
Abstract base class for 'objects'	188
OsiObject2	
Define a class to add a bit of complexity to OsiObject This assumes 2 way branching	194
OsiPresolve	
OSI interface to COIN problem simplification capabilities	196
OsiPseudoCosts	
This class is the placeholder for the pseudocosts used by OsiChooseStrong	202
OsiRowCut	
Row Cut Class	205
OsiRowCut2	
Row Cut Class which refers back to row which created it	211
OsiRowCutDebugger	
Validate cuts against a known solution	213
OsiSimpleInteger	
Define a single integer class	217
OsiSolverBranch	
Solver Branch Class	221
OsiSolverInterface	
Abstract Base Class for describing an interface to a solver	224
OsiSolverResult	
Solver Result Class	268
OsiSOS	
Define Special Ordered Sets of type 1 and 2	271
OsiSOSBranchingObject	
Branching object for Special ordered sets	276
OsiSpxSolverInterface	
SoPlex Solver Interface Instantiation of OsiSpxSolverInterface for SoPlex	277
OsiTwoWayBranchingObject	
This just adds two-wayness to a branching object	296
OsiXprSolverInterface	
XPRESS-MP Solver Interface	298

OsiUnitTest::TestOutcome	
A single test outcome record	321
OsiUnitTest::TestOutcomes	
Utility class to maintain a list of test outcomes	324

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

<code>/home/ted/COIN/trunk/Osi/src/Osi/config_default.h</code>	325
<code>/home/ted/COIN/trunk/Osi/src/Osi/config_osi_default.h</code>	325
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiAuxInfo.hpp</code>	326
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp</code>	326
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiChooseVariable.hpp</code>	327
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiColCut.hpp</code>	327
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiCollections.hpp</code>	328
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiConfig.h</code>	328
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiCut.hpp</code>	329
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiCuts.hpp</code>	329
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiPresolve.hpp</code>	330
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiRowCut.hpp</code>	330
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiRowCutDebugger.hpp</code>	330
Provides a facility to validate cut constraints to ensure that they do not cut off a given solution	330
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverBranch.hpp</code>	331
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverInterface.hpp</code>	331
<code>/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverParameters.hpp</code>	331
<code>/home/ted/COIN/trunk/Osi/src/OsiCommonTest/OsiUnitTests.hpp</code>	333
Utility methods for OSI unit tests	333
<code>/home/ted/COIN/trunk/Osi/src/OsiCpx/OsiCpxSolverInterface.hpp</code>	338
<code>/home/ted/COIN/trunk/Osi/src/OsiGlpk/OsiGlpkSolverInterface.hpp</code>	339
<code>/home/ted/COIN/trunk/Osi/src/OsiGrb/OsiGrbSolverInterface.hpp</code>	340
<code>/home/ted/COIN/trunk/Osi/src/OsiMsk/OsiMskSolverInterface.hpp</code>	340

/home/ted/COIN/trunk/Osi/src/OsiSpx/OsiSpxSolverInterface.hpp	341
/home/ted/COIN/trunk/Osi/src/OsiXpr/OsiXprSolverInterface.hpp	342

7 Namespace Documentation

7.1 OsiUnitTest Namespace Reference

A namespace so we can define a few 'global' variables to use during tests.

Classes

- class [TestOutcome](#)
A single test outcome record.
- class [TestOutcomes](#)
Utility class to maintain a list of test outcomes.

Functions

- void [failureMessage](#) (const std::string &solverName, const std::string &message)
Print an error message.
- void [failureMessage](#) (const [OsiSolverInterface](#) &si, const std::string &message)
- void [failureMessage](#) (const std::string &solverName, const std::string &testname, const std::string &testcond)
Print an error message, specifying the test name and condition.
- void [failureMessage](#) (const [OsiSolverInterface](#) &si, const std::string &testname, const std::string &testcond)
- void [testingMessage](#) (const char *const msg)
Print a message.
- bool [equivalentVectors](#) (const [OsiSolverInterface](#) *si1, const [OsiSolverInterface](#) *si2, double tol, const double *v1, const double *v2, int size)
Utility method to check equality.
- bool [compareProblems](#) ([OsiSolverInterface](#) *osi1, [OsiSolverInterface](#) *osi2)
Compare two problems for equality.
- bool [isEquivalent](#) (const CoinPackedVectorBase &pv, int n, const double *fv)
Compare a packed vector with an expanded vector.
- bool [processParameters](#) (int argc, const char **argv, std::map< std::string, std::string > &parms, const std::map< std::string, int > &ignorekeywords=std::map< std::string, int >())
Process command line parameters.
- template<typename Component >
bool [OsiUnitTestAssertSeverityExpected](#) (bool condition, const char *condition_str, const char *filename, int line, const Component &component, const std::string &testname, [TestOutcome::SeverityLevel](#) severity, bool expected)

Variables

- unsigned int [verbosity](#)
Verbosity level of unit tests.
- unsigned int [haltonerror](#)
Behaviour on failing a test.
- [TestOutcomes outcomes](#)
Test outcomes.

7.1.1 Detailed Description

A namespace so we can define a few 'global' variables to use during tests.

7.1.2 Function Documentation

7.1.2.1 void OsiUnitTest::failureMessage (const std::string & *solverName*, const std::string & *message*)

Print an error message.

Formatted as "XxxSolverInterface testing issue: message" where Xxx is the string provided as *solverName*.

Flushes std::cout before printing to std::cerr.

7.1.2.2 void OsiUnitTest::failureMessage (const OsiSolverInterface & *si*, const std::string & *message*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.1.2.3 void OsiUnitTest::failureMessage (const std::string & *solverName*, const std::string & *testname*, const std::string & *testcond*)

Print an error message, specifying the test name and condition.

Formatted as "XxxSolverInterface testing issue: testname failed: testcond" where Xxx is the OsiStrParam::OsiSolverName parameter of the *si*. Flushes std::cout before printing to std::cerr.

7.1.2.4 void OsiUnitTest::failureMessage (const OsiSolverInterface & *si*, const std::string & *testname*, const std::string & *testcond*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.1.2.5 void OsiUnitTest::testingMessage (const char *const *msg*)

Print a message.

Prints the message as given. Flushes std::cout before printing to std::cerr.

7.1.2.6 bool OsiUnitTest::equivalentVectors (const OsiSolverInterface * *si1*, const OsiSolverInterface * *si2*, double *tol*, const double * *v1*, const double * *v2*, int *size*)

Utility method to check equality.

Tests for equality using CoinRelFltEq with tolerance *tol*. Understands the notion of solver infinity and obtains the value for infinity from the solver interfaces supplied as parameters.

7.1.2.7 bool OsiUnitTest::compareProblems (OsiSolverInterface * *osi1*, OsiSolverInterface * *osi2*)

Compare two problems for equality.

Compares the problems held in the two solvers: constraint matrix, row and column bounds, column type, and objective. Rows are checked using upper and lower bounds and using sense, bound, and range.

7.1.2.8 bool OsiUnitTest::isEquivalent (const CoinPackedVectorBase & *pV*, int *n*, const double * *fv*)

Compare a packed vector with an expanded vector.

Checks that all values present in the packed vector are present in the full vector and checks that there are no extra

entries in the full vector. Uses CoinRelFltEq with the default tolerance.

```
7.1.2.9  bool OsiUnitTest::processParameters ( int argc, const char ** argv, std::map< std::string, std::string > & parms, const
        std::map< std::string, int > & ignorekeywords = std::map< std::string, int >() )
```

Process command line parameters.

An unrecognised keyword which is not in the `ignorekeywords` map will trigger the help message and a return value of false. For each keyword in `ignorekeywords`, you can specify the number of following parameters that should be ignored.

This should be replaced with the one of the standard CoinUtils parameter mechanisms.

```
7.1.2.10 template<typename Component > bool OsiUnitTest::OsiUnitTestAssertSeverityExpected ( bool condition, const
        char * condition_str, const char * filename, int line, const Component & component, const std::string & testname,
        TestOutcome::SeverityLevel severity, bool expected )
```

Definition at line 236 of file OsiUnitTests.hpp.

7.1.3 Variable Documentation

7.1.3.1 unsigned int OsiUnitTest::verbosity

Verbosity level of unit tests.

0 (default) for minimal output; larger numbers produce more output

7.1.3.2 unsigned int OsiUnitTest::haltonerror

Behaviour on failing a test.

- 0 (= default) continue
- 1 press any key to continue
- 2 stop with abort()

7.1.3.3 TestOutcomes OsiUnitTest::outcomes

Test outcomes.

A global [TestOutcomes](#) object to store test outcomes during the run of the unit test for an OSI.

7.2 soplex Namespace Reference

8 Class Documentation

8.1 OsiSolverInterface::ApplyCutsReturnCode Class Reference

Internal class for obtaining status from the `applyCuts` method.

```
#include <OsiSolverInterface.hpp>
```

Public Member Functions

Constructors and destructors

- [ApplyCutsReturnCode](#) ()
Default constructor.
- [ApplyCutsReturnCode](#) (const [ApplyCutsReturnCode](#) &rhs)
Copy constructor.
- [ApplyCutsReturnCode](#) & [operator=](#) (const [ApplyCutsReturnCode](#) &rhs)
Assignment operator.
- [~ApplyCutsReturnCode](#) ()
Destructor.

Accessing return code attributes

- int [getNumInconsistent](#) () const
Number of logically inconsistent cuts.
- int [getNumInconsistentWrtIntegerModel](#) () const
Number of cuts inconsistent with the current model.
- int [getNumInfeasible](#) () const
Number of cuts that cause obvious infeasibility.
- int [getNumIneffective](#) () const
Number of redundant or ineffective cuts.
- int [getNumApplied](#) () const
Number of cuts applied.

Private Member Functions

Private methods

- void [incrementInternallyInconsistent](#) ()
Increment logically inconsistent cut counter.
- void [incrementExternallyInconsistent](#) ()
Increment model-inconsistent counter.
- void [incrementInfeasible](#) ()
Increment infeasible cut counter.
- void [incrementIneffective](#) ()
Increment ineffective cut counter.
- void [incrementApplied](#) ()
Increment applied cut counter.

Private Attributes

Private member data

- int [intInconsistent_](#)
Counter for logically inconsistent cuts.
- int [extInconsistent_](#)
Counter for model-inconsistent cuts.
- int [infeasible_](#)
Counter for infeasible cuts.
- int [ineffective_](#)
Counter for ineffective cuts.
- int [applied_](#)
Counter for applied cuts.

Friends

- class [OsiSolverInterface](#)
- class [OsiClpSolverInterface](#)
- class [OsiGrbSolverInterface](#)

8.1.1 Detailed Description

Internal class for obtaining status from the applyCuts method.

Definition at line 74 of file OsiSolverInterface.hpp.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 OsiSolverInterface::ApplyCutsReturnCode::ApplyCutsReturnCode () `[inline]`

Default constructor.

Definition at line 83 of file OsiSolverInterface.hpp.

8.1.2.2 OsiSolverInterface::ApplyCutsReturnCode::ApplyCutsReturnCode (const ApplyCutsReturnCode & rhs) `[inline]`

Copy constructor.

Definition at line 90 of file OsiSolverInterface.hpp.

8.1.2.3 OsiSolverInterface::ApplyCutsReturnCode::~~ApplyCutsReturnCode () `[inline]`

Destructor.

Definition at line 109 of file OsiSolverInterface.hpp.

8.1.3 Member Function Documentation

8.1.3.1 ApplyCutsReturnCode& OsiSolverInterface::ApplyCutsReturnCode::operator= (const ApplyCutsReturnCode & rhs) `[inline]`

Assignment operator.

Definition at line 97 of file OsiSolverInterface.hpp.

8.1.3.2 int OsiSolverInterface::ApplyCutsReturnCode::getNumInconsistent () const `[inline]`

Number of logically inconsistent cuts.

Definition at line 115 of file OsiSolverInterface.hpp.

8.1.3.3 int OsiSolverInterface::ApplyCutsReturnCode::getNumInconsistentWrtIntegerModel () const `[inline]`

Number of cuts inconsistent with the current model.

Definition at line 118 of file OsiSolverInterface.hpp.

8.1.3.4 int OsiSolverInterface::ApplyCutsReturnCode::getNumInfeasible () const `[inline]`

Number of cuts that cause obvious infeasibility.

Definition at line 121 of file OsiSolverInterface.hpp.

8.1.3.5 `int OsiSolverInterface::ApplyCutsReturnCode::getNumIneffective () const [inline]`

Number of redundant or ineffective cuts.

Definition at line 124 of file OsiSolverInterface.hpp.

8.1.3.6 `int OsiSolverInterface::ApplyCutsReturnCode::getNumApplied () const [inline]`

Number of cuts applied.

Definition at line 127 of file OsiSolverInterface.hpp.

8.1.3.7 `void OsiSolverInterface::ApplyCutsReturnCode::incrementInternallyInconsistent () [inline],[private]`

Increment logically inconsistent cut counter.

Definition at line 135 of file OsiSolverInterface.hpp.

8.1.3.8 `void OsiSolverInterface::ApplyCutsReturnCode::incrementExternallyInconsistent () [inline],[private]`

Increment model-inconsistent counter.

Definition at line 137 of file OsiSolverInterface.hpp.

8.1.3.9 `void OsiSolverInterface::ApplyCutsReturnCode::incrementInfeasible () [inline],[private]`

Increment infeasible cut counter.

Definition at line 139 of file OsiSolverInterface.hpp.

8.1.3.10 `void OsiSolverInterface::ApplyCutsReturnCode::incrementIneffective () [inline],[private]`

Increment ineffective cut counter.

Definition at line 141 of file OsiSolverInterface.hpp.

8.1.3.11 `void OsiSolverInterface::ApplyCutsReturnCode::incrementApplied () [inline],[private]`

Increment applied cut counter.

Definition at line 143 of file OsiSolverInterface.hpp.

8.1.4 Friends And Related Function Documentation

8.1.4.1 `friend class OsiSolverInterface [friend]`

Definition at line 75 of file OsiSolverInterface.hpp.

8.1.4.2 `friend class OsiClpSolverInterface [friend]`

Definition at line 76 of file OsiSolverInterface.hpp.

8.1.4.3 `friend class OsiGrbSolverInterface [friend]`

Definition at line 77 of file OsiSolverInterface.hpp.

8.1.5 Member Data Documentation

8.1.5.1 `int OsiSolverInterface::ApplyCutsReturnCode::intInconsistent_` [private]

Counter for logically inconsistent cuts.

Definition at line 149 of file `OsiSolverInterface.hpp`.

8.1.5.2 `int OsiSolverInterface::ApplyCutsReturnCode::extInconsistent_` [private]

Counter for model-inconsistent cuts.

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

8.1.5.3 `int OsiSolverInterface::ApplyCutsReturnCode::infeasible_` [private]

Counter for infeasible cuts.

Definition at line 153 of file `OsiSolverInterface.hpp`.

8.1.5.4 `int OsiSolverInterface::ApplyCutsReturnCode::ineffective_` [private]

Counter for ineffective cuts.

Definition at line 155 of file `OsiSolverInterface.hpp`.

8.1.5.5 `int OsiSolverInterface::ApplyCutsReturnCode::applied_` [private]

Counter for applied cuts.

Definition at line 157 of file `OsiSolverInterface.hpp`.

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

- [/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverInterface.hpp](#)

8.2 `OsiCuts::const_iterator` Class Reference

Const Iterator.

```
#include <OsiCuts.hpp>
```

Public Types

- typedef `std::bidirectional_iterator_tag` [iterator_category](#)
- typedef [OsiCut](#) * [value_type](#)
- typedef `size_t` [difference_type](#)
- typedef [OsiCut](#) ** [pointer](#)
- typedef [OsiCut](#) *& [reference](#)

Public Member Functions

- [const_iterator](#) (const [OsiCuts](#) &cuts)
- [const_iterator](#) (const [const_iterator](#) &src)
- [const_iterator](#) & [operator=](#) (const [const_iterator](#) &rhs)

- [~const_iterator](#) ()
- const [OsiCut](#) * [operator*](#) () const
- [const_iterator](#) [operator++](#) ()
- [const_iterator](#) [operator++](#) (int)
- bool [operator==](#) (const [const_iterator](#) &it) const
- bool [operator!=](#) (const [const_iterator](#) &it) const
- bool [operator<](#) (const [const_iterator](#) &it) const

Private Member Functions

- [const_iterator](#) ()
- [const_iterator](#) [begin](#) ()
- [const_iterator](#) [end](#) ()

Private Attributes

- const [OsiCuts](#) * [cutsPtr_](#)
- int [rowCutIndex_](#)
- int [colCutIndex_](#)
- const [OsiCut](#) * [cutP_](#)

Friends

- class [OsiCuts](#)

8.2.1 Detailed Description

Const Iterator.

This is a class for iterating over the collection of cuts.

Definition at line 74 of file OsiCuts.hpp.

8.2.2 Member Typedef Documentation

8.2.2.1 `typedef std::bidirectional_iterator_tag OsiCuts::const_iterator::iterator_category`

Definition at line 77 of file OsiCuts.hpp.

8.2.2.2 `typedef OsiCut* OsiCuts::const_iterator::value_type`

Definition at line 78 of file OsiCuts.hpp.

8.2.2.3 `typedef size_t OsiCuts::const_iterator::difference_type`

Definition at line 79 of file OsiCuts.hpp.

8.2.2.4 `typedef OsiCut** OsiCuts::const_iterator::pointer`

Definition at line 80 of file OsiCuts.hpp.

8.2.2.5 typedef OsiCut*& OsiCuts::const_iterator::reference

Definition at line 81 of file OsiCuts.hpp.

8.2.3 Constructor & Destructor Documentation

8.2.3.1 OsiCuts::const_iterator::const_iterator (const OsiCuts & cuts)

8.2.3.2 OsiCuts::const_iterator::const_iterator (const const_iterator & src)

8.2.3.3 OsiCuts::const_iterator::~~const_iterator ()

8.2.3.4 OsiCuts::const_iterator::const_iterator () [inline],[private]

8.2.4 Member Function Documentation

8.2.4.1 const_iterator& OsiCuts::const_iterator::operator= (const const_iterator & rhs)

8.2.4.2 const OsiCut* OsiCuts::const_iterator::operator* () const [inline]

Definition at line 88 of file OsiCuts.hpp.

8.2.4.3 const_iterator OsiCuts::const_iterator::operator++ ()

8.2.4.4 const_iterator OsiCuts::const_iterator::operator++ (int) [inline]

Definition at line 92 of file OsiCuts.hpp.

8.2.4.5 bool OsiCuts::const_iterator::operator== (const const_iterator & it) const [inline]

Definition at line 99 of file OsiCuts.hpp.

8.2.4.6 bool OsiCuts::const_iterator::operator!= (const const_iterator & it) const [inline]

Definition at line 103 of file OsiCuts.hpp.

8.2.4.7 bool OsiCuts::const_iterator::operator< (const const_iterator & it) const [inline]

Definition at line 107 of file OsiCuts.hpp.

8.2.4.8 const_iterator OsiCuts::const_iterator::begin () [private]

8.2.4.9 const_iterator OsiCuts::const_iterator::end () [private]

8.2.5 Friends And Related Function Documentation

8.2.5.1 friend class OsiCuts [friend]

Definition at line 75 of file OsiCuts.hpp.

8.2.6 Member Data Documentation

8.2.6.1 const OsiCuts* OsiCuts::const_iterator::cutsPtr_ [private]

Definition at line 115 of file OsiCuts.hpp.

8.2.6.2 `int OsiCuts::const_iterator::rowCutIndex_ [private]`

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

8.2.6.3 `int OsiCuts::const_iterator::colCutIndex_ [private]`

Definition at line 117 of file `OsiCuts.hpp`.

8.2.6.4 `const OsiCut* OsiCuts::const_iterator::cutP_ [private]`

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

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

- [/home/ted/COIN/trunk/Osi/src/Osi/OsiCuts.hpp](#)

8.3 glp_prob Struct Reference

```
#include <OsiGlpkSolverInterface.hpp>
```

Public Attributes

- `double _opaque_prob [100]`

8.3.1 Detailed Description

Definition at line 30 of file `OsiGlpkSolverInterface.hpp`.

8.3.2 Member Data Documentation

8.3.2.1 `double glp_prob::_opaque_prob[100]`

Definition at line 30 of file `OsiGlpkSolverInterface.hpp`.

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

- [/home/ted/COIN/trunk/Osi/src/OsiGlpk/OsiGlpkSolverInterface.hpp](#)

8.4 OsiCuts::iterator Class Reference

Iterator.

```
#include <OsiCuts.hpp>
```

Public Member Functions

- `iterator (OsiCuts &cuts)`
- `iterator (const iterator &src)`
- `iterator & operator= (const iterator &rhs)`
- `~iterator ()`
- `OsiCut * operator* () const`

- `iterator operator++ ()`
- `iterator operator++ (int)`
- `bool operator== (const iterator &it) const`
- `bool operator!= (const iterator &it) const`
- `bool operator< (const iterator &it) const`

Private Member Functions

- `iterator ()`
- `iterator begin ()`
- `iterator end ()`

Private Attributes

- `OsiCuts & cuts_`
- `int rowCutIndex_`
- `int colCutIndex_`
- `OsiCut * cutP_`

Friends

- class `OsiCuts`

8.4.1 Detailed Description

Iterator.

This is a class for iterating over the collection of cuts.

Definition at line 30 of file `OsiCuts.hpp`.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 `OsiCuts::iterator::iterator (OsiCuts & cuts)`

8.4.2.2 `OsiCuts::iterator::iterator (const iterator & src)`

8.4.2.3 `OsiCuts::iterator::~~iterator ()`

8.4.2.4 `OsiCuts::iterator::iterator ()` `[private]`

8.4.3 Member Function Documentation

8.4.3.1 `iterator& OsiCuts::iterator::operator= (const iterator & rhs)`

8.4.3.2 `OsiCut* OsiCuts::iterator::operator* () const` `[inline]`

Definition at line 37 of file `OsiCuts.hpp`.

8.4.3.3 `iterator OsiCuts::iterator::operator++ ()`

8.4.3.4 `iterator OsiCuts::iterator::operator++ (int) [inline]`

Definition at line 40 of file OsiCuts.hpp.

8.4.3.5 `bool OsiCuts::iterator::operator== (const iterator & it) const [inline]`

Definition at line 47 of file OsiCuts.hpp.

8.4.3.6 `bool OsiCuts::iterator::operator!= (const iterator & it) const [inline]`

Definition at line 51 of file OsiCuts.hpp.

8.4.3.7 `bool OsiCuts::iterator::operator< (const iterator & it) const [inline]`

Definition at line 55 of file OsiCuts.hpp.

8.4.3.8 `iterator OsiCuts::iterator::begin () [private]`

8.4.3.9 `iterator OsiCuts::iterator::end () [private]`

8.4.4 Friends And Related Function Documentation

8.4.4.1 `friend class OsiCuts [friend]`

Definition at line 31 of file OsiCuts.hpp.

8.4.5 Member Data Documentation

8.4.5.1 `OsiCuts& OsiCuts::iterator::cuts_ [private]`

Definition at line 64 of file OsiCuts.hpp.

8.4.5.2 `int OsiCuts::iterator::rowCutIndex_ [private]`

Definition at line 65 of file OsiCuts.hpp.

8.4.5.3 `int OsiCuts::iterator::colCutIndex_ [private]`

Definition at line 66 of file OsiCuts.hpp.

8.4.5.4 `OsiCut* OsiCuts::iterator::cutP_ [private]`

Definition at line 67 of file OsiCuts.hpp.

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

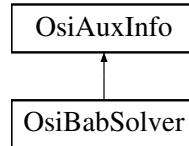
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiCuts.hpp](#)

8.5 OsiAuxInfo Class Reference

This class allows for a more structured use of algorithmic tweaking to an [OsiSolverInterface](#).

```
#include <OsiAuxInfo.hpp>
```

Inheritance diagram for OsiAuxInfo:



Public Member Functions

- [OsiAuxInfo](#) (void *appData=NULL)
- [OsiAuxInfo](#) (const [OsiAuxInfo](#) &rhs)
- virtual [~OsiAuxInfo](#) ()
- virtual [OsiAuxInfo * clone](#) () const
Clone.
- [OsiAuxInfo & operator=](#) (const [OsiAuxInfo](#) &rhs)
Assignment operator.
- void * [getApplicationData](#) () const
Get application data.

Protected Attributes

- void * [appData_](#)
Pointer to user-defined data structure.

8.5.1 Detailed Description

This class allows for a more structured use of algorithmic tweaking to an [OsiSolverInterface](#).

It is designed to replace the simple use of appData_ pointer.

This has been done to make it easier to use NonLinear solvers and other exotic beasts in a branch and bound mode. After this class definition there is one for a derived class for just such a purpose.

Definition at line 21 of file OsiAuxInfo.hpp.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 [OsiAuxInfo::OsiAuxInfo](#) (void * *appData* = NULL)

8.5.2.2 [OsiAuxInfo::OsiAuxInfo](#) (const [OsiAuxInfo](#) & *rhs*)

8.5.2.3 [OsiAuxInfo::~~OsiAuxInfo](#) () [virtual]

8.5.3 Member Function Documentation

8.5.3.1 [OsiAuxInfo* OsiAuxInfo::clone](#) () const [virtual]

Clone.

Reimplemented in [OsiBabSolver](#).

8.5.3.2 [OsiAuxInfo& OsiAuxInfo::operator=](#) (const [OsiAuxInfo](#) & *rhs*)

Assignment operator.

8.5.3.3 void* OsiAuxInfo::getApplicationData () const [inline]

Get application data.

Definition at line 37 of file OsiAuxInfo.hpp.

8.5.4 Member Data Documentation

8.5.4.1 void* OsiAuxInfo::appData_ [protected]

Pointer to user-defined data structure.

Definition at line 41 of file OsiAuxInfo.hpp.

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

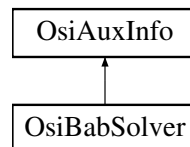
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiAuxInfo.hpp](#)

8.6 OsiBabSolver Class Reference

This class allows for the use of more exotic solvers e.g.

```
#include <OsiAuxInfo.hpp>
```

Inheritance diagram for OsiBabSolver:



Public Member Functions

- [OsiBabSolver](#) (int [solverType](#)=0)
- [OsiBabSolver](#) (const [OsiBabSolver](#) &rhs)
- virtual [~OsiBabSolver](#) ()
- virtual [OsiAuxInfo](#) * [clone](#) () const
Clone.
- [OsiBabSolver](#) & [operator=](#) (const [OsiBabSolver](#) &rhs)
Assignment operator.
- void [setSolver](#) (const [OsiSolverInterface](#) *solver)
Update solver.
- void [setSolver](#) (const [OsiSolverInterface](#) &solver)
Update solver.
- int [solution](#) (double &objectiveValue, double *newSolution, int numberColumns)
returns 0 if no heuristic solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value numberColumns is size of newSolution
- void [setSolution](#) (const double *solution, int numberColumns, double objectiveValue)
Set solution and objective value.
- bool [hasSolution](#) (double &solutionValue, double *solution)
returns true if the object stores a solution, false otherwise.

- void `setSolverType` (int value)
Sets solver type 0 - normal LP solver 1 - DW - may also return heuristic solutions 2 - NLP solver or similar - can't compute objective value just from solution check solver to see if feasible and what objective value is.
- int `solverType` () const
gets solver type 0 - normal LP solver 1 - DW - may also return heuristic solutions 2 - NLP solver or similar - can't compute objective value just from solution check this (rather than solver) to see if feasible and what objective value is
- bool `solutionAddsCuts` () const
Return true if getting solution may add cuts so hot start etc will be obsolete.
- bool `alwaysTryCutsAtRootNode` () const
Return true if we should try cuts at root even if looks satisfied.
- bool `solverAccurate` () const
Returns true if can use solver objective or feasible values, otherwise use mipBound etc.
- bool `reducedCostsAccurate` () const
Returns true if can use reduced costs for fixing.
- double `mipBound` () const
Get objective (well mip bound)
- bool `mipFeasible` () const
Returns true if node feasible.
- void `setMipBound` (double value)
Set mip bound (only used for some solvers)
- double `bestObjectiveValue` () const
Get objective value of saved solution.
- bool `tryCuts` () const
Says whether we want to try cuts at all.
- bool `warmStart` () const
Says whether we have a warm start (so can do strong branching)
- int `extraCharacteristics` () const
Get bit mask for odd actions of solvers 1 - solution or bound arrays may move in mysterious ways e.g.
- void `setExtraCharacteristics` (int value)
Set bit mask for odd actions of solvers 1 - solution or bound arrays may move in mysterious ways e.g.
- const double * `beforeLower` () const
Pointer to lower bounds before branch (only if extraCharacteristics set)
- void `setBeforeLower` (const double *array)
Set pointer to lower bounds before branch (only if extraCharacteristics set)
- const double * `beforeUpper` () const
Pointer to upper bounds before branch (only if extraCharacteristics set)
- void `setBeforeUpper` (const double *array)
Set pointer to upper bounds before branch (only if extraCharacteristics set)

Protected Attributes

- double `bestObjectiveValue_`
Objective value of best solution (if there is one) (minimization)
- double `mipBound_`
Current lower bound on solution (if > 1.0e50 infeasible)
- const `OsiSolverInterface` * `solver_`
Solver to use for getting/setting solutions etc.

- double * [bestSolution_](#)
Best integer feasible solution.
- const double * [beforeLower_](#)
Pointer to lower bounds before branch (only if extraCharacteristics set)
- const double * [beforeUpper_](#)
Pointer to upper bounds before branch (only if extraCharacteristics set)
- int [solverType_](#)
Solver type 0 - normal LP solver 1 - DW - may also return heuristic solutions 2 - NLP solver or similar - can't compute objective value just from solution check this (rather than solver) to see if feasible and what objective value is.
- int [sizeSolution_](#)
Size of solution.
- int [extraCharacteristics_](#)
Bit mask for odd actions of solvers 1 - solution or bound arrays may move in mysterious ways e.g.

8.6.1 Detailed Description

This class allows for the use of more exotic solvers e.g.

Non-Linear or Volume.

You can derive from this although at present I can't see the need.

Definition at line 49 of file OsiAuxInfo.hpp.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 **OsiBabSolver::OsiBabSolver (int *solverType* = 0)**

8.6.2.2 **OsiBabSolver::OsiBabSolver (const OsiBabSolver & *rhs*)**

8.6.2.3 **virtual OsiBabSolver::~~OsiBabSolver ()** [virtual]

8.6.3 Member Function Documentation

8.6.3.1 **virtual OsiAuxInfo* OsiBabSolver::clone () const** [virtual]

Clone.

Reimplemented from [OsiAuxInfo](#).

8.6.3.2 **OsiBabSolver& OsiBabSolver::operator= (const OsiBabSolver & *rhs*)**

Assignment operator.

8.6.3.3 **void OsiBabSolver::setSolver (const OsiSolverInterface * *solver*)** [inline]

Update solver.

Definition at line 65 of file OsiAuxInfo.hpp.

8.6.3.4 **void OsiBabSolver::setSolver (const OsiSolverInterface & *solver*)** [inline]

Update solver.

Definition at line 68 of file OsiAuxInfo.hpp.

8.6.3.5 `int OsiBabSolver::solution (double & objectiveValue, double * newSolution, int numberColumns)`

returns 0 if no heuristic solution, 1 if valid solution with better objective value than one passed in Sets solution values if good, sets objective value numberColumns is size of newSolution

8.6.3.6 `void OsiBabSolver::setSolution (const double * solution, int numberColumns, double objectiveValue)`

Set solution and objective value.

Number of columns and optimization direction taken from current solver. Size of solution is numberColumns (may be padded or truncated in function)

8.6.3.7 `bool OsiBabSolver::hasSolution (double & solutionValue, double * solution)`

returns true if the object stores a solution, false otherwise.

If there is a solution then solutionValue and solution will be filled out as well. In that case the user needs to allocate solution to be a big enough array.

8.6.3.8 `void OsiBabSolver::setSolverType (int value) [inline]`

Sets solver type 0 - normal LP solver 1 - DW - may also return heuristic solutions 2 - NLP solver or similar - can't compute objective value just from solution check solver to see if feasible and what objective value is.

- may also return heuristic solution

3 - NLP solver or similar - can't compute objective value just from solution check this (rather than solver) to see if feasible and what objective value is. Using Outer Approximation so called lp based

- may also return heuristic solution 4 - normal solver but cuts are needed for integral solution

Definition at line 102 of file OsiAuxInfo.hpp.

8.6.3.9 `int OsiBabSolver::solverType () const [inline]`

gets solver type 0 - normal LP solver 1 - DW - may also return heuristic solutions 2 - NLP solver or similar - can't compute objective value just from solution check this (rather than solver) to see if feasible and what objective value is

- may also return heuristic solution

3 - NLP solver or similar - can't compute objective value just from solution check this (rather than solver) to see if feasible and what objective value is. Using Outer Approximation so called lp based

- may also return heuristic solution 4 - normal solver but cuts are needed for integral solution

Definition at line 116 of file OsiAuxInfo.hpp.

8.6.3.10 `bool OsiBabSolver::solutionAddsCuts () const [inline]`

Return true if getting solution may add cuts so hot start etc will be obsolete.

Definition at line 120 of file OsiAuxInfo.hpp.

8.6.3.11 `bool OsiBabSolver::alwaysTryCutsAtRootNode () const [inline]`

Return true if we should try cuts at root even if looks satisfied.

Definition at line 123 of file OsiAuxInfo.hpp.

8.6.3.12 bool OsiBabSolver::solverAccurate () const [inline]

Returns true if can use solver objective or feasible values, otherwise use mipBound etc.

Definition at line 127 of file OsiAuxInfo.hpp.

8.6.3.13 bool OsiBabSolver::reducedCostsAccurate () const [inline]

Returns true if can use reduced costs for fixing.

Definition at line 130 of file OsiAuxInfo.hpp.

8.6.3.14 double OsiBabSolver::mipBound () const

Get objective (well mip bound)

8.6.3.15 bool OsiBabSolver::mipFeasible () const

Returns true if node feasible.

8.6.3.16 void OsiBabSolver::setMipBound (double value) [inline]

Set mip bound (only used for some solvers)

Definition at line 137 of file OsiAuxInfo.hpp.

8.6.3.17 double OsiBabSolver::bestObjectiveValue () const [inline]

Get objective value of saved solution.

Definition at line 140 of file OsiAuxInfo.hpp.

8.6.3.18 bool OsiBabSolver::tryCuts () const [inline]

Says whether we want to try cuts at all.

Definition at line 143 of file OsiAuxInfo.hpp.

8.6.3.19 bool OsiBabSolver::warmStart () const [inline]

Says whether we have a warm start (so can do strong branching)

Definition at line 146 of file OsiAuxInfo.hpp.

8.6.3.20 int OsiBabSolver::extraCharacteristics () const [inline]

Get bit mask for odd actions of solvers 1 - solution or bound arrays may move in mysterious ways e.g.

cplex 2 - solver may want bounds before branch

Definition at line 152 of file OsiAuxInfo.hpp.

8.6.3.21 void OsiBabSolver::setExtraCharacteristics (int value) [inline]

Set bit mask for odd actions of solvers 1 - solution or bound arrays may move in mysterious ways e.g.

cplex 2 - solver may want bounds before branch

Definition at line 158 of file OsiAuxInfo.hpp.

8.6.3.22 `const double* OsiBabSolver::beforeLower () const` `[inline]`

Pointer to lower bounds before branch (only if extraCharacteristics set)

Definition at line 161 of file OsiAuxInfo.hpp.

8.6.3.23 `void OsiBabSolver::setBeforeLower (const double * array)` `[inline]`

Set pointer to lower bounds before branch (only if extraCharacteristics set)

Definition at line 164 of file OsiAuxInfo.hpp.

8.6.3.24 `const double* OsiBabSolver::beforeUpper () const` `[inline]`

Pointer to upper bounds before branch (only if extraCharacteristics set)

Definition at line 167 of file OsiAuxInfo.hpp.

8.6.3.25 `void OsiBabSolver::setBeforeUpper (const double * array)` `[inline]`

Set pointer to upper bounds before branch (only if extraCharacteristics set)

Definition at line 170 of file OsiAuxInfo.hpp.

8.6.4 Member Data Documentation

8.6.4.1 `double OsiBabSolver::bestObjectiveValue_` `[protected]`

Objective value of best solution (if there is one) (minimization)

Definition at line 174 of file OsiAuxInfo.hpp.

8.6.4.2 `double OsiBabSolver::mipBound_` `[protected]`

Current lower bound on solution (if > 1.0e50 infeasible)

Definition at line 176 of file OsiAuxInfo.hpp.

8.6.4.3 `const OsiSolverInterface* OsiBabSolver::solver_` `[protected]`

Solver to use for getting/setting solutions etc.

Definition at line 178 of file OsiAuxInfo.hpp.

8.6.4.4 `double* OsiBabSolver::bestSolution_` `[protected]`

Best integer feasible solution.

Definition at line 180 of file OsiAuxInfo.hpp.

8.6.4.5 `const double* OsiBabSolver::beforeLower_` `[protected]`

Pointer to lower bounds before branch (only if extraCharacteristics set)

Definition at line 182 of file OsiAuxInfo.hpp.

8.6.4.6 `const double* OsiBabSolver::beforeUpper_` `[protected]`

Pointer to upper bounds before branch (only if extraCharacteristics set)

Definition at line 184 of file OsiAuxInfo.hpp.

8.6.4.7 int OsiBabSolver::solverType_ [protected]

Solver type 0 - normal LP solver 1 - DW - may also return heuristic solutions 2 - NLP solver or similar - can't compute objective value just from solution check this (rather than solver) to see if feasible and what objective value is.

- may also return heuristic solution

3 - NLP solver or similar - can't compute objective value just from solution check this (rather than solver) to see if feasible and what objective value is. Using Outer Approximation so called lp based

- may also return heuristic solution

Definition at line 196 of file OsiAuxInfo.hpp.

8.6.4.8 int OsiBabSolver::sizeSolution_ [protected]

Size of solution.

Definition at line 198 of file OsiAuxInfo.hpp.

8.6.4.9 int OsiBabSolver::extraCharacteristics_ [protected]

Bit mask for odd actions of solvers 1 - solution or bound arrays may move in mysterious ways e.g.

cplex 2 - solver may want bounds before branch

Definition at line 203 of file OsiAuxInfo.hpp.

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

- [/home/ted/COIN/trunk/Osi/src/Osi/OsiAuxInfo.hpp](#)

8.7 OsiBranchingInformation Class Reference

```
#include <OsiBranchingObject.hpp>
```

Public Member Functions

- [OsiBranchingInformation](#) ()
Default Constructor.
- [OsiBranchingInformation](#) (const [OsiSolverInterface](#) *solver, bool normalSolver, bool copySolution=false)
Useful Constructor (normalSolver true if has matrix etc etc) copySolution true if constructot should make a copy.
- [OsiBranchingInformation](#) (const [OsiBranchingInformation](#) &)
Copy constructor.
- [OsiBranchingInformation](#) & operator= (const [OsiBranchingInformation](#) &rhs)
Assignment operator.
- virtual [OsiBranchingInformation](#) * clone () const
Clone.
- virtual ~[OsiBranchingInformation](#) ()
Destructor.

Public Attributes

- int [stateOfSearch_](#)
data
- double [objectiveValue_](#)
Value of objective function (in minimization sense)
- double [cutoff_](#)
Value of objective cutoff (in minimization sense)
- double [direction_](#)
Direction 1.0 for minimization, -1.0 for maximization.
- double [integerTolerance_](#)
Integer tolerance.
- double [primalTolerance_](#)
Primal tolerance.
- double [timeRemaining_](#)
Maximum time remaining before stopping on time.
- double [defaultDual_](#)
Dual to use if row bound violated (if negative then pseudoShadowPrices off)
- const [OsiSolverInterface](#) * [solver_](#)
Pointer to solver.
- int [numberColumns_](#)
The number of columns.
- const double * [lower_](#)
Pointer to current lower bounds on columns.
- const double * [solution_](#)
Pointer to current solution.
- const double * [upper_](#)
Pointer to current upper bounds on columns.
- const double * [hotstartSolution_](#)
Highly optional target (hot start) solution.
- const double * [pi_](#)
Pointer to duals.
- const double * [rowActivity_](#)
Pointer to row activity.
- const double * [objective_](#)
Objective.
- const double * [rowLower_](#)
Pointer to current lower bounds on rows.
- const double * [rowUpper_](#)
Pointer to current upper bounds on rows.
- const double * [elementByColumn_](#)
Elements in column copy of matrix.
- const CoinBigIndex * [columnStart_](#)
Column starts.
- const int * [columnLength_](#)
Column lengths.
- const int * [row_](#)

Row indices.

- double * [usefulRegion_](#)

Useful region of length $\text{CoinMax}(\text{numberColumns}, 2 * \text{numberRows})$ This is allocated and deleted before [OsiObject::infeasibility](#) It is zeroed on entry and should be so on exit It only exists if `defaultDual_ >= 0.0`.

- int * [indexRegion_](#)

Useful index region to go with `usefulRegion_`.

- int [numberSolutions_](#)

Number of solutions found.

- int [numberBranchingSolutions_](#)

Number of branching solutions found (i.e. exclude heuristics)

- int [depth_](#)

Depth in tree.

- bool [owningSolution_](#)

TEMP.

8.7.1 Detailed Description

Definition at line 367 of file `OsiBranchingObject.hpp`.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 `OsiBranchingInformation::OsiBranchingInformation ()`

Default Constructor.

8.7.2.2 `OsiBranchingInformation::OsiBranchingInformation (const OsiSolverInterface * solver, bool normalSolver, bool copySolution = false)`

Useful Constructor (normalSolver true if has matrix etc etc) copySolution true if constructot should make a copy.

8.7.2.3 `OsiBranchingInformation::OsiBranchingInformation (const OsiBranchingInformation &)`

Copy constructor.

8.7.2.4 `virtual OsiBranchingInformation::~~OsiBranchingInformation () [virtual]`

Destructor.

8.7.3 Member Function Documentation

8.7.3.1 `OsiBranchingInformation& OsiBranchingInformation::operator= (const OsiBranchingInformation & rhs)`

Assignment operator.

8.7.3.2 `virtual OsiBranchingInformation* OsiBranchingInformation::clone () const [virtual]`

Clone.

8.7.4 Member Data Documentation

8.7.4.1 `int OsiBranchingInformation::stateOfSearch_`

data

State of search 0 - no solution 1 - only heuristic solutions 2 - branched to a solution 3 - no solution but many nodes

Definition at line 402 of file OsiBranchingObject.hpp.

8.7.4.2 `double OsiBranchingInformation::objectiveValue_`

Value of objective function (in minimization sense)

Definition at line 404 of file OsiBranchingObject.hpp.

8.7.4.3 `double OsiBranchingInformation::cutoff_`

Value of objective cutoff (in minimization sense)

Definition at line 406 of file OsiBranchingObject.hpp.

8.7.4.4 `double OsiBranchingInformation::direction_`

Direction 1.0 for minimization, -1.0 for maximization.

Definition at line 408 of file OsiBranchingObject.hpp.

8.7.4.5 `double OsiBranchingInformation::integerTolerance_`

Integer tolerance.

Definition at line 410 of file OsiBranchingObject.hpp.

8.7.4.6 `double OsiBranchingInformation::primalTolerance_`

Primal tolerance.

Definition at line 412 of file OsiBranchingObject.hpp.

8.7.4.7 `double OsiBranchingInformation::timeRemaining_`

Maximum time remaining before stopping on time.

Definition at line 414 of file OsiBranchingObject.hpp.

8.7.4.8 `double OsiBranchingInformation::defaultDual_`

Dual to use if row bound violated (if negative then pseudoShadowPrices off)

Definition at line 416 of file OsiBranchingObject.hpp.

8.7.4.9 `const OsiSolverInterface* OsiBranchingInformation::solver_` [mutable]

Pointer to solver.

Definition at line 418 of file OsiBranchingObject.hpp.

8.7.4.10 `int OsiBranchingInformation::numberColumns_`

The number of columns.

Definition at line 420 of file OsiBranchingObject.hpp.

8.7.4.11 `const double* OsiBranchingInformation::lower_` `[mutable]`

Pointer to current lower bounds on columns.

Definition at line 422 of file OsiBranchingObject.hpp.

8.7.4.12 `const double* OsiBranchingInformation::solution_` `[mutable]`

Pointer to current solution.

Definition at line 424 of file OsiBranchingObject.hpp.

8.7.4.13 `const double* OsiBranchingInformation::upper_` `[mutable]`

Pointer to current upper bounds on columns.

Definition at line 426 of file OsiBranchingObject.hpp.

8.7.4.14 `const double* OsiBranchingInformation::hotstartSolution_`

Highly optional target (hot start) solution.

Definition at line 428 of file OsiBranchingObject.hpp.

8.7.4.15 `const double* OsiBranchingInformation::pi_`

Pointer to duals.

Definition at line 430 of file OsiBranchingObject.hpp.

8.7.4.16 `const double* OsiBranchingInformation::rowActivity_`

Pointer to row activity.

Definition at line 432 of file OsiBranchingObject.hpp.

8.7.4.17 `const double* OsiBranchingInformation::objective_`

Objective.

Definition at line 434 of file OsiBranchingObject.hpp.

8.7.4.18 `const double* OsiBranchingInformation::rowLower_`

Pointer to current lower bounds on rows.

Definition at line 436 of file OsiBranchingObject.hpp.

8.7.4.19 `const double* OsiBranchingInformation::rowUpper_`

Pointer to current upper bounds on rows.

Definition at line 438 of file OsiBranchingObject.hpp.

8.7.4.20 `const double* OsiBranchingInformation::elementByColumn_`

Elements in column copy of matrix.

Definition at line 440 of file OsiBranchingObject.hpp.

8.7.4.21 const CoinBigIndex* OsiBranchingInformation::columnStart_

Column starts.

Definition at line 442 of file OsiBranchingObject.hpp.

8.7.4.22 const int* OsiBranchingInformation::columnLength_

Column lengths.

Definition at line 444 of file OsiBranchingObject.hpp.

8.7.4.23 const int* OsiBranchingInformation::row_

Row indices.

Definition at line 446 of file OsiBranchingObject.hpp.

8.7.4.24 double* OsiBranchingInformation::usefulRegion_

Useful region of length `CoinMax(numberColumns,2*numberRows)` This is allocated and deleted before [OsiObject::infeasibility](#) It is zeroed on entry and should be so on exit It only exists if `defaultDual_ >= 0.0`.

Definition at line 452 of file OsiBranchingObject.hpp.

8.7.4.25 int* OsiBranchingInformation::indexRegion_

Useful index region to go with `usefulRegion_`.

Definition at line 454 of file OsiBranchingObject.hpp.

8.7.4.26 int OsiBranchingInformation::numberSolutions_

Number of solutions found.

Definition at line 456 of file OsiBranchingObject.hpp.

8.7.4.27 int OsiBranchingInformation::numberBranchingSolutions_

Number of branching solutions found (i.e. exclude heuristics)

Definition at line 458 of file OsiBranchingObject.hpp.

8.7.4.28 int OsiBranchingInformation::depth_

Depth in tree.

Definition at line 460 of file OsiBranchingObject.hpp.

8.7.4.29 bool OsiBranchingInformation::owningSolution_

TEMP.

Definition at line 462 of file OsiBranchingObject.hpp.

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

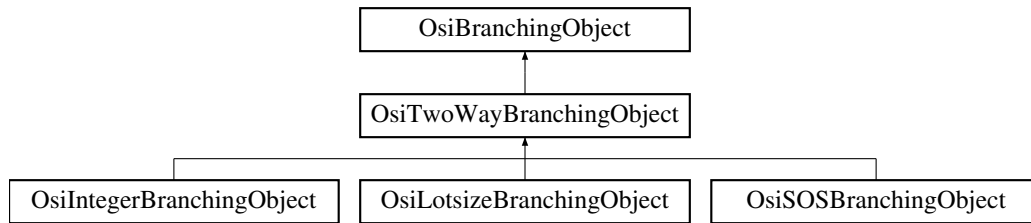
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp](#)

8.8 OsiBranchingObject Class Reference

Abstract branching object base class.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for OsiBranchingObject:



Public Member Functions

- [OsiBranchingObject](#) ()
Default Constructor.
- [OsiBranchingObject](#) ([OsiSolverInterface](#) *solver, double value)
Constructor.
- [OsiBranchingObject](#) (const [OsiBranchingObject](#) &)
Copy constructor.
- [OsiBranchingObject](#) & operator= (const [OsiBranchingObject](#) &rhs)
Assignment operator.
- virtual [OsiBranchingObject](#) * clone () const =0
Clone.
- virtual ~[OsiBranchingObject](#) ()
Destructor.
- int [numberBranches](#) () const
The number of branch arms created for this branching object.
- int [numberBranchesLeft](#) () const
The number of branch arms left for this branching object.
- void [incrementNumberBranchesLeft](#) ()
Increment the number of branch arms left for this branching object.
- void [setNumberBranchesLeft](#) (int)
Set the number of branch arms left for this branching object Just for forcing.
- void [decrementNumberBranchesLeft](#) ()
Decrement the number of branch arms left for this branching object.
- virtual double [branch](#) ([OsiSolverInterface](#) *solver)=0
Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.
- virtual double [branch](#) ()
Execute the actions required to branch, as specified by the current state of the branching object, and advance the object's state.
- virtual bool [boundBranch](#) () const
Return true if branch should fix variables.
- int [branchIndex](#) () const

- *Get the state of the branching object This is just the branch index.*
- void `setBranchingIndex` (int `branchIndex`)
 - *Set the state of the branching object.*
- double `value` () const
 - *Current value.*
- const `OsiObject` * `originalObject` () const
 - *Return pointer back to object which created.*
- void `setOriginalObject` (const `OsiObject` *object)
 - *Set pointer back to object which created.*
- virtual void `checkIsCutoff` (double)
 - *Double checks in case node can change its mind! Returns objective value Can change objective etc.*
- int `columnNumber` () const
 - *For debug.*
- virtual void `print` (const `OsiSolverInterface` *=`NULL`) const
 - *Print something about branch - only if log level high.*

Protected Attributes

- double `value_`
 - *Current value - has some meaning about branch.*
- const `OsiObject` * `originalObject_`
 - *Pointer back to object which created.*
- int `numberBranches_`
 - *Number of branches.*
- short `branchIndex_`
 - *The state of the branching object.*

8.8.1 Detailed Description

Abstract branching object base class.

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

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

See `OsiObject` for an overview of the two classes (`OsiObject` and `OsiBranchingObject`) which make up Osi's branching model.

Definition at line 254 of file `OsiBranchingObject.hpp`.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 `OsiBranchingObject::OsiBranchingObject ()`

Default Constructor.

8.8.2.2 `OsiBranchingObject::OsiBranchingObject (OsiSolverInterface * solver, double value)`

Constructor.

8.8.2.3 OsiBranchingObject::OsiBranchingObject (const OsiBranchingObject &)

Copy constructor.

8.8.2.4 virtual OsiBranchingObject::~~OsiBranchingObject () [virtual]

Destructor.

8.8.3 Member Function Documentation**8.8.3.1 OsiBranchingObject& OsiBranchingObject::operator= (const OsiBranchingObject & rhs)**

Assignment operator.

8.8.3.2 virtual OsiBranchingObject* OsiBranchingObject::clone () const [pure virtual]

Clone.

Implemented in [OsiLotsizeBranchingObject](#), [OsiSOSBranchingObject](#), and [OsiIntegerBranchingObject](#).

8.8.3.3 int OsiBranchingObject::numberBranches () const [inline]

The number of branch arms created for this branching object.

Definition at line 277 of file [OsiBranchingObject.hpp](#).

8.8.3.4 int OsiBranchingObject::numberBranchesLeft () const [inline]

The number of branch arms left for this branching object.

Definition at line 281 of file [OsiBranchingObject.hpp](#).

8.8.3.5 void OsiBranchingObject::incrementNumberBranchesLeft () [inline]

Increment the number of branch arms left for this branching object.

Definition at line 285 of file [OsiBranchingObject.hpp](#).

8.8.3.6 void OsiBranchingObject::setNumberBranchesLeft (int) [inline]

Set the number of branch arms left for this branching object Just for forcing.

Definition at line 291 of file [OsiBranchingObject.hpp](#).

8.8.3.7 void OsiBranchingObject::decrementNumberBranchesLeft () [inline]

Decrement the number of branch arms left for this branching object.

Definition at line 295 of file [OsiBranchingObject.hpp](#).

8.8.3.8 virtual double OsiBranchingObject::branch (OsiSolverInterface * solver) [pure virtual]

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

Returns change in guessed objective on next branch

Implemented in [OsiLotsizeBranchingObject](#), [OsiSOSBranchingObject](#), [OsiIntegerBranchingObject](#), and [OsiTwoWayBranchingObject](#).

8.8.3.9 `virtual double OsiBranchingObject::branch () [inline],[virtual]`

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

Returns change in guessed objective on next branch

Definition at line 309 of file OsiBranchingObject.hpp.

8.8.3.10 `virtual bool OsiBranchingObject::boundBranch () const [inline],[virtual]`

Return true if branch should fix variables.

Definition at line 312 of file OsiBranchingObject.hpp.

8.8.3.11 `int OsiBranchingObject::branchIndex () const [inline]`

Get the state of the branching object This is just the branch index.

Definition at line 317 of file OsiBranchingObject.hpp.

8.8.3.12 `void OsiBranchingObject::setBranchingIndex (int branchIndex) [inline]`

Set the state of the branching object.

Definition at line 322 of file OsiBranchingObject.hpp.

8.8.3.13 `double OsiBranchingObject::value () const [inline]`

Current value.

Definition at line 326 of file OsiBranchingObject.hpp.

8.8.3.14 `const OsiObject* OsiBranchingObject::originalObject () const [inline]`

Return pointer back to object which created.

Definition at line 330 of file OsiBranchingObject.hpp.

8.8.3.15 `void OsiBranchingObject::setOriginalObject (const OsiObject * object) [inline]`

Set pointer back to object which created.

Definition at line 333 of file OsiBranchingObject.hpp.

8.8.3.16 `virtual void OsiBranchingObject::checkIsCutoff (double) [inline],[virtual]`

Double checks in case node can change its mind! Returns objective value Can change objective etc.

Definition at line 338 of file OsiBranchingObject.hpp.

8.8.3.17 `int OsiBranchingObject::columnNumber () const`

For debug.

8.8.3.18 `virtual void OsiBranchingObject::print (const OsiSolverInterface * =NULL) const [inline],[virtual]`

Print something about branch - only if log level high.

Definition at line 343 of file OsiBranchingObject.hpp.

8.8.4 Member Data Documentation

8.8.4.1 `double OsiBranchingObject::value_` [protected]

Current value - has some meaning about branch.

Definition at line 348 of file OsiBranchingObject.hpp.

8.8.4.2 `const OsiObject* OsiBranchingObject::originalObject_` [protected]

Pointer back to object which created.

Definition at line 351 of file OsiBranchingObject.hpp.

8.8.4.3 `int OsiBranchingObject::numberBranches_` [protected]

Number of branches.

Definition at line 355 of file OsiBranchingObject.hpp.

8.8.4.4 `short OsiBranchingObject::branchIndex_` [protected]

The state of the branching object.

i.e. branch index This starts at 0 when created

Definition at line 360 of file OsiBranchingObject.hpp.

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

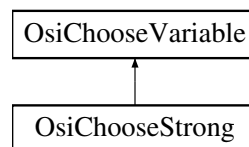
- </home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp>

8.9 OsiChooseStrong Class Reference

This class chooses a variable to branch on.

```
#include <OsiChooseVariable.hpp>
```

Inheritance diagram for OsiChooseStrong:



Public Member Functions

- [OsiChooseStrong](#) ()
Default Constructor.
- [OsiChooseStrong](#) (const [OsiSolverInterface](#) *solver)
Constructor from solver (so we can set up arrays etc)
- [OsiChooseStrong](#) (const [OsiChooseStrong](#) &)
Copy constructor.
- [OsiChooseStrong](#) & `operator=` (const [OsiChooseStrong](#) &rhs)
Assignment operator.

- virtual `OsiChooseVariable * clone ()` const
Clone.
- virtual `~OsiChooseStrong ()`
Destructor.
- virtual int `setupList (OsiBranchingInformation *info, bool initialize)`
Sets up strong list and clears all if initialize is true.
- virtual int `chooseVariable (OsiSolverInterface *solver, OsiBranchingInformation *info, bool fixVariables)`
Choose a variable Returns - -1 Node is infeasible 0 Normal termination - we have a candidate 1 All looks satisfied - no candidate 2 We can change the bound on a variable - but we also have a strong branching candidate 3 We can change the bound on a variable - but we have a non-strong branching candidate 4 We can change the bound on a variable - no other candidates We can pick up branch from `bestObjectIndex()` and `bestWhichWay()` We can pick up a forced branch (can change bound) from `firstForcedObjectIndex()` and `firstForcedWhichWay()` If we have a solution then we can pick up from `goodObjectiveValue()` and `goodSolution()` If fixVariables is true then 2,3,4 are all really same as problem changed.
- int `shadowPriceMode ()` const
Pseudo Shadow Price mode 0 - off 1 - use if no strong info 2 - use if strong not trusted 3 - use even if trusted.
- void `setShadowPriceMode (int value)`
Set Shadow price mode.
- const `OsiPseudoCosts & pseudoCosts ()` const
Accessor method to pseudo cost object.
- `OsiPseudoCosts & pseudoCosts ()`
Accessor method to pseudo cost object.
- int `numberBeforeTrusted ()` const
A few pass-through methods to access members of `pseudoCosts_` as if they were members of `OsiChooseStrong` object.
- void `setNumberBeforeTrusted (int value)`
- int `numberObjects ()` const

Protected Member Functions

- int `doStrongBranching (OsiSolverInterface *solver, OsiBranchingInformation *info, int numberToDo, int return-Criterion)`
This is a utility function which does strong branching on a list of objects and stores the results in `OsiHotInfo.objects`.
- void `resetResults (int num)`
Clear out the results array.

Protected Attributes

- int `shadowPriceMode_`
Pseudo Shadow Price mode 0 - off 1 - use and multiply by strong info 2 - use.
- `OsiPseudoCosts pseudoCosts_`
The pseudo costs for the chooser.
- `OsiHotInfo * results_`
The results of the strong branching done on the candidates where the pseudocosts were not sufficient.
- int `numResults_`
The number of `OsiHotInfo` objects that contain information.

8.9.1 Detailed Description

This class chooses a variable to branch on.

This chooses the variable and direction with reliability strong branching.

The flow is : a) initialize the process. This decides on strong branching list and stores indices of all infeasible objects b) do strong branching on list. If list is empty then just choose one candidate and return without strong branching. If not empty then go through list and return best. However we may find that the node is infeasible or that we can fix a variable. If so we return and it is up to user to call again (after fixing a variable).

Definition at line 318 of file OsiChooseVariable.hpp.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 OsiChooseStrong::OsiChooseStrong ()

Default Constructor.

8.9.2.2 OsiChooseStrong::OsiChooseStrong (const OsiSolverInterface * *solver*)

Constructor from solver (so we can set up arrays etc)

8.9.2.3 OsiChooseStrong::OsiChooseStrong (const OsiChooseStrong &)

Copy constructor.

8.9.2.4 virtual OsiChooseStrong::~~OsiChooseStrong () [virtual]

Destructor.

8.9.3 Member Function Documentation

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

Assignment operator.

8.9.3.2 virtual OsiChooseVariable* OsiChooseStrong::clone () const [virtual]

Clone.

Reimplemented from [OsiChooseVariable](#).

8.9.3.3 virtual int OsiChooseStrong::setupList (OsiBranchingInformation * *info*, bool *initialize*) [virtual]

Sets up strong list and clears all if initialize is true.

Returns number of infeasibilities. If returns -1 then has worked out node is infeasible!

Reimplemented from [OsiChooseVariable](#).

8.9.3.4 virtual int OsiChooseStrong::chooseVariable (OsiSolverInterface * *solver*, OsiBranchingInformation * *info*, bool *fixVariables*) [virtual]

Choose a variable Returns - -1 Node is infeasible 0 Normal termination - we have a candidate 1 All looks satisfied - no candidate 2 We can change the bound on a variable - but we also have a strong branching candidate 3 We can change the bound on a variable - but we have a non-strong branching candidate 4 We can change the bound on a variable - no other candidates We can pick up branch from [bestObjectIndex\(\)](#) and [bestWhichWay\(\)](#) We can pick up a forced branch

(can change bound) from [firstForcedObjectIndex\(\)](#) and [firstForcedWhichWay\(\)](#) If we have a solution then we can pick up from [goodObjectiveValue\(\)](#) and [goodSolution\(\)](#) If fixVariables is true then 2,3,4 are all really same as problem changed.

Reimplemented from [OsiChooseVariable](#).

8.9.3.5 `int OsiChooseStrong::shadowPriceMode () const [inline]`

Pseudo Shadow Price mode 0 - off 1 - use if no strong info 2 - use if strong not trusted 3 - use even if trusted.

Definition at line 366 of file [OsiChooseVariable.hpp](#).

8.9.3.6 `void OsiChooseStrong::setShadowPriceMode (int value) [inline]`

Set Shadow price mode.

Definition at line 369 of file [OsiChooseVariable.hpp](#).

8.9.3.7 `const OsiPseudoCosts& OsiChooseStrong::pseudoCosts () const [inline]`

Accessor method to pseudo cost object.

Definition at line 373 of file [OsiChooseVariable.hpp](#).

8.9.3.8 `OsiPseudoCosts& OsiChooseStrong::pseudoCosts () [inline]`

Accessor method to pseudo cost object.

Definition at line 377 of file [OsiChooseVariable.hpp](#).

8.9.3.9 `int OsiChooseStrong::numberBeforeTrusted () const [inline]`

A feww pass-through methods to access members of `pseudoCosts_` as if they were members of [OsiChooseStrong](#) object.

Definition at line 382 of file [OsiChooseVariable.hpp](#).

8.9.3.10 `void OsiChooseStrong::setNumberBeforeTrusted (int value) [inline]`

Definition at line 384 of file [OsiChooseVariable.hpp](#).

8.9.3.11 `int OsiChooseStrong::numberObjects () const [inline]`

Definition at line 386 of file [OsiChooseVariable.hpp](#).

8.9.3.12 `int OsiChooseStrong::doStrongBranching (OsiSolverInterface * solver, OsiBranchingInformation * info, int numberToDo, int returnCriterion) [protected]`

This is a utility function which does strong branching on a list of objects and stores the results in [OsiHotInfo](#).objects.

On entry the object sequence is stored in the [OsiHotInfo](#) object and maybe more. It returns - -1 - one branch was infeasible both ways 0 - all inspected - nothing can be fixed 1 - all inspected - some can be fixed (returnCriterion==0) 2 - may be returning early - one can be fixed (last one done) (returnCriterion==1) 3 - returning because max time

8.9.3.13 `void OsiChooseStrong::resetResults (int num) [protected]`

Clear out the results array.

8.9.4 Member Data Documentation

8.9.4.1 int OsiChooseStrong::shadowPriceMode_ [protected]

Pseudo Shadow Price mode 0 - off 1 - use and multiply by strong info 2 - use.

Definition at line 416 of file OsiChooseVariable.hpp.

8.9.4.2 OsiPseudoCosts OsiChooseStrong::pseudoCosts_ [protected]

The pseudo costs for the chooser.

Definition at line 419 of file OsiChooseVariable.hpp.

8.9.4.3 OsiHotInfo* OsiChooseStrong::results_ [protected]

The results of the strong branching done on the candidates where the pseudocosts were not sufficient.

Definition at line 423 of file OsiChooseVariable.hpp.

8.9.4.4 int OsiChooseStrong::numResults_ [protected]

The number of [OsiHotInfo](#) objects that contain information.

Definition at line 425 of file OsiChooseVariable.hpp.

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

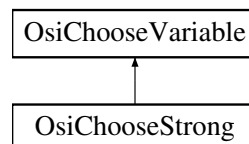
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiChooseVariable.hpp](#)

8.10 OsiChooseVariable Class Reference

This class chooses a variable to branch on.

```
#include <OsiChooseVariable.hpp>
```

Inheritance diagram for OsiChooseVariable:



Public Member Functions

- [OsiChooseVariable](#) ()
Default Constructor.
- [OsiChooseVariable](#) (const [OsiSolverInterface](#) *solver)
Constructor from solver (so we can set up arrays etc)
- [OsiChooseVariable](#) (const [OsiChooseVariable](#) &)
Copy constructor.
- [OsiChooseVariable](#) & [operator=](#) (const [OsiChooseVariable](#) &rhs)
Assignment operator.
- virtual [OsiChooseVariable](#) * [clone](#) () const
Clone.
- virtual [~OsiChooseVariable](#) ()

Destructor.

- virtual int **setupList** (**OsiBranchingInformation** *info, bool initialize)
Sets up strong list and clears all if initialize is true.
- virtual int **chooseVariable** (**OsiSolverInterface** *solver, **OsiBranchingInformation** *info, bool fixVariables)
*Choose a variable Returns - -1 Node is infeasible 0 Normal termination - we have a candidate 1 All looks satisfied - no candidate 2 We can change the bound on a variable - but we also have a strong branching candidate 3 We can change the bound on a variable - but we have a non-strong branching candidate 4 We can change the bound on a variable - no other candidates We can pick up branch from **bestObjectIndex()** and **bestWhichWay()** We can pick up a forced branch (can change bound) from **firstForcedObjectIndex()** and **firstForcedWhichWay()** If we have a solution then we can pick up from **goodObjectiveValue()** and **goodSolution()** If fixVariables is true then 2,3,4 are all really same as problem changed.*
- virtual bool **feasibleSolution** (const **OsiBranchingInformation** *info, const double *solution, int numberObjects, const **OsiObject** **objects)
Returns true if solution looks feasible against given objects.
- void **saveSolution** (const **OsiSolverInterface** *solver)
Saves a good solution.
- void **clearGoodSolution** ()
Clears out good solution after use.
- virtual void **updateInformation** (const **OsiBranchingInformation** *info, int branch, **OsiHotInfo** *hotInfo)
Given a candidate fill in useful information e.g. estimates.
- virtual void **updateInformation** (int whichObject, int branch, double changeInObjective, double changeInValue, int status)
Given a branch fill in useful information e.g. estimates.
- double **goodObjectiveValue** () const
Objective value for feasible solution.
- double **upChange** () const
Estimate of up change or change on chosen if n-way.
- double **downChange** () const
Estimate of down change or max change on other possibilities if n-way.
- const double * **goodSolution** () const
Good solution - deleted by finalize.
- int **bestObjectIndex** () const
Index of chosen object.
- void **setBestObjectIndex** (int value)
Set index of chosen object.
- int **bestWhichWay** () const
Preferred way of chosen object.
- void **setBestWhichWay** (int value)
Set preferred way of chosen object.
- int **firstForcedObjectIndex** () const
Index of forced object.
- void **setFirstForcedObjectIndex** (int value)
Set index of forced object.
- int **firstForcedWhichWay** () const
Preferred way of forced object.
- void **setFirstForcedWhichWay** (int value)
Set preferred way of forced object.
- int **numberUnsatisfied** () const
Get the number of objects unsatisfied at this node - accurate on first pass.

- int [numberStrong](#) () const
Number of objects to choose for strong branching.
- void [setNumberStrong](#) (int value)
Set number of objects to choose for strong branching.
- int [numberOnList](#) () const
Number left on strong list.
- int [numberStrongDone](#) () const
Number of strong branches actually done.
- int [numberStrongIterations](#) () const
Number of strong iterations actually done.
- int [numberStrongFixed](#) () const
Number of strong branches which changed bounds.
- const int * [candidates](#) () const
List of candidates.
- bool [trustStrongForBound](#) () const
Trust results from strong branching for changing bounds.
- void [setTrustStrongForBound](#) (bool yesNo)
Set trust results from strong branching for changing bounds.
- bool [trustStrongForSolution](#) () const
Trust results from strong branching for valid solution.
- void [setTrustStrongForSolution](#) (bool yesNo)
Set trust results from strong branching for valid solution.
- void [setSolver](#) (const [OsiSolverInterface](#) *solver)
Set solver and redo arrays.
- int [status](#) () const
Return status - -1 Node is infeasible 0 Normal termination - we have a candidate 1 All looks satisfied - no candidate 2 We can change the bound on a variable - but we also have a strong branching candidate 3 We can change the bound on a variable - but we have a non-strong branching candidate 4 We can change the bound on a variable - no other candidates We can pick up branch from [bestObjectIndex\(\)](#) and [bestWhichWay\(\)](#) We can pick up a forced branch (can change bound) from [firstForcedObjectIndex\(\)](#) and [firstForcedWhichWay\(\)](#) If we have a solution then we can pick up from [goodObjectiveValue\(\)](#) and [goodSolution\(\)](#)
- void [setStatus](#) (int value)

Protected Attributes

- double [goodObjectiveValue_](#)
Objective value for feasible solution.
- double [upChange_](#)
Estimate of up change or change on chosen if n-way.
- double [downChange_](#)
Estimate of down change or max change on other possibilities if n-way.
- double * [goodSolution_](#)
Good solution - deleted by finalize.
- int * [list_](#)
List of candidates.
- double * [useful_](#)
Useful array (for sorting etc)
- const [OsiSolverInterface](#) * [solver_](#)

- Pointer to solver.*
- int [status_](#)
- int [bestObjectIndex_](#)
Index of chosen object.
- int [bestWhichWay_](#)
Preferred way of chosen object.
- int [firstForcedObjectIndex_](#)
Index of forced object.
- int [firstForcedWhichWay_](#)
Preferred way of forced object.
- int [numberUnsatisfied_](#)
The number of objects unsatisfied at this node.
- int [numberStrong_](#)
Number of objects to choose for strong branching.
- int [numberOnList_](#)
Number left on strong list.
- int [numberStrongDone_](#)
Number of strong branches actually done.
- int [numberStrongIterations_](#)
Number of strong iterations actually done.
- int [numberStrongFixed_](#)
Number of bound changes due to strong branching.
- bool [trustStrongForBound_](#)
List of unsatisfied objects - first numberOnList_ for strong branching Trust results from strong branching for changing bounds.
- bool [trustStrongForSolution_](#)
Trust results from strong branching for valid solution.

8.10.1 Detailed Description

This class chooses a variable to branch on.

The base class just chooses the variable and direction without strong branching but it has information which would normally be used by strong branching e.g. to re-enter having fixed a variable but using same candidates for strong branching.

The flow is : a) initialize the process. This decides on strong branching list and stores indices of all infeasible objects b) do strong branching on list. If list is empty then just choose one candidate and return without strong branching. If not empty then go through list and return best. However we may find that the node is infeasible or that we can fix a variable. If so we return and it is up to user to call again (after fixing a variable).

Definition at line 33 of file `OsiChooseVariable.hpp`.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 `OsiChooseVariable::OsiChooseVariable ()`

Default Constructor.

8.10.2.2 `OsiChooseVariable::OsiChooseVariable (const OsiSolverInterface * solver)`

Constructor from solver (so we can set up arrays etc)

8.10.2.3 OsiChooseVariable::OsiChooseVariable (const OsiChooseVariable &)

Copy constructor.

8.10.2.4 virtual OsiChooseVariable::~~OsiChooseVariable () [virtual]

Destructor.

8.10.3 Member Function Documentation

8.10.3.1 OsiChooseVariable& OsiChooseVariable::operator= (const OsiChooseVariable & rhs)

Assignment operator.

8.10.3.2 virtual OsiChooseVariable* OsiChooseVariable::clone () const [virtual]

Clone.

Reimplemented in [OsiChooseStrong](#).

8.10.3.3 virtual int OsiChooseVariable::setupList (OsiBranchingInformation * info, bool initialize) [virtual]

Sets up strong list and clears all if initialize is true.

Returns number of infeasibilities. If returns -1 then has worked out node is infeasible!

Reimplemented in [OsiChooseStrong](#).

8.10.3.4 virtual int OsiChooseVariable::chooseVariable (OsiSolverInterface * solver, OsiBranchingInformation * info, bool fixVariables) [virtual]

Choose a variable Returns - -1 Node is infeasible 0 Normal termination - we have a candidate 1 All looks satisfied - no candidate 2 We can change the bound on a variable - but we also have a strong branching candidate 3 We can change the bound on a variable - but we have a non-strong branching candidate 4 We can change the bound on a variable - no other candidates We can pick up branch from [bestObjectIndex\(\)](#) and [bestWhichWay\(\)](#) We can pick up a forced branch (can change bound) from [firstForcedObjectIndex\(\)](#) and [firstForcedWhichWay\(\)](#) If we have a solution then we can pick up from [goodObjectiveValue\(\)](#) and [goodSolution\(\)](#) If fixVariables is true then 2,3,4 are all really same as problem changed.

Reimplemented in [OsiChooseStrong](#).

8.10.3.5 virtual bool OsiChooseVariable::feasibleSolution (const OsiBranchingInformation * info, const double * solution, int numberObjects, const OsiObject ** objects) [virtual]

Returns true if solution looks feasible against given objects.

8.10.3.6 void OsiChooseVariable::saveSolution (const OsiSolverInterface * solver)

Saves a good solution.

8.10.3.7 void OsiChooseVariable::clearGoodSolution ()

Clears out good solution after use.

8.10.3.8 virtual void OsiChooseVariable::updateInformation (const OsiBranchingInformation * info, int branch, OsiHotInfo * hotInfo) [virtual]

Given a candidate fill in useful information e.g. estimates.

8.10.3.9 `virtual void OsiChooseVariable::updateInformation (int whichObject, int branch, double changeInObjective, double changeInValue, int status)` `[virtual]`

Given a branch fill in useful information e.g. estimates.

8.10.3.10 `double OsiChooseVariable::goodObjectiveValue () const` `[inline]`

Objective value for feasible solution.

Definition at line 93 of file `OsiChooseVariable.hpp`.

8.10.3.11 `double OsiChooseVariable::upChange () const` `[inline]`

Estimate of up change or change on chosen if n-way.

Definition at line 96 of file `OsiChooseVariable.hpp`.

8.10.3.12 `double OsiChooseVariable::downChange () const` `[inline]`

Estimate of down change or max change on other possibilities if n-way.

Definition at line 99 of file `OsiChooseVariable.hpp`.

8.10.3.13 `const double* OsiChooseVariable::goodSolution () const` `[inline]`

Good solution - deleted by finalize.

Definition at line 102 of file `OsiChooseVariable.hpp`.

8.10.3.14 `int OsiChooseVariable::bestObjectIndex () const` `[inline]`

Index of chosen object.

Definition at line 105 of file `OsiChooseVariable.hpp`.

8.10.3.15 `void OsiChooseVariable::setBestObjectIndex (int value)` `[inline]`

Set index of chosen object.

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

8.10.3.16 `int OsiChooseVariable::bestWhichWay () const` `[inline]`

Preferred way of chosen object.

Definition at line 111 of file `OsiChooseVariable.hpp`.

8.10.3.17 `void OsiChooseVariable::setBestWhichWay (int value)` `[inline]`

Set preferred way of chosen object.

Definition at line 114 of file `OsiChooseVariable.hpp`.

8.10.3.18 `int OsiChooseVariable::firstForcedObjectIndex () const` `[inline]`

Index of forced object.

Definition at line 117 of file `OsiChooseVariable.hpp`.

8.10.3.19 `void OsiChooseVariable::setFirstForcedObjectIndex (int value)` `[inline]`

Set index of forced object.

Definition at line 120 of file OsiChooseVariable.hpp.

8.10.3.20 `int OsiChooseVariable::firstForcedWhichWay () const [inline]`

Preferred way of forced object.

Definition at line 123 of file OsiChooseVariable.hpp.

8.10.3.21 `void OsiChooseVariable::setFirstForcedWhichWay (int value) [inline]`

Set preferred way of forced object.

Definition at line 126 of file OsiChooseVariable.hpp.

8.10.3.22 `int OsiChooseVariable::numberUnsatisfied () const [inline]`

Get the number of objects unsatisfied at this node - accurate on first pass.

Definition at line 129 of file OsiChooseVariable.hpp.

8.10.3.23 `int OsiChooseVariable::numberStrong () const [inline]`

Number of objects to choose for strong branching.

Definition at line 132 of file OsiChooseVariable.hpp.

8.10.3.24 `void OsiChooseVariable::setNumberStrong (int value) [inline]`

Set number of objects to choose for strong branching.

Definition at line 135 of file OsiChooseVariable.hpp.

8.10.3.25 `int OsiChooseVariable::numberOnList () const [inline]`

Number left on strong list.

Definition at line 138 of file OsiChooseVariable.hpp.

8.10.3.26 `int OsiChooseVariable::numberStrongDone () const [inline]`

Number of strong branches actually done.

Definition at line 141 of file OsiChooseVariable.hpp.

8.10.3.27 `int OsiChooseVariable::numberStrongIterations () const [inline]`

Number of strong iterations actually done.

Definition at line 144 of file OsiChooseVariable.hpp.

8.10.3.28 `int OsiChooseVariable::numberStrongFixed () const [inline]`

Number of strong branches which changed bounds.

Definition at line 147 of file OsiChooseVariable.hpp.

8.10.3.29 `const int* OsiChooseVariable::candidates () const [inline]`

List of candidates.

Definition at line 150 of file OsiChooseVariable.hpp.

8.10.3.30 `bool OsiChooseVariable::trustStrongForBound () const` `[inline]`

Trust results from strong branching for changing bounds.

Definition at line 153 of file OsiChooseVariable.hpp.

8.10.3.31 `void OsiChooseVariable::setTrustStrongForBound (bool yesNo)` `[inline]`

Set trust results from strong branching for changing bounds.

Definition at line 156 of file OsiChooseVariable.hpp.

8.10.3.32 `bool OsiChooseVariable::trustStrongForSolution () const` `[inline]`

Trust results from strong branching for valid solution.

Definition at line 159 of file OsiChooseVariable.hpp.

8.10.3.33 `void OsiChooseVariable::setTrustStrongForSolution (bool yesNo)` `[inline]`

Set trust results from strong branching for valid solution.

Definition at line 162 of file OsiChooseVariable.hpp.

8.10.3.34 `void OsiChooseVariable::setSolver (const OsiSolverInterface * solver)`

Set solver and redo arrays.

8.10.3.35 `int OsiChooseVariable::status () const` `[inline]`

Return status - -1 Node is infeasible 0 Normal termination - we have a candidate 1 All looks satisfied - no candidate 2 We can change the bound on a variable - but we also have a strong branching candidate 3 We can change the bound on a variable - but we have a non-strong branching candidate 4 We can change the bound on a variable - no other candidates We can pick up branch from [bestObjectIndex\(\)](#) and [bestWhichWay\(\)](#) We can pick up a forced branch (can change bound) from [firstForcedObjectIndex\(\)](#) and [firstForcedWhichWay\(\)](#) If we have a solution then we can pick up from [goodObjectiveValue\(\)](#) and [goodSolution\(\)](#)

Definition at line 177 of file OsiChooseVariable.hpp.

8.10.3.36 `void OsiChooseVariable::setStatus (int value)` `[inline]`

Definition at line 179 of file OsiChooseVariable.hpp.

8.10.4 Member Data Documentation

8.10.4.1 `double OsiChooseVariable::goodObjectiveValue_` `[protected]`

Objective value for feasible solution.

Definition at line 186 of file OsiChooseVariable.hpp.

8.10.4.2 `double OsiChooseVariable::upChange_` `[protected]`

Estimate of up change or change on chosen if n-way.

Definition at line 188 of file OsiChooseVariable.hpp.

8.10.4.3 `double OsiChooseVariable::downChange_` [protected]

Estimate of down change or max change on other possibilities if n-way.

Definition at line 190 of file OsiChooseVariable.hpp.

8.10.4.4 `double* OsiChooseVariable::goodSolution_` [protected]

Good solution - deleted by finalize.

Definition at line 192 of file OsiChooseVariable.hpp.

8.10.4.5 `int* OsiChooseVariable::list_` [protected]

List of candidates.

Definition at line 194 of file OsiChooseVariable.hpp.

8.10.4.6 `double* OsiChooseVariable::useful_` [protected]

Useful array (for sorting etc)

Definition at line 196 of file OsiChooseVariable.hpp.

8.10.4.7 `const OsiSolverInterface* OsiChooseVariable::solver_` [protected]

Pointer to solver.

Definition at line 198 of file OsiChooseVariable.hpp.

8.10.4.8 `int OsiChooseVariable::status_` [protected]

Definition at line 207 of file OsiChooseVariable.hpp.

8.10.4.9 `int OsiChooseVariable::bestObjectIndex_` [protected]

Index of chosen object.

Definition at line 209 of file OsiChooseVariable.hpp.

8.10.4.10 `int OsiChooseVariable::bestWhichWay_` [protected]

Preferred way of chosen object.

Definition at line 211 of file OsiChooseVariable.hpp.

8.10.4.11 `int OsiChooseVariable::firstForcedObjectIndex_` [protected]

Index of forced object.

Definition at line 213 of file OsiChooseVariable.hpp.

8.10.4.12 `int OsiChooseVariable::firstForcedWhichWay_` [protected]

Preferred way of forced object.

Definition at line 215 of file OsiChooseVariable.hpp.

8.10.4.13 `int OsiChooseVariable::numberUnsatisfied_` [protected]

The number of objects unsatisfied at this node.

Definition at line 217 of file OsiChooseVariable.hpp.

8.10.4.14 `int OsiChooseVariable::numberStrong_` `[protected]`

Number of objects to choose for strong branching.

Definition at line 219 of file OsiChooseVariable.hpp.

8.10.4.15 `int OsiChooseVariable::numberOnList_` `[protected]`

Number left on strong list.

Definition at line 221 of file OsiChooseVariable.hpp.

8.10.4.16 `int OsiChooseVariable::numberStrongDone_` `[protected]`

Number of strong branches actually done.

Definition at line 223 of file OsiChooseVariable.hpp.

8.10.4.17 `int OsiChooseVariable::numberStrongIterations_` `[protected]`

Number of strong iterations actually done.

Definition at line 225 of file OsiChooseVariable.hpp.

8.10.4.18 `int OsiChooseVariable::numberStrongFixed_` `[protected]`

Number of bound changes due to strong branching.

Definition at line 227 of file OsiChooseVariable.hpp.

8.10.4.19 `bool OsiChooseVariable::trustStrongForBound_` `[protected]`

List of unsatisfied objects - first numberOnList_ for strong branching Trust results from strong branching for changing bounds.

Definition at line 230 of file OsiChooseVariable.hpp.

8.10.4.20 `bool OsiChooseVariable::trustStrongForSolution_` `[protected]`

Trust results from strong branching for valid solution.

Definition at line 232 of file OsiChooseVariable.hpp.

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

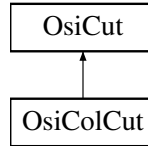
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiChooseVariable.hpp](#)

8.11 OsiColCut Class Reference

Column Cut Class.

```
#include <OsiColCut.hpp>
```

Inheritance diagram for OsiColCut:



Public Member Functions

Setting column bounds

- void [setLbs](#) (int nElements, const int *colIndices, const double *lbElements)
Set column lower bounds.
- void [setLbs](#) (const CoinPackedVector &lbs)
Set column lower bounds from a packed vector.
- void [setUbs](#) (int nElements, const int *colIndices, const double *ubElements)
Set column upper bounds.
- void [setUbs](#) (const CoinPackedVector &ubs)
Set column upper bounds from a packed vector.

Getting column bounds

- const CoinPackedVector & [lbs](#) () const
Get column lower bounds.
- const CoinPackedVector & [ubs](#) () const
Get column upper bounds.

Comparison operators

- virtual bool [operator==](#) (const [OsiColCut](#) &rhs) const
equal - true if lower bounds, upper bounds, and [OsiCut](#) are equal.
- virtual bool [operator!=](#) (const [OsiColCut](#) &rhs) const
not equal

Sanity checks on cut

- virtual bool [consistent](#) () const
Returns true if the cut is consistent with respect to itself.
- virtual bool [consistent](#) (const [OsiSolverInterface](#) &im) const
Returns true if cut is consistent with respect to the solver interface's model.
- virtual bool [infeasible](#) (const [OsiSolverInterface](#) &im) const
Returns true if the cut is infeasible with respect to its bounds and the column bounds in the solver interface's models.
- virtual double [violated](#) (const double *solution) const
Returns infeasibility of the cut with respect to solution passed in i.e.

Constructors and destructors

- [OsiColCut](#) & [operator=](#) (const [OsiColCut](#) &rhs)
Assignment operator.
- [OsiColCut](#) (const [OsiColCut](#) &)
Copy constructor.

- `OsiColCut ()`
Default Constructor.
- virtual `OsiColCut * clone () const`
Clone.
- virtual `~OsiColCut ()`
Destructor.

Debug stuff

- virtual void `print () const`
Print cuts in collection.

Private Attributes

Private member data

- `CoinPackedVector lbs_`
Lower bounds.
- `CoinPackedVector ubs_`
Upper bounds.

Friends

- void `OsiColCutUnitTest (const OsiSolverInterface *baseSiP, const std::string &mpsDir)`
A function that tests the methods in the `OsiColCut` class.

Additional Inherited Members

8.11.1 Detailed Description

Column Cut Class.

Column Cut Class has:

- a sparse vector of column lower bounds
- a sparse vector of column upper bounds

Definition at line 23 of file `OsiColCut.hpp`.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 `OsiColCut::OsiColCut (const OsiColCut &)`

Copy constructor.

8.11.2.2 `OsiColCut::OsiColCut ()`

Default Constructor.

8.11.2.3 `virtual OsiColCut::~OsiColCut () [virtual]`

Destructor.

8.11.3 Member Function Documentation

8.11.3.1 `void OsiColCut::setLbs (int nElements, const int * colIndices, const double * lbElements)` `[inline]`

Set column lower bounds.

Definition at line 161 of file OsiColCut.hpp.

8.11.3.2 `void OsiColCut::setLbs (const CoinPackedVector & lbs)` `[inline]`

Set column lower bounds from a packed vector.

Definition at line 177 of file OsiColCut.hpp.

8.11.3.3 `void OsiColCut::setUbs (int nElements, const int * colIndices, const double * ubElements)` `[inline]`

Set column upper bounds.

Definition at line 169 of file OsiColCut.hpp.

8.11.3.4 `void OsiColCut::setUbs (const CoinPackedVector & ubs)` `[inline]`

Set column upper bounds from a packed vector.

Definition at line 182 of file OsiColCut.hpp.

8.11.3.5 `const CoinPackedVector & OsiColCut::lbs () const` `[inline]`

Get column lower bounds.

Definition at line 190 of file OsiColCut.hpp.

8.11.3.6 `const CoinPackedVector & OsiColCut::ubs () const` `[inline]`

Get column upper bounds.

Definition at line 195 of file OsiColCut.hpp.

8.11.3.7 `bool OsiColCut::operator== (const OsiColCut & rhs) const` `[inline],[virtual]`

equal - true if lower bounds, upper bounds,

and [OsiCut](#) are equal.

Definition at line 204 of file OsiColCut.hpp.

8.11.3.8 `bool OsiColCut::operator!= (const OsiColCut & rhs) const` `[inline],[virtual]`

not equal

Definition at line 217 of file OsiColCut.hpp.

8.11.3.9 `bool OsiColCut::consistent () const` `[inline],[virtual]`

Returns true if the cut is consistent with respect to itself.

This checks to ensure that:

- The bound vectors do not have duplicate indices,
- The bound vectors indices are ≥ 0

Implements [OsiCut](#).

Definition at line 226 of file OsiColCut.hpp.

8.11.3.10 `bool OsiColCut::consistent (const OsiSolverInterface & im) const` `[inline],[virtual]`

Returns true if cut is consistent with respect to the solver

interface's model.

This checks to ensure that the lower & upperbound packed vectors:

- do not have an index \geq the number of column in the model.

Implements [OsiCut](#).

Definition at line 239 of file OsiColCut.hpp.

8.11.3.11 `bool OsiColCut::infeasible (const OsiSolverInterface & im) const` `[inline],[virtual]`

Returns true if the cut is infeasible with respect to its bounds and the

column bounds in the solver interface's models.

This checks whether:

- the maximum of the new and existing lower bounds is strictly greater than the minimum of the new and existing upper bounds.

Implements [OsiCut](#).

Definition at line 290 of file OsiColCut.hpp.

8.11.3.12 `virtual double OsiColCut::violated (const double * solution) const` `[virtual]`

Returns infeasibility of the cut with respect to solution passed in i.e.

is positive if cuts off that solution. solution is getNumCols() long..

Implements [OsiCut](#).

8.11.3.13 `OsiColCut& OsiColCut::operator= (const OsiColCut & rhs)`

Assignment operator.

8.11.3.14 `virtual OsiColCut* OsiColCut::clone () const` `[virtual]`

Clone.

8.11.3.15 `virtual void OsiColCut::print () const` `[virtual]`

Print cuts in collection.

Reimplemented from [OsiCut](#).

8.11.4 Friends And Related Function Documentation

8.11.4.1 `void OsiColCutUnitTest (const OsiSolverInterface * baseSiP, const std::string & mpsDir)` `[friend]`

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

8.11.5 Member Data Documentation

8.11.5.1 CoinPackedVector OsiColCut::lbs_ [private]

Lower bounds.

Definition at line 149 of file OsiColCut.hpp.

8.11.5.2 CoinPackedVector OsiColCut::ubs_ [private]

Upper bounds.

Definition at line 151 of file OsiColCut.hpp.

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

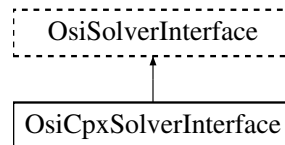
- </home/ted/COIN/trunk/Osi/src/Osi/OsiColCut.hpp>

8.12 OsiCpxSolverInterface Class Reference

CPLEX Solver Interface.

```
#include <OsiCpxSolverInterface.hpp>
```

Inheritance diagram for OsiCpxSolverInterface:



Public Member Functions

- virtual void [setObjSense](#) (double s)
Set objective function sense (1 for min (default), -1 for max,)
- virtual void [setColSolution](#) (const double *colsol)
Set the primal solution column values.
- virtual void [setRowPrice](#) (const double *rowprice)
Set dual solution vector.
- const char * [getCtype](#) () const
return a vector of variable types (continous, binary, integer)

Solve methods

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

Parameter set/get methods

The set methods return true if the parameter was set to the given value, false otherwise.

There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the volume algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.

The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.

- bool `setIntParam` (`OsiIntParam` key, int value)
Set an integer parameter.
- bool `setDbiParam` (`OsiDbiParam` key, double value)
Set a double parameter.
- bool `setStrParam` (`OsiStrParam` key, const std::string &value)
Set a string parameter.
- bool `getIntParam` (`OsiIntParam` key, int &value) const
Get an integer parameter.
- bool `getDbiParam` (`OsiDbiParam` key, double &value) const
Get a double parameter.
- bool `getStrParam` (`OsiStrParam` key, std::string &value) const
Get a string parameter.
- void `setMipStart` (bool value)
- bool `getMipStart` () const

Methods returning info on how the solution process terminated

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

WarmStart related methods

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

Hotstart related methods (primarily used in strong branching).

The user can create a hotstart (a snapshot) of the optimization process then reoptimize over and over again always starting from there.

NOTE: between hotstarted optimizations only bound changes are allowed.

- virtual void [markHotStart](#) ()
Create a hotstart point of the optimization process.
- virtual void [solveFromHotStart](#) ()
Optimize starting from the hotstart.
- virtual void [unmarkHotStart](#) ()
Delete the snapshot.

Methods related to querying the input data

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

Methods related to querying the solution

- virtual const double * [getColSolution](#) () const
Get pointer to array[getNumCols()] of primal solution vector.
- virtual const double * [getRowPrice](#) () const
Get pointer to array[getNumRows()] of dual prices.
- virtual const double * [getReducedCost](#) () const
Get a pointer to array[getNumCols()] of reduced costs.
- virtual const double * [getRowActivity](#) () const
Get pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).
- virtual double [getObjValue](#) () const
Get objective function value.

- virtual int [getIterationCount](#) () const
Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver).
- virtual std::vector< double * > [getDualRays](#) (int maxNumRays, bool fullRay=false) const
Get as many dual rays as the solver can provide.
- virtual std::vector< double * > [getPrimalRays](#) (int maxNumRays) const
Get as many primal rays as the solver can provide.

Changing bounds on variables and constraints

- virtual void [setObjCoeff](#) (int elementIndex, double elementValue)
Set an objective function coefficient.
- virtual void [setObjCoeffSet](#) (const int *indexFirst, const int *indexLast, const double *coeffList)
Set a a set of objective function coefficients.
- virtual void [setColLower](#) (int elementIndex, double elementValue)
Set a single column lower bound
Use -COIN_DBL_MAX for -infinity.
- virtual void [setColUpper](#) (int elementIndex, double elementValue)
Set a single column upper bound
Use COIN_DBL_MAX for infinity.
- virtual void [setColBounds](#) (int elementIndex, double lower, double upper)
Set a single column lower and upper bound
The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#)
- virtual void [setColSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a number of columns simultaneously
The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#) over and over again.
- virtual void [setRowLower](#) (int elementIndex, double elementValue)
Set a single row lower bound
Use -COIN_DBL_MAX for -infinity.
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)
Set a single row upper bound
Use COIN_DBL_MAX for infinity.
- virtual void [setRowBounds](#) (int elementIndex, double lower, double upper)
Set a single row lower and upper bound
The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#)
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)
Set the type of a single row
- virtual void [setRowSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a number of rows simultaneously
The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#) over and over again.
- virtual void [setRowSetTypes](#) (const int *indexFirst, const int *indexLast, const char *senseList, const double *rhsList, const double *rangeList)
Set the type of a number of rows simultaneously
The default implementation just invokes [setRowType\(\)](#) and over and over again.

Integrality related changing methods

- virtual void [setContinuous](#) (int index)
Set the index-th variable to be a continuous variable.
- virtual void [setInteger](#) (int index)
Set the index-th variable to be an integer variable.
- virtual void [setContinuous](#) (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be continuous variables.
- virtual void [setInteger](#) (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be integer variables.

**Methods to expand a problem.
**

Note that if a column is added then by default it will correspond to a continuous variable.

- virtual void [addCol](#) (const CoinPackedVectorBase &vec, const double collb, const double colub, const double obj)
Add a column (primal variable) to the problem.
- virtual void [addCols](#) (const int numcols, const CoinPackedVectorBase *const *cols, const double *collb, const double *colub, const double *obj)
Add a set of columns (primal variables) 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)
Add a row (constraint) to the problem.
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase *const *rows, const double *rowlb, const double *rowub)
Add a set of rows (constraints) to the problem.
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase *const *rows, const char *rowsen, const double *rowrhs, const double *rowrng)
Add a set of rows (constraints) to the problem.
- virtual void [deleteRows](#) (const int num, const int *rowIndices)
Delete a set of rows (constraints) from the problem.

Methods to input a problem

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

Message handling

- void `passInMessageHandler` (CoinMessageHandler *handler)
Pass in a message handler It is the client's responsibility to destroy a message handler installed by this routine; it will not be destroyed when the solver interface is destroyed.

Constructors and destructor

- `OsiCpxSolverInterface` ()
Default Constructor.
- virtual `OsiSolverInterface` * `clone` (bool copyData=true) const
Clone.
- `OsiCpxSolverInterface` (const `OsiCpxSolverInterface` &)
Copy constructor.
- `OsiCpxSolverInterface` & `operator=` (const `OsiCpxSolverInterface` &rhs)
Assignment operator.
- virtual `~OsiCpxSolverInterface` ()
Destructor.
- virtual void `reset` ()
Resets as if default constructor.

OsiSimplexInterface methods

Cplex adds a slack with coeff +1 in "<=" and "=" constraints, with coeff -1 in ">=", slack being non negative.

We switch in order to get a "Cp tableau" where all the slacks have coefficient +1 in the original tableau.

If a slack for ">=" is non basic, invB is not changed; column of the slack in the optimal tableau is flipped.

If a slack for ">=" is basic, corresp. row of invB is flipped; whole row of the optimal tableau is flipped; then whole column for the slack in opt tableau is flipped.

Ranged rows are not supported. It might work, but no guarantee is given.

Code implemented only for Cplex9.0 and higher, lower version number of Cplex will abort the code.

- virtual int `canDoSimplexInterface` () const
Returns 1 if can just do getBInv etc 2 if has all OsiSimplex methods and 0 if it has none.
- virtual void `enableSimplexInterface` (int doingPrimal)
Useless function, defined only for compatibility with OsiSimplexInterface.
- virtual void `disableSimplexInterface` ()
Useless function, defined only for compatibility with OsiSimplexInterface.
- virtual void `enableFactorization` () const
Useless function, defined only for compatibility with OsiSimplexInterface.
- virtual void `disableFactorization` () const
Useless function, defined only for compatibility with OsiSimplexInterface.
- virtual bool `basisIsAvailable` () const
Returns true if a basis is available.
- virtual void `getBasisStatus` (int *cstat, int *rstat) const
Returns a basis status of the structural/artificial variables At present as warm start i.e 0: free, 1: basic, 2: upper, 3: lower.
- virtual void `getBInvARow` (int row, double *z, double *slack=NULL) const
Get a row of the tableau (slack part in slack if not NULL)
- virtual void `getBInvRow` (int row, double *z) const
Get a row of the basis inverse.
- virtual void `getBInvACol` (int col, double *vec) const
Get a column of the tableau.
- virtual void `getBInvCol` (int col, double *vec) const
Get a column of the basis inverse.

- virtual void [getBasics](#) (int *index) const
Get indices of the pivot variable in each row (order of indices corresponds to the order of elements in a vector returned by [getBlvACol\(\)](#) and [getBlvCol\(\)](#)).
- void [switchToLP](#) ()
switches CPLEX to prob type LP
- void [switchToMIP](#) ()
switches CPLEX to prob type MIP

Protected Member Functions

Protected methods

- virtual void [applyRowCut](#) (const [OsiRowCut](#) &rc)
Apply a row cut. Return true if cut was applied.
- virtual void [applyColCut](#) (const [OsiColCut](#) &cc)
Apply a column cut (bound adjustment).

Private Member Functions

Private static class functions

- void [resizeColType](#) (int minsize)
resizes coltype_ vector to be able to store at least minsize elements
- void [freeColType](#) ()
frees colsize_ vector

Private methods

- [CPXLPptr](#) [getMutableLpPtr](#) () const
Get LP Pointer for const methods.
- void [gutsOfCopy](#) (const [OsiCpxSolverInterface](#) &source)
The real work of a copy constructor (used by copy and assignment)
- void [gutsOfConstructor](#) ()
The real work of the constructor.
- void [gutsOfDestructor](#) ()
The real work of the destructor.
- void [freeCachedColRim](#) ()
free cached column rim vectors
- void [freeCachedRowRim](#) ()
free cached row rim vectors
- void [freeCachedResults](#) ()
free cached result vectors
- void [freeCachedMatrix](#) ()
free cached matrices
- void [freeCachedData](#) (int keepCached=[KEEPCACHED_NONE](#))
free all cached data (except specified entries, see [getLpPtr\(\)](#))
- void [freeAllMemory](#) ()
free all allocated memory

Private Attributes

Private member data

- [CPXENVptr env_](#)
CPLEX environment used in this class instance.
- [CPXLPptr lp_](#)
CPLEX model represented by this class instance.
- `int * hotStartCStat_`
Hotstart information.
- `int hotStartCStatSize_`
- `int * hotStartRStat_`
- `int hotStartRStatSize_`
- `int hotStartMaxIteration_`

Cached information derived from the CPLEX model

- `double * obj_`
Pointer to objective vector.
- `double * collower_`
Pointer to dense vector of variable lower bounds.
- `double * colupper_`
Pointer to dense vector of variable lower bounds.
- `char * rowsense_`
Pointer to dense vector of row sense indicators.
- `double * rhs_`
Pointer to dense vector of row right-hand side values.
- `double * rowrange_`
Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)
- `double * rowlower_`
Pointer to dense vector of row lower bounds.
- `double * rowupper_`
Pointer to dense vector of row upper bounds.
- `double * colsol_`
Pointer to primal solution vector.
- `double * rowsol_`
Pointer to dual solution vector.
- `double * redcost_`
Pointer to reduced cost vector.
- `double * rowact_`
Pointer to row activity (slack) vector.
- `CoinPackedMatrix * matrixByRow_`
Pointer to row-wise copy of problem matrix coefficients.
- `CoinPackedMatrix * matrixByCol_`
Pointer to row-wise copy of problem matrix coefficients.

Additional information needed for storing MIP problems

- `char * coltype_`
Pointer to dense vector of variable types (continuous, binary, integer)
- `int coltypesize_`
Size of allocated memory for coltype_.
- `bool probtymip_`
Stores whether CPLEX' prob type is currently set to MIP.
- `bool domipstart`
Whether to pass a column solution to CPLEX before starting MIP solve (copymipstart)
- `bool disableadvbasis`
Whether to disable use of advanced basis (if given)

Friends

- void [OsiCpxSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)

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

CPLEX specific public interfaces

- enum [keepCachedFlag](#) {
[KEEPCACHED_NONE](#) = 0, [KEEPCACHED_COLUMN](#) = 1, [KEEPCACHED_ROW](#) = 2, [KEEPCACHED_MATRIX](#) = 4,
[KEEPCACHED_RESULTS](#) = 8, [KEEPCACHED_PROBLEM](#) = [KEEPCACHED_COLUMN](#) | [KEEPCACHED_ROW](#) | [KEEPCACHED_MATRIX](#), [KEEPCACHED_ALL](#) = [KEEPCACHED_PROBLEM](#) | [KEEPCACHED_RESULTS](#),
[FREECACHED_COLUMN](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_COLUMN](#),
[FREECACHED_ROW](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_ROW](#), [FREECACHED_MATRIX](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_MATRIX](#), [FREECACHED_RESULTS](#) = [KEEPCACHED_ALL](#) & ~[KEEPCACHED_RESULTS](#) }

Get pointer to CPLEX model and free all specified cached data entries (combined with logical or-operator '|'):

- [CPXLpPtr getLpPtr](#) (int keepCached=[KEEPCACHED_NONE](#))
- [CPXENVptr getEnvironmentPtr](#) ()

Method to access CPLEX environment pointer.

Additional Inherited Members

8.12.1 Detailed Description

CPLEX Solver Interface.

Instantiation of [OsiCpxSolverInterface](#) for CPLEX

Definition at line 29 of file [OsiCpxSolverInterface.hpp](#).

8.12.2 Member Enumeration Documentation

8.12.2.1 enum [OsiCpxSolverInterface::keepCachedFlag](#)

Get pointer to CPLEX model and free all specified cached data entries (combined with logical or-operator '|'):

Enumerator

[KEEPCACHED_NONE](#) discard all cached data (default)
[KEEPCACHED_COLUMN](#) column information: objective values, lower and upper bounds, variable types
[KEEPCACHED_ROW](#) row information: right hand sides, ranges and senses, lower and upper bounds for row
[KEEPCACHED_MATRIX](#) problem matrix: matrix ordered by column and by row
[KEEPCACHED_RESULTS](#) LP solution: primal and dual solution, reduced costs, row activities.
[KEEPCACHED_PROBLEM](#) only discard cached LP solution
[KEEPCACHED_ALL](#) keep all cached data (similar to [getMutableLpPtr\(\)](#))
[FREECACHED_COLUMN](#) free only cached column and LP solution information
[FREECACHED_ROW](#) free only cached row and LP solution information
[FREECACHED_MATRIX](#) free only cached matrix and LP solution information
[FREECACHED_RESULTS](#) free only cached LP solution information

Definition at line 614 of file [OsiCpxSolverInterface.hpp](#).

8.12.3 Constructor & Destructor Documentation

8.12.3.1 `OsiCpxSolverInterface::OsiCpxSolverInterface ()`

Default Constructor.

8.12.3.2 `OsiCpxSolverInterface::OsiCpxSolverInterface (const OsiCpxSolverInterface &)`

Copy constructor.

8.12.3.3 `virtual OsiCpxSolverInterface::~~OsiCpxSolverInterface () [virtual]`

Destructor.

8.12.4 Member Function Documentation

8.12.4.1 `virtual void OsiCpxSolverInterface::initialSolve () [virtual]`

Solve initial LP relaxation.

Implements [OsiSolverInterface](#).

8.12.4.2 `virtual void OsiCpxSolverInterface::resolve () [virtual]`

Resolve an LP relaxation after problem modification.

Implements [OsiSolverInterface](#).

8.12.4.3 `virtual void OsiCpxSolverInterface::branchAndBound () [virtual]`

Invoke solver's built-in enumeration algorithm.

Implements [OsiSolverInterface](#).

8.12.4.4 `bool OsiCpxSolverInterface::setIntParam (OsiIntParam key, int value) [virtual]`

Set an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.12.4.5 `bool OsiCpxSolverInterface::setDbiParam (OsiDbiParam key, double value) [virtual]`

Set a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.12.4.6 `bool OsiCpxSolverInterface::setStrParam (OsiStrParam key, const std::string & value) [virtual]`

Set a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.12.4.7 `bool OsiCpxSolverInterface::getIntParam (OsiIntParam key, int & value) const [virtual]`

Get an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.12.4.8 `bool OsiCpxSolverInterface::getDbParam (OsiDbiParam key, double & value) const` [virtual]

Get a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.12.4.9 `bool OsiCpxSolverInterface::getStrParam (OsiStrParam key, std::string & value) const` [virtual]

Get a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.12.4.10 `void OsiCpxSolverInterface::setMipStart (bool value)` [inline]

Definition at line 76 of file `OsiCpxSolverInterface.hpp`.

8.12.4.11 `bool OsiCpxSolverInterface::getMipStart () const` [inline]

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

8.12.4.12 `virtual bool OsiCpxSolverInterface::isAbandoned () const` [virtual]

Are there a numerical difficulties?

Implements [OsiSolverInterface](#).

8.12.4.13 `virtual bool OsiCpxSolverInterface::isProvenOptimal () const` [virtual]

Is optimality proven?

Implements [OsiSolverInterface](#).

8.12.4.14 `virtual bool OsiCpxSolverInterface::isProvenPrimalInfeasible () const` [virtual]

Is primal infeasibility proven?

Implements [OsiSolverInterface](#).

8.12.4.15 `virtual bool OsiCpxSolverInterface::isProvenDualInfeasible () const` [virtual]

Is dual infeasibility proven?

Implements [OsiSolverInterface](#).

8.12.4.16 `virtual bool OsiCpxSolverInterface::isPrimalObjectiveLimitReached () const` [virtual]

Is the given primal objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.12.4.17 `virtual bool OsiCpxSolverInterface::isDualObjectiveLimitReached () const` [virtual]

Is the given dual objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.12.4.18 `virtual bool OsiCpxSolverInterface::isIterationLimitReached () const` [virtual]

Iteration limit reached?

Implements [OsiSolverInterface](#).

8.12.4.19 `CoinWarmStart* OsiCpxSolverInterface::getEmptyWarmStart () const [virtual]`

Get an empty warm start object.

This routine returns an empty `CoinWarmStartBasis` object. Its purpose is to provide a way to give a client a warm start basis object of the appropriate type, which can be resized and modified as desired.

Implements [OsiSolverInterface](#).

8.12.4.20 `virtual CoinWarmStart* OsiCpxSolverInterface::getWarmStart () const [virtual]`

Get warmstarting information.

Implements [OsiSolverInterface](#).

8.12.4.21 `virtual bool OsiCpxSolverInterface::setWarmStart (const CoinWarmStart * warmstart) [virtual]`

Set warmstarting information.

Return true/false depending on whether the warmstart information was accepted or not.

Implements [OsiSolverInterface](#).

8.12.4.22 `virtual void OsiCpxSolverInterface::markHotStart () [virtual]`

Create a hotstart point of the optimization process.

Reimplemented from [OsiSolverInterface](#).

8.12.4.23 `virtual void OsiCpxSolverInterface::solveFromHotStart () [virtual]`

Optimize starting from the hotstart.

Reimplemented from [OsiSolverInterface](#).

8.12.4.24 `virtual void OsiCpxSolverInterface::unmarkHotStart () [virtual]`

Delete the snapshot.

Reimplemented from [OsiSolverInterface](#).

8.12.4.25 `virtual int OsiCpxSolverInterface::getNumCols () const [virtual]`

Get number of columns.

Implements [OsiSolverInterface](#).

8.12.4.26 `virtual int OsiCpxSolverInterface::getNumRows () const [virtual]`

Get number of rows.

Implements [OsiSolverInterface](#).

8.12.4.27 `virtual int OsiCpxSolverInterface::getNumElements () const [virtual]`

Get number of nonzero elements.

Implements [OsiSolverInterface](#).

8.12.4.28 `virtual const double* OsiCpxSolverInterface::getColLower () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column lower bounds.

Implements [OsiSolverInterface](#).

8.12.4.29 `virtual const double* OsiCpxSolverInterface::getColUpper () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column upper bounds.

Implements [OsiSolverInterface](#).

8.12.4.30 `virtual const char* OsiCpxSolverInterface::getRowSense () const [virtual]`

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

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

Implements [OsiSolverInterface](#).

8.12.4.31 `virtual const double* OsiCpxSolverInterface::getRightHandSide () const [virtual]`

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

- if `rowsense()[i] == 'L'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'G'` then `rhs()[i] == rowlower()[i]`
- if `rowsense()[i] == 'R'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'N'` then `rhs()[i] == 0.0`

Implements [OsiSolverInterface](#).

8.12.4.32 `virtual const double* OsiCpxSolverInterface::getRowRange () const [virtual]`

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

- if `rowsense()[i] == 'R'` then `rowrange()[i] == rowupper()[i] - rowlower()[i]`
- if `rowsense()[i] != 'R'` then `rowrange()[i]` is 0.0

Implements [OsiSolverInterface](#).

8.12.4.33 `virtual const double* OsiCpxSolverInterface::getRowLower () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.12.4.34 `virtual const double* OsiCpxSolverInterface::getRowUpper () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.12.4.35 `virtual const double* OsiCpxSolverInterface::getObjCoefficients () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of objective function coefficients.

Implements [OsiSolverInterface](#).

8.12.4.36 `virtual double OsiCpxSolverInterface::getObjSense () const [virtual]`

Get objective function sense (1 for min (default), -1 for max)

Implements [OsiSolverInterface](#).

8.12.4.37 `virtual bool OsiCpxSolverInterface::isContinuous (int colNumber) const [virtual]`

Return true if column is continuous.

Implements [OsiSolverInterface](#).

8.12.4.38 `virtual const CoinPackedMatrix* OsiCpxSolverInterface::getMatrixByRow () const [virtual]`

Get pointer to row-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.12.4.39 `virtual const CoinPackedMatrix* OsiCpxSolverInterface::getMatrixByCol () const [virtual]`

Get pointer to column-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.12.4.40 `virtual double OsiCpxSolverInterface::getInfinity () const [virtual]`

Get solver's value for infinity.

Implements [OsiSolverInterface](#).

8.12.4.41 `virtual const double* OsiCpxSolverInterface::getColSolution () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of primal solution vector.

Implements [OsiSolverInterface](#).

8.12.4.42 `virtual const double* OsiCpxSolverInterface::getRowPrice () const [virtual]`

Get pointer to array[[getNumRows\(\)](#)] of dual prices.

Implements [OsiSolverInterface](#).

8.12.4.43 `virtual const double* OsiCpxSolverInterface::getReducedCost () const [virtual]`

Get a pointer to array[[getNumCols\(\)](#)] of reduced costs.

Implements [OsiSolverInterface](#).

8.12.4.44 `virtual const double* OsiCpxSolverInterface::getRowActivity () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.12.4.45 `virtual double OsiCpxSolverInterface::getObjValue () const [virtual]`

Get objective function value.

Implements [OsiSolverInterface](#).

8.12.4.46 `virtual int OsiCpxSolverInterface::getIterationCount () const [virtual]`

Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver.

Implements [OsiSolverInterface](#).

8.12.4.47 `virtual std::vector<double*> OsiCpxSolverInterface::getDualRays (int maxNumRays, bool fullRay = false) const [virtual]`

Get as many dual rays as the solver can provide.

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

The first [getNumRows\(\)](#) ray components will always be associated with the row duals (as returned by [getRowPrice\(\)](#)). If *fullRay* is true, the final [getNumCols\(\)](#) entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.12.4.48 `virtual std::vector<double*> OsiCpxSolverInterface::getPrimalRays (int maxNumRays) const [virtual]`

Get as many primal rays as the solver can provide.

(In case of proven dual infeasibility there should be at least one.)

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.12.4.49 `virtual void OsiCpxSolverInterface::setObjCoeff (int elementIndex, double elementValue) [virtual]`

Set an objective function coefficient.

Implements [OsiSolverInterface](#).

8.12.4.50 `virtual void OsiCpxSolverInterface::setObjCoeffSet (const int * indexFirst, const int * indexLast, const double * coeffList) [virtual]`

Set a a set of objective function coefficients.

Reimplemented from [OsiSolverInterface](#).

8.12.4.51 `virtual void OsiCpxSolverInterface::setColLower (int elementIndex, double elementValue) [virtual]`

Set a single column lower bound

Use `-COIN_DBL_MAX` for -infinity.

Implements [OsiSolverInterface](#).

8.12.4.52 `virtual void OsiCpxSolverInterface::setColUpper (int elementIndex, double elementValue) [virtual]`

Set a single column upper bound

Use `COIN_DBL_MAX` for infinity.

Implements [OsiSolverInterface](#).

8.12.4.53 `virtual void OsiCpxSolverInterface::setColBounds (int elementIndex, double lower, double upper) [virtual]`

Set a single column lower and upper bound

The default implementation just invokes `setColLower()` and `setColUpper()`

Reimplemented from [OsiSolverInterface](#).

8.12.4.54 `virtual void OsiCpxSolverInterface::setColSetBounds (const int * indexFirst, const int * indexLast, const double * boundList) [virtual]`

Set the bounds on a number of columns simultaneously

The default implementation just invokes `setColLower()` and `setColUpper()` over and over again.

Parameters

<code><code>[<i>indexfirst</i>,<i>index-</i> <i>Last</i>]</code></code>	contains the indices of the constraints whose either bound changes
<code><i>boundList</i></code>	the new lower/upper bound pairs for the variables

Reimplemented from [OsiSolverInterface](#).

8.12.4.55 `virtual void OsiCpxSolverInterface::setRowLower (int elementIndex, double elementValue) [virtual]`

Set a single row lower bound

Use `-COIN_DBL_MAX` for -infinity.

Implements [OsiSolverInterface](#).

8.12.4.56 `virtual void OsiCpxSolverInterface::setRowUpper (int elementIndex, double elementValue) [virtual]`

Set a single row upper bound

Use `COIN_DBL_MAX` for infinity.

Implements [OsiSolverInterface](#).

8.12.4.57 `virtual void OsiCpxSolverInterface::setRowBounds (int elementIndex, double lower, double upper) [virtual]`

Set a single row lower and upper bound

The default implementation just invokes `setRowLower()` and `setRowUpper()`

Reimplemented from [OsiSolverInterface](#).

8.12.4.58 `virtual void OsiCpxSolverInterface::setRowType (int index, char sense, double rightHandSide, double range)`
`[virtual]`

Set the type of a single row

Implements [OsiSolverInterface](#).

8.12.4.59 `virtual void OsiCpxSolverInterface::setRowSetBounds (const int * indexFirst, const int * indexLast, const double * boundList)`
`[virtual]`

Set the bounds on a number of rows simultaneously

The default implementation just invokes `setRowLower ()` and `setRowUpper ()` over and over again.

Parameters

<code><code>[<i>indexFirst</i>,<i>indexLast</i>]</code>	contains the indices of the constraints whose either bound changes
<code><i>boundList</i></code>	the new lower/upper bound pairs for the constraints

Reimplemented from [OsiSolverInterface](#).

8.12.4.60 `virtual void OsiCpxSolverInterface::setRowSetTypes (const int * indexFirst, const int * indexLast, const char * senseList, const double * rhsList, const double * rangeList)`
`[virtual]`

Set the type of a number of rows simultaneously

The default implementation just invokes `setRowType ()` and over and over again.

Parameters

<code><code>[<i>indexFirst</i>,<i>indexLast</i>]</code>	contains the indices of the constraints whose type changes
<code><i>senseList</i></code>	the new senses
<code><i>rhsList</i></code>	the new right hand sides
<code><i>rangeList</i></code>	the new ranges

Reimplemented from [OsiSolverInterface](#).

8.12.4.61 `virtual void OsiCpxSolverInterface::setContinuous (int index)`
`[virtual]`

Set the index-th variable to be a continuous variable.

Implements [OsiSolverInterface](#).

8.12.4.62 `virtual void OsiCpxSolverInterface::setInteger (int index)`
`[virtual]`

Set the index-th variable to be an integer variable.

Implements [OsiSolverInterface](#).

8.12.4.63 `virtual void OsiCpxSolverInterface::setContinuous (const int * indices, int len)`
`[virtual]`

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

Reimplemented from [OsiSolverInterface](#).

8.12.4.64 `virtual void OsiCpxSolverInterface::setInteger (const int * indices, int len) [virtual]`

Set the variables listed in *indices* (which is of length *len*) to be integer variables.

Reimplemented from [OsiSolverInterface](#).

8.12.4.65 `virtual void OsiCpxSolverInterface::setObjSense (double s) [virtual]`

Set objective function sense (1 for min (default), -1 for max,)

Implements [OsiSolverInterface](#).

8.12.4.66 `virtual void OsiCpxSolverInterface::setColSolution (const double * colsol) [virtual]`

Set the primal solution column values.

colsol[*numcols*()] is an array of values of the problem column variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of *colsol*() until changed by another call to *setColsol*() or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

Implements [OsiSolverInterface](#).

8.12.4.67 `virtual void OsiCpxSolverInterface::setRowPrice (const double * rowprice) [virtual]`

Set dual solution vector.

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

Implements [OsiSolverInterface](#).

8.12.4.68 `virtual void OsiCpxSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [virtual]`

Add a column (primal variable) to the problem.

Implements [OsiSolverInterface](#).

8.12.4.69 `virtual void OsiCpxSolverInterface::addCols (const int numcols, const CoinPackedVectorBase *const * cols, const double * collb, const double * colub, const double * obj) [virtual]`

Add a set of columns (primal variables) to the problem.

The default implementation simply makes repeated calls to [addCol](#)().

Reimplemented from [OsiSolverInterface](#).

8.12.4.70 `virtual void OsiCpxSolverInterface::deleteCols (const int num, const int * colIndices) [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](#).

8.12.4.71 `virtual void OsiCpxSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.12.4.72 `virtual void OsiCpxSolverInterface::addRow (const CoinPackedVectorBase & vec, const char rowsen, const double rowrhs, const double rowrng) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.12.4.73 `virtual void OsiCpxSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const double * rowlb, const double * rowub) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.12.4.74 `virtual void OsiCpxSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const char * rowsen, const double * rowrhs, const double * rowrng) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.12.4.75 `virtual void OsiCpxSolverInterface::deleteRows (const int num, const int * rowIndices) [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](#).

8.12.4.76 `virtual void OsiCpxSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub) [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](#).

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

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

For default values see the previous method.

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

Implements [OsiSolverInterface](#).

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

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

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

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

Implements [OsiSolverInterface](#).

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

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

For default values see the previous method.

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

Implements [OsiSolverInterface](#).

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

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

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

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

8.12.4.82 `virtual int OsiCpxSolverInterface::readMps (const char * filename, const char * extension = "mps") [virtual]`

Read an mps file from the given filename.

Reimplemented from [OsiSolverInterface](#).

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

Write the problem into an mps file of the given filename.

If objSense is non zero then -1.0 forces the code to write a maximization objective and +1.0 to write a minimization one.
If 0.0 then solver can do what it wants

Implements [OsiSolverInterface](#).

8.12.4.84 void OsiCpxSolverInterface::passInMessageHandler (CoinMessageHandler * *handler*) [virtual]

Pass in a message handler It is the client's responsibility to destroy a message handler installed by this routine; it will not be destroyed when the solver interface is destroyed.

Reimplemented from [OsiSolverInterface](#).

8.12.4.85 CPXLPptr OsiCpxSolverInterface::getLpPtr (int *keepCached* = KEEPCACHED_NONE)

8.12.4.86 CPXENVptr OsiCpxSolverInterface::getEnvironmentPtr ()

Method to access CPLEX environment pointer.

8.12.4.87 const char* OsiCpxSolverInterface::getCtype () const

return a vector of variable types (continous, binary, integer)

8.12.4.88 virtual OsiSolverInterface* OsiCpxSolverInterface::clone (bool *copyData* = true) const [virtual]

Clone.

Implements [OsiSolverInterface](#).

8.12.4.89 OsiCpxSolverInterface& OsiCpxSolverInterface::operator= (const OsiCpxSolverInterface & *rhs*)

Assignment operator.

8.12.4.90 virtual void OsiCpxSolverInterface::reset () [virtual]

Resets as if default constructor.

Reimplemented from [OsiSolverInterface](#).

8.12.4.91 virtual int OsiCpxSolverInterface::canDoSimplexInterface () const [virtual]

Returns 1 if can just do getBlv etc 2 if has all OsiSimplex methods and 0 if it has none.

Reimplemented from [OsiSolverInterface](#).

8.12.4.92 virtual void OsiCpxSolverInterface::enableSimplexInterface (int *doingPrimal*) [inline],[virtual]

Useless function, defined only for compatibility with OsiSimplexInterface.

Definition at line 700 of file OsiCpxSolverInterface.hpp.

8.12.4.93 virtual void OsiCpxSolverInterface::disableSimplexInterface () [inline],[virtual]

Useless function, defined only for compatibility with OsiSimplexInterface.

Reimplemented from [OsiSolverInterface](#).

Definition at line 705 of file OsiCpxSolverInterface.hpp.

8.12.4.94 virtual void OsiCpxSolverInterface::enableFactorization () const [inline],[virtual]

Useless function, defined only for compatibility with OsiSimplexInterface.

Reimplemented from [OsiSolverInterface](#).

Definition at line 710 of file OsiCpxSolverInterface.hpp.

8.12.4.95 `virtual void OsiCpxSolverInterface::disableFactorization () const [inline],[virtual]`

Useless function, defined only for compatibility with OsiSimplexInterface.

Reimplemented from [OsiSolverInterface](#).

Definition at line 715 of file OsiCpxSolverInterface.hpp.

8.12.4.96 `virtual bool OsiCpxSolverInterface::basisIsAvailable () const [virtual]`

Returns true if a basis is available.

Reimplemented from [OsiSolverInterface](#).

8.12.4.97 `virtual void OsiCpxSolverInterface::getBasisStatus (int * cstat, int * rstat) const [virtual]`

Returns a basis status of the structural/artificial variables At present as warm start i.e 0: free, 1: basic, 2: upper, 3: lower.

Reimplemented from [OsiSolverInterface](#).

8.12.4.98 `virtual void OsiCpxSolverInterface::getBlInvARow (int row, double * z, double * slack = NULL) const [virtual]`

Get a row of the tableau (slack part in slack if not NULL)

Reimplemented from [OsiSolverInterface](#).

8.12.4.99 `virtual void OsiCpxSolverInterface::getBlInvRow (int row, double * z) const [virtual]`

Get a row of the basis inverse.

Reimplemented from [OsiSolverInterface](#).

8.12.4.100 `virtual void OsiCpxSolverInterface::getBlInvACol (int col, double * vec) const [virtual]`

Get a column of the tableau.

Reimplemented from [OsiSolverInterface](#).

8.12.4.101 `virtual void OsiCpxSolverInterface::getBlInvCol (int col, double * vec) const [virtual]`

Get a column of the basis inverse.

Reimplemented from [OsiSolverInterface](#).

8.12.4.102 `virtual void OsiCpxSolverInterface::getBasics (int * index) const [virtual]`

Get indices of the pivot variable in each row (order of indices corresponds to the order of elements in a vector returned by [getBlInvACol\(\)](#) and [getBlInvCol\(\)](#)).

Reimplemented from [OsiSolverInterface](#).

8.12.4.103 `void OsiCpxSolverInterface::switchToLP ()`

switches CPLEX to prob type LP

8.12.4.104 `void OsiCpxSolverInterface::switchToMIP ()`

switches CPLEX to prob type MIP

8.12.4.105 `virtual void OsiCpxSolverInterface::applyRowCut (const OsiRowCut & rc)` [protected],[virtual]

Apply a row cut. Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.12.4.106 `virtual void OsiCpxSolverInterface::applyColCut (const OsiColCut & cc)` [protected],[virtual]

Apply a column cut (bound adjustment).

Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.12.4.107 `void OsiCpxSolverInterface::resizeColType (int minsize)` [private]

resizes coltype_ vector to be able to store at least minsize elements

8.12.4.108 `void OsiCpxSolverInterface::freeColType ()` [private]

frees colsize_ vector

8.12.4.109 `CPXLPptr OsiCpxSolverInterface::getMutableLpPtr () const` [private]

Get LP Pointer for const methods.

8.12.4.110 `void OsiCpxSolverInterface::gutsOfCopy (const OsiCpxSolverInterface & source)` [private]

The real work of a copy constructor (used by copy and assignment)

8.12.4.111 `void OsiCpxSolverInterface::gutsOfConstructor ()` [private]

The real work of the constructor.

8.12.4.112 `void OsiCpxSolverInterface::gutsOfDestructor ()` [private]

The real work of the destructor.

8.12.4.113 `void OsiCpxSolverInterface::freeCachedColRim ()` [private]

free cached column rim vectors

8.12.4.114 `void OsiCpxSolverInterface::freeCachedRowRim ()` [private]

free cached row rim vectors

8.12.4.115 `void OsiCpxSolverInterface::freeCachedResults ()` [private]

free cached result vectors

8.12.4.116 `void OsiCpxSolverInterface::freeCachedMatrix ()` [private]

free cached matrices

8.12.4.117 `void OsiCpxSolverInterface::freeCachedData (int keepCached = KEEPCACHED_NONE)` [private]

free all cached data (except specified entries, see [getLpPtr\(\)](#))

8.12.4.118 `void OsiCpxSolverInterface::freeAllMemory ()` [private]

free all allocated memory

8.12.5 Friends And Related Function Documentation

8.12.5.1 `void OsiCpxSolverInterfaceUnitTest (const std::string & mpsDir, const std::string & netlibDir)` [friend]

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

8.12.6 Member Data Documentation

8.12.6.1 `CPXENVptr OsiCpxSolverInterface::env_` [mutable],[private]

CPLEX environment used in this class instance.

Definition at line 815 of file `OsiCpxSolverInterface.hpp`.

8.12.6.2 `CPXLPptr OsiCpxSolverInterface::lp_` [mutable],[private]

CPLEX model represented by this class instance.

Definition at line 817 of file `OsiCpxSolverInterface.hpp`.

8.12.6.3 `int* OsiCpxSolverInterface::hotStartCStat_` [private]

Hotstart information.

Definition at line 820 of file `OsiCpxSolverInterface.hpp`.

8.12.6.4 `int OsiCpxSolverInterface::hotStartCStatSize_` [private]

Definition at line 821 of file `OsiCpxSolverInterface.hpp`.

8.12.6.5 `int* OsiCpxSolverInterface::hotStartRStat_` [private]

Definition at line 822 of file `OsiCpxSolverInterface.hpp`.

8.12.6.6 `int OsiCpxSolverInterface::hotStartRStatSize_` [private]

Definition at line 823 of file `OsiCpxSolverInterface.hpp`.

8.12.6.7 `int OsiCpxSolverInterface::hotStartMaxIteration_` [private]

Definition at line 824 of file `OsiCpxSolverInterface.hpp`.

8.12.6.8 `double* OsiCpxSolverInterface::obj_` [mutable],[private]

Pointer to objective vector.

Definition at line 829 of file `OsiCpxSolverInterface.hpp`.

8.12.6.9 `double* OsiCpxSolverInterface::collower_` [mutable], [private]

Pointer to dense vector of variable lower bounds.

Definition at line 832 of file OsiCpxSolverInterface.hpp.

8.12.6.10 `double* OsiCpxSolverInterface::colupper_` [mutable], [private]

Pointer to dense vector of variable lower bounds.

Definition at line 835 of file OsiCpxSolverInterface.hpp.

8.12.6.11 `char* OsiCpxSolverInterface::rowsense_` [mutable], [private]

Pointer to dense vector of row sense indicators.

Definition at line 838 of file OsiCpxSolverInterface.hpp.

8.12.6.12 `double* OsiCpxSolverInterface::rhs_` [mutable], [private]

Pointer to dense vector of row right-hand side values.

Definition at line 841 of file OsiCpxSolverInterface.hpp.

8.12.6.13 `double* OsiCpxSolverInterface::rowrange_` [mutable], [private]

Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)

Definition at line 844 of file OsiCpxSolverInterface.hpp.

8.12.6.14 `double* OsiCpxSolverInterface::rowlower_` [mutable], [private]

Pointer to dense vector of row lower bounds.

Definition at line 847 of file OsiCpxSolverInterface.hpp.

8.12.6.15 `double* OsiCpxSolverInterface::rowupper_` [mutable], [private]

Pointer to dense vector of row upper bounds.

Definition at line 850 of file OsiCpxSolverInterface.hpp.

8.12.6.16 `double* OsiCpxSolverInterface::colsol_` [mutable], [private]

Pointer to primal solution vector.

Definition at line 853 of file OsiCpxSolverInterface.hpp.

8.12.6.17 `double* OsiCpxSolverInterface::rowsol_` [mutable], [private]

Pointer to dual solution vector.

Definition at line 856 of file OsiCpxSolverInterface.hpp.

8.12.6.18 `double* OsiCpxSolverInterface::redcost_` [mutable], [private]

Pointer to reduced cost vector.

Definition at line 859 of file OsiCpxSolverInterface.hpp.

8.12.6.19 `double* OsiCpxSolverInterface::rowact_` [mutable],[private]

Pointer to row activity (slack) vector.

Definition at line 862 of file OsiCpxSolverInterface.hpp.

8.12.6.20 `CoinPackedMatrix* OsiCpxSolverInterface::matrixByRow_` [mutable],[private]

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 865 of file OsiCpxSolverInterface.hpp.

8.12.6.21 `CoinPackedMatrix* OsiCpxSolverInterface::matrixByCol_` [mutable],[private]

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 868 of file OsiCpxSolverInterface.hpp.

8.12.6.22 `char* OsiCpxSolverInterface::coltype_` [private]

Pointer to dense vector of variable types (continuous, binary, integer)

Definition at line 874 of file OsiCpxSolverInterface.hpp.

8.12.6.23 `int OsiCpxSolverInterface::coltypesize_` [private]

Size of allocated memory for coltype_.

Definition at line 877 of file OsiCpxSolverInterface.hpp.

8.12.6.24 `bool OsiCpxSolverInterface::probtymip_` [mutable],[private]

Stores whether CPLEX' prob type is currently set to MIP.

Definition at line 880 of file OsiCpxSolverInterface.hpp.

8.12.6.25 `bool OsiCpxSolverInterface::domipstart` [private]

Whether to pass a column solution to CPLEX before starting MIP solve (copymipstart)

Definition at line 883 of file OsiCpxSolverInterface.hpp.

8.12.6.26 `bool OsiCpxSolverInterface::disableadvbasis` [private]

Whether to disable use of advanced basis (if given)

Definition at line 886 of file OsiCpxSolverInterface.hpp.

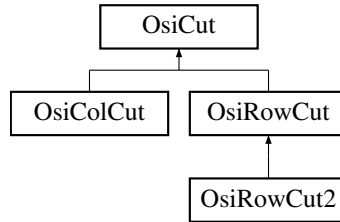
The documentation for this class was generated from the following file:

- </home/ted/COIN/trunk/Osi/src/OsiCpx/OsiCpxSolverInterface.hpp>

8.13 OsiCut Class Reference

```
#include <OsiCut.hpp>
```

Inheritance diagram for OsiCut:



Public Member Functions

Effectiveness

- void `setEffectiveness` (double e)
Set effectiveness.
- double `effectiveness` () const
Get effectiveness.

GloballyValid

- void `setGloballyValid` (bool trueFalse)
Set globallyValid (nonzero true)
- void `setGloballyValid` ()
- void `setNotGloballyValid` ()
- bool `globallyValid` () const
Get globallyValid.
- void `setGloballyValidAsInteger` (int trueFalse)
Set globallyValid as integer (nonzero true)
- int `globallyValidAsInteger` () const
Get globallyValid.

Debug stuff

- virtual void `print` () const
Print cuts in collection.

Comparison operators

- virtual bool `operator==` (const `OsiCut` &rhs) const
equal. 2 cuts are equal if there effectiveness are equal
- virtual bool `operator!=` (const `OsiCut` &rhs) const
not equal
- virtual bool `operator<` (const `OsiCut` &rhs) const
less than. True if this.effectiveness < rhs.effectiveness
- virtual bool `operator>` (const `OsiCut` &rhs) const
less than. True if this.effectiveness > rhs.effectiveness

Sanity checks on cut

- virtual bool `consistent` () const =0
Returns true if the cut is consistent with respect to itself, without considering any data in the model.
- virtual bool `consistent` (const `OsiSolverInterface` &si) const =0
Returns true if cut is consistent when considering the solver interface's model.
- virtual bool `infeasible` (const `OsiSolverInterface` &si) const =0
Returns true if the cut is infeasible "with respect to itself" and cannot be satisfied.
- virtual double `violated` (const double *solution) const =0
Returns infeasibility of the cut with respect to solution passed in i.e.

Protected Member Functions

Constructors and destructors

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

Private Attributes

Private member data

- [double effectiveness_](#)
Effectiveness.
- [int globallyValid_](#)
If cut has global validity i.e. can be used anywhere in tree.

8.13.1 Detailed Description

Definition at line 36 of file OsiCut.hpp.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 [OsiCut::OsiCut \(\)](#) `[protected]`

Default Constructor.

8.13.2.2 [OsiCut::OsiCut \(const OsiCut & \)](#) `[protected]`

Copy constructor.

8.13.2.3 [virtual OsiCut::~~OsiCut \(\)](#) `[protected],[virtual]`

Destructor.

8.13.3 Member Function Documentation

8.13.3.1 [void OsiCut::setEffectiveness \(double e \)](#) `[inline]`

Set effectiveness.

Definition at line 209 of file OsiCut.hpp.

8.13.3.2 [double OsiCut::effectiveness \(\) const](#) `[inline]`

Get effectiveness.

Definition at line 210 of file OsiCut.hpp.

8.13.3.3 `void OsiCut::setGloballyValid (bool trueFalse)` `[inline]`

Set globallyValid (nonzero true)

Definition at line 52 of file OsiCut.hpp.

8.13.3.4 `void OsiCut::setGloballyValid ()` `[inline]`

Definition at line 54 of file OsiCut.hpp.

8.13.3.5 `void OsiCut::setNotGloballyValid ()` `[inline]`

Definition at line 56 of file OsiCut.hpp.

8.13.3.6 `bool OsiCut::globallyValid () const` `[inline]`

Get globallyValid.

Definition at line 59 of file OsiCut.hpp.

8.13.3.7 `void OsiCut::setGloballyValidAsInteger (int trueFalse)` `[inline]`

Set globallyValid as integer (nonzero true)

Definition at line 62 of file OsiCut.hpp.

8.13.3.8 `int OsiCut::globallyValidAsInteger () const` `[inline]`

Get globallyValid.

Definition at line 65 of file OsiCut.hpp.

8.13.3.9 `virtual void OsiCut::print () const` `[inline], [virtual]`

Print cuts in collection.

Reimplemented in [OsiRowCut](#), and [OsiColCut](#).

Definition at line 72 of file OsiCut.hpp.

8.13.3.10 `bool OsiCut::operator== (const OsiCut & rhs) const` `[inline], [virtual]`

equal. 2 cuts are equal if there effectiveness are equal

Definition at line 226 of file OsiCut.hpp.

8.13.3.11 `bool OsiCut::operator!= (const OsiCut & rhs) const` `[inline], [virtual]`

not equal

Definition at line 231 of file OsiCut.hpp.

8.13.3.12 `bool OsiCut::operator< (const OsiCut & rhs) const` `[inline], [virtual]`

less than. True if this.effectiveness < rhs.effectiveness

Definition at line 236 of file OsiCut.hpp.

8.13.3.13 `bool OsiCut::operator> (const OsiCut & rhs) const` `[inline], [virtual]`

less than. True if this.effectiveness > rhs.effectiveness

Definition at line 241 of file OsiCut.hpp.

8.13.3.14 `virtual bool OsiCut::consistent () const [inline],[pure virtual]`

Returns true if the cut is consistent with respect to itself, without considering any data in the model.

For example, it might check to ensure that a column index is not negative.

Implemented in [OsiRowCut](#), and [OsiColCut](#).

8.13.3.15 `virtual bool OsiCut::consistent (const OsiSolverInterface & si) const [inline],[pure virtual]`

Returns true if cut is consistent when considering the solver interface's model.

For example, it might check to ensure that a column index is not greater than the number of columns in the model.

Assumes [consistent\(\)](#) is true.

Implemented in [OsiRowCut](#), and [OsiColCut](#).

8.13.3.16 `virtual bool OsiCut::infeasible (const OsiSolverInterface & si) const [inline],[pure virtual]`

Returns true if the cut is infeasible "with respect to itself" and cannot be satisfied.

This method does NOT check whether adding the cut to the solver interface's model will make the -model- infeasible. A cut which returns `linfeasible(si)` may very well make the model infeasible. (Of course, adding a cut with returns `infeasible(si)` will make the model infeasible.)

The "with respect to itself" is in quotes because in the case where the cut simply replaces existing bounds, it may make sense to test infeasibility with respect to the current bounds held in the solver interface's model. For example, if the cut has a single variable in it, it might check that the maximum of new and existing lower bounds is greater than the minimum of the new and existing upper bounds.

Assumes that `consistent(si)` is true.

Infeasible cuts can be a useful mechanism for a cut generator to inform the solver interface that its detected infeasibility of the problem.

Implemented in [OsiRowCut](#), and [OsiColCut](#).

8.13.3.17 `virtual double OsiCut::violated (const double * solution) const [pure virtual]`

Returns infeasibility of the cut with respect to solution passed in i.e.

is positive if cuts off that solution. solution is `getNumCols()` long..

Implemented in [OsiRowCut](#), and [OsiColCut](#).

8.13.3.18 `OsiCut& OsiCut::operator= (const OsiCut & rhs) [protected]`

Assignment operator.

8.13.4 Member Data Documentation

8.13.4.1 `double OsiCut::effectiveness_ [private]`

Effectiveness.

Definition at line 193 of file OsiCut.hpp.

8.13.4.2 int OsiCut::globallyValid_ [private]

If cut has global validity i.e. can be used anywhere in tree.

Definition at line 195 of file OsiCut.hpp.

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

- [/home/ted/COIN/trunk/Osi/src/Osi/OsiCut.hpp](#)

8.14 OsiCuts::OsiCutCompare Class Reference

Public Member Functions

- bool [operator\(\)](#) (const [OsiCut](#) *c1P, const [OsiCut](#) *c2P)
Function for sorting cuts by effectiveness.

8.14.1 Detailed Description

Definition at line 275 of file OsiCuts.hpp.

8.14.2 Member Function Documentation

8.14.2.1 bool OsiCuts::OsiCutCompare::operator() (const OsiCut * c1P, const OsiCut * c2P) [inline]

Function for sorting cuts by effectiveness.

Definition at line 279 of file OsiCuts.hpp.

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

- [/home/ted/COIN/trunk/Osi/src/Osi/OsiCuts.hpp](#)

8.15 OsiCuts Class Reference

Collections of row cuts and column cuts.

```
#include <OsiCuts.hpp>
```

Classes

- class [const_iterator](#)
Const Iterator.
- class [iterator](#)
Iterator.
- class [OsiCutCompare](#)

Public Member Functions

Inserting a cut into collection

- void [insert](#) (const [OsiRowCut](#) &rc)

- Insert a row cut.*
 - void `insertIfNotDuplicate` (`OsiRowCut` &rc, `CoinAbsFltEq` treatAsSame=`CoinAbsFltEq`(1.0e-12))
- Insert a row cut unless it is a duplicate - cut may get sorted.*
 - void `insertIfNotDuplicate` (`OsiRowCut` &rc, `CoinRelFltEq` treatAsSame)
- Insert a row cut unless it is a duplicate - cut may get sorted.*
 - void `insert` (const `OsiColCut` &cc)
- Insert a column cut.*
 - void `insert` (`OsiRowCut` *&rcPtr)
- Insert a row cut.*
 - void `insert` (`OsiColCut` *&ccPtr)
- Insert a column cut.*
 - void `insert` (const `OsiCuts` &cs)
- Insert a set of cuts.*

Number of cuts in collection

- int `sizeRowCuts` () const
Number of row cuts in collection.
- int `sizeColCuts` () const
Number of column cuts in collection.
- int `sizeCuts` () const
Number of cuts in collection.

Debug stuff

- void `printCuts` () const
Print cuts in collection.

Get a cut from collection

- `OsiRowCut` * `rowCutPtr` (int i)
Get pointer to i'th row cut.
- const `OsiRowCut` * `rowCutPtr` (int i) const
Get const pointer to i'th row cut.
- `OsiColCut` * `colCutPtr` (int i)
Get pointer to i'th column cut.
- const `OsiColCut` * `colCutPtr` (int i) const
Get const pointer to i'th column cut.
- `OsiRowCut` & `rowCut` (int i)
Get reference to i'th row cut.
- const `OsiRowCut` & `rowCut` (int i) const
Get const reference to i'th row cut.
- `OsiColCut` & `colCut` (int i)
Get reference to i'th column cut.
- const `OsiColCut` & `colCut` (int i) const
Get const reference to i'th column cut.
- const `OsiCut` * `mostEffectiveCutPtr` () const
Get const pointer to the most effective cut.
- `OsiCut` * `mostEffectiveCutPtr` ()
Get pointer to the most effective cut.

Deleting cut from collection

- void `eraseRowCut` (int i)

- *Remove i'th row cut from collection.*
- void `eraseColCut` (int i)
Remove i'th column cut from collection.
- `OsiRowCut * rowCutPtrAndZap` (int i)
Get pointer to i'th row cut and remove ptr from collection.
- void `dumpCuts` ()
Clear all row cuts without deleting them.
- void `eraseAndDumpCuts` (const std::vector< int > to_erase)
Selective delete and clear for row cuts.

Sorting collection

- void `sort` ()
Cuts with greatest effectiveness are first.

Iterators

Example of using an iterator to sum effectiveness of all cuts in the collection.

```
double sumEff=0.0;
for ( OsiCuts::iterator it=cuts.begin(); it!=cuts.end(); ++it )
    sumEff+= (*it)->effectiveness();
```

- `iterator begin` ()
Get iterator to beginning of collection.
- `const_iterator begin` () const
Get const iterator to beginning of collection.
- `iterator end` ()
Get iterator to end of collection.
- `const_iterator end` () const
Get const iterator to end of collection.

Constructors and destructors

- `OsiCuts` ()
Default constructor.
- `OsiCuts` (const `OsiCuts` &)
Copy constructor.
- `OsiCuts & operator=` (const `OsiCuts` &rhs)
Assignment operator.
- virtual `~OsiCuts` ()
Destructor.

Private Member Functions

Private methods

- void `gutsOfCopy` (const `OsiCuts` &source)
Copy internal data.
- void `gutsOfDestructor` ()
Delete internal data.

Private Attributes

Private member data

- [OsiVectorRowCutPtr rowCutPtrs_](#)
Vector of row cuts pointers.
- [OsiVectorColCutPtr colCutPtrs_](#)
Vector of column cuts pointers.

Friends

- void [OsiCutsUnitTest](#) ()
A function that tests the methods in the [OsiCuts](#) class.

8.15.1 Detailed Description

Collections of row cuts and column cuts.

Definition at line 19 of file OsiCuts.hpp.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 [OsiCuts::OsiCuts \(\)](#)

Default constructor.

8.15.2.2 [OsiCuts::OsiCuts \(const \[OsiCuts\]\(#\) & \)](#)

Copy constructor.

8.15.2.3 [virtual \[OsiCuts::~~OsiCuts \\(\\)\]\(#\) \[virtual\]](#)

Destructor.

8.15.3 Member Function Documentation

8.15.3.1 [void \[OsiCuts::insert \\(const \\[OsiRowCut\\]\\(#\\) & rc \\)\]\(#\) \[inline\]](#)

Insert a row cut.

Definition at line 306 of file OsiCuts.hpp.

8.15.3.2 [void \[OsiCuts::insertIfNotDuplicate \\(\\[OsiRowCut\\]\\(#\\) & rc, \\[CoinAbsFltEq\\]\\(#\\) treatAsSame = \\[CoinAbsFltEq\\]\\(#\\)\\(1.0e-12\\) \\)\]\(#\)](#)

Insert a row cut unless it is a duplicate - cut may get sorted.

Duplicate is defined as [CoinAbsFltEq](#) says same

8.15.3.3 [void \[OsiCuts::insertIfNotDuplicate \\(\\[OsiRowCut\\]\\(#\\) & rc, \\[CoinRelFltEq\\]\\(#\\) treatAsSame \\)\]\(#\)](#)

Insert a row cut unless it is a duplicate - cut may get sorted.

Duplicate is defined as [CoinRelFltEq](#) says same

8.15.3.4 `void OsiCuts::insert (const OsiColCut & cc) [inline]`

Insert a column cut.

Definition at line 312 of file OsiCuts.hpp.

8.15.3.5 `void OsiCuts::insert (OsiRowCut *& rcPtr) [inline]`

Insert a row cut.

The [OsiCuts](#) object takes control of the cut object. On return, `rcPtr` is NULL.

Definition at line 319 of file OsiCuts.hpp.

8.15.3.6 `void OsiCuts::insert (OsiColCut *& ccPtr) [inline]`

Insert a column cut.

The [OsiCuts](#) object takes control of the cut object. On return `ccPtr` is NULL.

Definition at line 324 of file OsiCuts.hpp.

8.15.3.7 `void OsiCuts::insert (const OsiCuts & cs) [inline]`

Insert a set of cuts.

Definition at line 347 of file OsiCuts.hpp.

8.15.3.8 `int OsiCuts::sizeRowCuts () const [inline]`

Number of row cuts in collection.

Definition at line 374 of file OsiCuts.hpp.

8.15.3.9 `int OsiCuts::sizeColCuts () const [inline]`

Number of column cuts in collection.

Definition at line 376 of file OsiCuts.hpp.

8.15.3.10 `int OsiCuts::sizeCuts () const [inline]`

Number of cuts in collection.

Definition at line 378 of file OsiCuts.hpp.

8.15.3.11 `void OsiCuts::printCuts () const [inline]`

Print cuts in collection.

Definition at line 423 of file OsiCuts.hpp.

8.15.3.12 `OsiRowCut * OsiCuts::rowCutPtr (int i) [inline]`

Get pointer to i'th row cut.

Definition at line 386 of file OsiCuts.hpp.

8.15.3.13 `const OsiRowCut * OsiCuts::rowCutPtr (int i) const [inline]`

Get const pointer to i'th row cut.

Definition at line 384 of file OsiCuts.hpp.

8.15.3.14 `OsiColCut * OsiCuts::colCutPtr (int i) [inline]`

Get pointer to i'th column cut.

Definition at line 387 of file OsiCuts.hpp.

8.15.3.15 `const OsiColCut * OsiCuts::colCutPtr (int i) const [inline]`

Get const pointer to i'th column cut.

Definition at line 385 of file OsiCuts.hpp.

8.15.3.16 `OsiRowCut & OsiCuts::rowCut (int i) [inline]`

Get reference to i'th row cut.

Definition at line 391 of file OsiCuts.hpp.

8.15.3.17 `const OsiRowCut & OsiCuts::rowCut (int i) const [inline]`

Get const reference to i'th row cut.

Definition at line 389 of file OsiCuts.hpp.

8.15.3.18 `OsiColCut & OsiCuts::colCut (int i) [inline]`

Get reference to i'th column cut.

Definition at line 392 of file OsiCuts.hpp.

8.15.3.19 `const OsiColCut & OsiCuts::colCut (int i) const [inline]`

Get const reference to i'th column cut.

Definition at line 390 of file OsiCuts.hpp.

8.15.3.20 `const OsiCut * OsiCuts::mostEffectiveCutPtr () const [inline]`

Get const pointer to the most effective cut.

Definition at line 397 of file OsiCuts.hpp.

8.15.3.21 `OsiCut * OsiCuts::mostEffectiveCutPtr () [inline]`

Get pointer to the most effective cut.

Definition at line 403 of file OsiCuts.hpp.

8.15.3.22 `void OsiCuts::eraseRowCut (int i) [inline]`

Remove i'th row cut from collection.

Definition at line 442 of file OsiCuts.hpp.

8.15.3.23 `void OsiCuts::eraseColCut (int i) [inline]`

Remove i'th column cut from collection.

Definition at line 447 of file OsiCuts.hpp.

8.15.3.24 OsiRowCut * OsiCuts::rowCutPtrAndZap (int i) [inline]

Get pointer to i'th row cut and remove ptr from collection.

Definition at line 454 of file OsiCuts.hpp.

8.15.3.25 void OsiCuts::dumpCuts () [inline]

Clear all row cuts without deleting them.

Handy in case one wants to use CGL without managing cuts in one of the OSI containers. Client is ultimately responsible for deleting the data structures holding the row cuts.

Definition at line 461 of file OsiCuts.hpp.

8.15.3.26 void OsiCuts::eraseAndDumpCuts (const std::vector< int > to_erase) [inline]

Selective delete and clear for row cuts.

Deletes the cuts specified in `to_erase` then clears remaining cuts without deleting them. A hybrid of [eraseRowCut\(int\)](#) and [dumpCuts\(\)](#). Client is ultimately responsible for deleting the data structures for row cuts not specified in `to_erase`.

Definition at line 465 of file OsiCuts.hpp.

8.15.3.27 void OsiCuts::sort () [inline]

Cuts with greatest effectiveness are first.

Definition at line 364 of file OsiCuts.hpp.

8.15.3.28 iterator OsiCuts::begin () [inline]

Get iterator to beginning of collection.

Definition at line 247 of file OsiCuts.hpp.

8.15.3.29 const_iterator OsiCuts::begin () const [inline]

Get const iterator to beginning of collection.

Definition at line 249 of file OsiCuts.hpp.

8.15.3.30 iterator OsiCuts::end () [inline]

Get iterator to end of collection.

Definition at line 251 of file OsiCuts.hpp.

8.15.3.31 const_iterator OsiCuts::end () const [inline]

Get const iterator to end of collection.

Definition at line 253 of file OsiCuts.hpp.

8.15.3.32 OsiCuts& OsiCuts::operator= (const OsiCuts & rhs)

Assignment operator.

8.15.3.33 void OsiCuts::gutsOfCopy (const OsiCuts & source) [private]

Copy internal data.

8.15.3.34 void OsiCuts::gutsOfDestructor () [private]

Delete internal data.

8.15.4 Friends And Related Function Documentation

8.15.4.1 void OsiCutsUnitTest () [friend]

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

8.15.5 Member Data Documentation

8.15.5.1 OsiVectorRowCutPtr OsiCuts::rowCutPtrs_ [private]

Vector of row cuts pointers.

Definition at line 295 of file [OsiCuts.hpp](#).

8.15.5.2 OsiVectorColCutPtr OsiCuts::colCutPtrs_ [private]

Vector of column cuts pointers.

Definition at line 297 of file [OsiCuts.hpp](#).

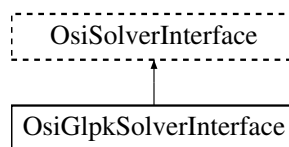
The documentation for this class was generated from the following file:

- [/home/ted/COIN/trunk/Osi/src/Osi/OsiCuts.hpp](#)

8.16 OsiGlpkSolverInterface Class Reference

```
#include <OsiGlpkSolverInterface.hpp>
```

Inheritance diagram for OsiGlpkSolverInterface:



Public Member Functions

- virtual void [setObjSense](#) (double s)
Set objective function sense (1 for min (default), -1 for max.)
- virtual void [setColSolution](#) (const double *colsol)
Set the primal solution column values.
- virtual void [setRowPrice](#) (const double *rowprice)
Set dual solution vector.

Solve methods

- virtual void [initialSolve](#) ()

- virtual void `resolve` ()
Solve initial LP relaxation.
- virtual void `branchAndBound` ()
Resolve an LP relaxation after problem modification.
- virtual void `branchAndBound` ()
Invoke solver's built-in enumeration algorithm.

Parameter set/get methods

The set methods return true if the parameter was set to the given value, false otherwise.

There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the volume algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.

The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.

- bool `setIntParam` (OsiIntParam key, int value)
Set an integer parameter.
- bool `setDbParam` (OsiDbParam key, double value)
Set a double parameter.
- bool `setStrParam` (OsiStrParam key, const std::string &value)
Set a string parameter.
- bool `setHintParam` (OsiHintParam key, bool sense=true, OsiHintStrength strength=OsiHintTry, void *info=0)
Set a hint parameter.
- bool `getIntParam` (OsiIntParam key, int &value) const
Get an integer parameter.
- bool `getDbParam` (OsiDbParam key, double &value) const
Get a double parameter.
- bool `getStrParam` (OsiStrParam key, std::string &value) const
Get a string parameter.

Methods returning info on how the solution process terminated

- virtual bool `isAbandoned` () const
Are there a numerical difficulties?
- virtual bool `isProvenOptimal` () const
Is optimality proven?
- virtual bool `isProvenPrimalInfeasible` () const
Is primal infeasibility proven?
- virtual bool `isProvenDualInfeasible` () const
Is dual infeasibility proven?
- virtual bool `isPrimalObjectiveLimitReached` () const
Is the given primal objective limit reached?
- virtual bool `isDualObjectiveLimitReached` () const
Is the given dual objective limit reached?
- virtual bool `isIterationLimitReached` () const
Iteration limit reached?
- virtual bool `isTimeLimitReached` () const
Time limit reached?
- virtual bool `isFeasible` () const
(Integer) Feasible solution found?

WarmStart related methods

- CoinWarmStart * `getEmptyWarmStart` () const

- *Get an empty warm start object.*
- virtual CoinWarmStart * [getWarmStart](#) () const
- *Get warmstarting information.*
- virtual bool [setWarmStart](#) (const CoinWarmStart *warmstart)
- *Set warmstarting information.*

Hotstart related methods (primarily used in strong branching).

The user can create a hotstart (a snapshot) of the optimization process then reoptimize over and over again always starting from there.

NOTE: between hotstarted optimizations only bound changes are allowed.

- virtual void [markHotStart](#) ()
- *Create a hotstart point of the optimization process.*
- virtual void [solveFromHotStart](#) ()
- *Optimize starting from the hotstart.*
- virtual void [unmarkHotStart](#) ()
- *Delete the snapshot.*

Methods related to querying the input data

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

Methods related to querying the solution

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

Changing bounds on variables and constraints

- virtual void [setObjCoeff](#) (int elementIndex, double elementValue)
Set an objective function coefficient.
- virtual void [setColLower](#) (int elementIndex, double elementValue)
Set a single column lower bound
Use -COIN_DBL_MAX for -infinity.
- virtual void [setColUpper](#) (int elementIndex, double elementValue)
Set a single column upper bound
Use COIN_DBL_MAX for infinity.
- virtual void [setColBounds](#) (int elementIndex, double lower, double upper)
Set a single column lower and upper bound
The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#)
- virtual void [setColSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a number of columns simultaneously
The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#) over and over again.
- virtual void [setRowLower](#) (int elementIndex, double elementValue)
Set a single row lower bound
Use -COIN_DBL_MAX for -infinity.
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)
Set a single row upper bound
Use COIN_DBL_MAX for infinity.
- virtual void [setRowBounds](#) (int elementIndex, double lower, double upper)
Set a single row lower and upper bound
The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#)
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)
Set the type of a single row
- virtual void [setRowSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a number of rows simultaneously
The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#) over and over again.
- virtual void [setRowSetTypes](#) (const int *indexFirst, const int *indexLast, const char *senseList, const double *rhsList, const double *rangeList)
Set the type of a number of rows simultaneously
The default implementation just invokes [setRowType\(\)](#) over and over again.

Integrality related changing methods

- virtual void [setContinuous](#) (int index)

- *Set the index-th variable to be a continuous variable.*
- virtual void `setInteger` (int index)
- *Set the index-th variable to be an integer variable.*
- virtual void `setContinuous` (const int *indices, int len)
- *Set the variables listed in indices (which is of length len) to be continuous variables.*
- virtual void `setInteger` (const int *indices, int len)
- *Set the variables listed in indices (which is of length len) to be integer variables.*

Methods to expand a problem.

Note that if a column is added then by default it will correspond to a continuous variable.

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

Methods to input a problem

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

Methods for row and column names.

Only the set methods need to be overridden to ensure consistent names between OsiGlpk and the OSI base class.

- void [setObjName](#) (std::string name)
Set the objective function name.
- void [setRowName](#) (int ndx, std::string name)
Set a row name.
- void [setColName](#) (int ndx, std::string name)
Set a column name.

Constructors and destructor

- [OsiGlpkSolverInterface](#) ()
Default Constructor.
- virtual [OsiSolverInterface](#) * [clone](#) (bool copyData=true) const
Clone.
- [OsiGlpkSolverInterface](#) (const [OsiGlpkSolverInterface](#) &)
Copy constructor.
- [OsiGlpkSolverInterface](#) & [operator=](#) (const [OsiGlpkSolverInterface](#) &rhs)
Assignment operator.
- virtual [~OsiGlpkSolverInterface](#) ()
Destructor.
- virtual void [reset](#) ()
Resets as if default constructor.

Static Public Member Functions

Static instance counter methods

- static void [incrementInstanceCounter](#) ()
GLPK has a context which must be freed after all GLPK LPs (or MIPs) are freed.
- static void [decrementInstanceCounter](#) ()
GLPK has a context which must be freed after all GLPK LPs (or MIPs) are freed.
- static unsigned int [getNumInstances](#) ()
Return the number of LP/MIP instances of instantiated objects using the GLPK environment.

Protected Member Functions

Protected methods

- virtual void [applyRowCut](#) (const [OsiRowCut](#) &rc)
Apply a row cut. Return true if cut was applied.
- virtual void [applyColCut](#) (const [OsiColCut](#) &cc)
Apply a column cut (bound adjustment).
- [LPX](#) * [getMutableModelPtr](#) () const
Pointer to the model.

Private Member Functions

Private methods

- void [gutsOfCopy](#) (const [OsiGlpkSolverInterface](#) &source)
The real work of a copy constructor (used by copy and assignment)
- void [gutsOfConstructor](#) ()
The real work of the constructor.
- void [gutsOfDestructor](#) ()
The real work of the destructor.
- void [freeCachedColRim](#) ()
free cached column rim vectors
- void [freeCachedRowRim](#) ()
free cached row rim vectors
- void [freeCachedResults](#) ()
free cached result vectors
- void [freeCachedMatrix](#) ()
free cached matrices
- void [freeCachedData](#) (int keepCached=[KEEPCACHED_NONE](#))
free all cached data (except specified entries, see getLpPtr())
- void [freeAllMemory](#) ()
free all allocated memory
- void [printBounds](#) ()
Just for testing purposes.
- void [fillColBounds](#) () const
Fill cached collumn bounds.

Private Attributes

Cached information derived from the GLPK model

- int [iter_used_](#)
Number of iterations.
- double * [obj_](#)
Pointer to objective vector.
- double * [collower_](#)
Pointer to dense vector of variable lower bounds.
- double * [colupper_](#)
Pointer to dense vector of variable lower bounds.
- char * [ctype_](#)
Pointer to dense vector of variable types (continous, binary, integer)
- char * [rowsense_](#)
Pointer to dense vector of row sense indicators.
- double * [rhs_](#)
Pointer to dense vector of row right-hand side values.
- double * [rowrange_](#)
Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)
- double * [rowlower_](#)
Pointer to dense vector of row lower bounds.
- double * [rowupper_](#)
Pointer to dense vector of row upper bounds.
- double * [colsol_](#)
Pointer to primal solution vector.
- double * [rowsol_](#)

- *Pointer to dual solution vector.*
- double * [redcost_](#)
Pointer to reduced cost vector.
- double * [rowact_](#)
Pointer to row activity (slack) vector.
- CoinPackedMatrix * [matrixByRow_](#)
Pointer to row-wise copy of problem matrix coefficients.
- CoinPackedMatrix * [matrixByCol_](#)
Pointer to row-wise copy of problem matrix coefficients.

Friends

- void [OsiGlpkSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiGlpkSolverInterface](#) class.

GLPK specific public interfaces

- enum [keepCachedFlag](#) {
[KEEPCACHED_NONE](#) = 0, [KEEPCACHED_COLUMN](#) = 1, [KEEPCACHED_ROW](#) = 2, [KEEPCACHED_MATRIX](#) = 4,
[KEEPCACHED_RESULTS](#) = 8, [KEEPCACHED_PROBLEM](#) = [KEEPCACHED_COLUMN](#) | [KEEPCACHED_ROW](#) | [KEEPCACHED_MATRIX](#), [KEEPCACHED_ALL](#) = [KEEPCACHED_PROBLEM](#) | [KEEPCACHED_RESULTS](#),
[FREECACHED_COLUMN](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_COLUMN](#),
[FREECACHED_ROW](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_ROW](#), [FREECACHED_MATRIX](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_MATRIX](#), [FREECACHED_RESULTS](#) = [KEEPCACHED_ALL](#) & ~[KEEPCACHED_RESULTS](#) }
- LPX * [getModelPtr](#) ()
Get pointer to GLPK model.

Private member data

- LPX * [lp_](#)
GLPK model represented by this class instance.
- int [bbWasLast_](#)
- int [maxIteration_](#)
simplex iteration limit (per call to solver)
- int [hotStartMaxIteration_](#)
simplex iteration limit (for hot start)
- int [nameDisc_](#)
OSI name discipline.
- double [dualObjectiveLimit_](#)
dual objective limit (measure of badness; stop if we're worse)
- double [primalObjectiveLimit_](#)
primal objective limit (measure of goodness; stop if we're better)
- double [dualTolerance_](#)
dual feasibility tolerance
- double [primalTolerance_](#)
primal feasibility tolerance
- double [objOffset_](#)

- constant offset for objective function*
- `std::string probName_`
Problem name.
- `void * info_ [OsiLastHintParam]`
Array for info blocks associated with hints.
- `int hotStartCStatSize_`
Hotstart information.
- `int * hotStartCStat_`
column status array
- `double * hotStartCVal_`
primal variable values
- `double * hotStartCDualVal_`
dual variable values
- `int hotStartRStatSize_`
size of row status and value arrays
- `int * hotStartRStat_`
row status array
- `double * hotStartRVal_`
row slack values
- `double * hotStartRDualVal_`
row dual values
- `bool isIterationLimitReached_`
glpk stopped on iteration limit
- `bool isTimeLimitReached_`
glpk stopped on time limit
- `bool isAbandoned_`
glpk abandoned the problem
- `bool isObjLowerLimitReached_`
glpk stopped on lower objective limit
- `bool isObjUpperLimitReached_`
glpk stopped on upper objective limit
- `bool isPrimInfeasible_`
glpk declared the problem primal infeasible
- `bool isDualInfeasible_`
glpk declared the problem dual infeasible
- `bool isFeasible_`
glpk declared the problem feasible
- `static unsigned int numInstances_`
number of GLPK instances currently in use (counts only those created by OsiGlpk)

Additional Inherited Members

8.16.1 Detailed Description

Definition at line 34 of file OsiGlpkSolverInterface.hpp.

8.16.2 Member Enumeration Documentation

8.16.2.1 enum OsiGlpkSolverInterface::keepCachedFlag

Enumerator

KEEPCACHED_NONE discard all cached data (default)
KEEPCACHED_COLUMN column information: objective values, lower and upper bounds, variable types
KEEPCACHED_ROW row information: right hand sides, ranges and senses, lower and upper bounds for row
KEEPCACHED_MATRIX problem matrix: matrix ordered by column and by row
KEEPCACHED_RESULTS LP solution: primal and dual solution, reduced costs, row activities.
KEEPCACHED_PROBLEM only discard cached LP solution
KEEPCACHED_ALL keep all cached data (similar to getMutableLpPtr())
FREECACHED_COLUMN free only cached column and LP solution information
FREECACHED_ROW free only cached row and LP solution information
FREECACHED_MATRIX free only cached matrix and LP solution information
FREECACHED_RESULTS free only cached LP solution information

Definition at line 635 of file OsiGlpkSolverInterface.hpp.

8.16.3 Constructor & Destructor Documentation

8.16.3.1 OsiGlpkSolverInterface::OsiGlpkSolverInterface ()

Default Constructor.

8.16.3.2 OsiGlpkSolverInterface::OsiGlpkSolverInterface (const OsiGlpkSolverInterface &)

Copy constructor.

8.16.3.3 virtual OsiGlpkSolverInterface::~OsiGlpkSolverInterface () [virtual]

Destructor.

8.16.4 Member Function Documentation

8.16.4.1 virtual void OsiGlpkSolverInterface::initialSolve () [virtual]

Solve initial LP relaxation.

Implements [OsiSolverInterface](#).

8.16.4.2 virtual void OsiGlpkSolverInterface::resolve () [virtual]

Resolve an LP relaxation after problem modification.

Implements [OsiSolverInterface](#).

8.16.4.3 virtual void OsiGlpkSolverInterface::branchAndBound () [virtual]

Invoke solver's built-in enumeration algorithm.

Implements [OsiSolverInterface](#).

8.16.4.4 `bool OsiGlpkSolverInterface::setIntParam (OsiIntParam key, int value) [virtual]`

Set an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.16.4.5 `bool OsiGlpkSolverInterface::setDbiParam (OsiDbiParam key, double value) [virtual]`

Set a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.16.4.6 `bool OsiGlpkSolverInterface::setStrParam (OsiStrParam key, const std::string & value) [virtual]`

Set a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.16.4.7 `bool OsiGlpkSolverInterface::setHintParam (OsiHintParam key, bool yesNo = true, OsiHintStrength strength = OsiHintTry, void * = 0) [virtual]`

Set a hint parameter.

The `otherInformation` parameter can be used to pass in an arbitrary block of information which is interpreted by the OSI and the underlying solver. Users are cautioned that this hook is solver-specific.

Implementors: The default implementation completely ignores `otherInformation` and always throws an exception for `OsiForceDo`. This is almost certainly not the behaviour you want; you really should override this method.

Reimplemented from [OsiSolverInterface](#).

8.16.4.8 `bool OsiGlpkSolverInterface::getIntParam (OsiIntParam key, int & value) const [virtual]`

Get an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.16.4.9 `bool OsiGlpkSolverInterface::getDbiParam (OsiDbiParam key, double & value) const [virtual]`

Get a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.16.4.10 `bool OsiGlpkSolverInterface::getStrParam (OsiStrParam key, std::string & value) const [virtual]`

Get a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.16.4.11 `virtual bool OsiGlpkSolverInterface::isAbandoned () const [virtual]`

Are there a numerical difficulties?

Implements [OsiSolverInterface](#).

8.16.4.12 `virtual bool OsiGlpkSolverInterface::isProvenOptimal () const [virtual]`

Is optimality proven?

Implements [OsiSolverInterface](#).

8.16.4.13 `virtual bool OsiGlpkSolverInterface::isProvenPrimalInfeasible () const [virtual]`

Is primal infeasibility proven?

Implements [OsiSolverInterface](#).

8.16.4.14 `virtual bool OsiGlpkSolverInterface::isProvenDualInfeasible () const [virtual]`

Is dual infeasibility proven?

Implements [OsiSolverInterface](#).

8.16.4.15 `virtual bool OsiGlpkSolverInterface::isPrimalObjectiveLimitReached () const [virtual]`

Is the given primal objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.16.4.16 `virtual bool OsiGlpkSolverInterface::isDualObjectiveLimitReached () const [virtual]`

Is the given dual objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.16.4.17 `virtual bool OsiGlpkSolverInterface::isIterationLimitReached () const [virtual]`

Iteration limit reached?

Implements [OsiSolverInterface](#).

8.16.4.18 `virtual bool OsiGlpkSolverInterface::isTimeLimitReached () const [virtual]`

Time limit reached?

8.16.4.19 `virtual bool OsiGlpkSolverInterface::isFeasible () const [virtual]`

(Integer) Feasible solution found?

8.16.4.20 `CoinWarmStart* OsiGlpkSolverInterface::getEmptyWarmStart () const [inline],[virtual]`

Get an empty warm start object.

This routine returns an empty `CoinWarmStartBasis` object. Its purpose is to provide a way to give a client a warm start basis object of the appropriate type, which can be resized and modified as desired.

Implements [OsiSolverInterface](#).

Definition at line 117 of file `OsiGlpkSolverInterface.hpp`.

8.16.4.21 `virtual CoinWarmStart* OsiGlpkSolverInterface::getWarmStart () const [virtual]`

Get warmstarting information.

Implements [OsiSolverInterface](#).

8.16.4.22 `virtual bool OsiGlpkSolverInterface::setWarmStart (const CoinWarmStart * warmstart) [virtual]`

Set warmstarting information.

Return true/false depending on whether the warmstart information was accepted or not.

Implements [OsiSolverInterface](#).

8.16.4.23 `virtual void OsiGlpkSolverInterface::markHotStart () [virtual]`

Create a hotstart point of the optimization process.

Reimplemented from [OsiSolverInterface](#).

8.16.4.24 `virtual void OsiGlpkSolverInterface::solveFromHotStart () [virtual]`

Optimize starting from the hotstart.

Reimplemented from [OsiSolverInterface](#).

8.16.4.25 `virtual void OsiGlpkSolverInterface::unmarkHotStart () [virtual]`

Delete the snapshot.

Reimplemented from [OsiSolverInterface](#).

8.16.4.26 `virtual int OsiGlpkSolverInterface::getNumCols () const [virtual]`

Get number of columns.

Implements [OsiSolverInterface](#).

8.16.4.27 `virtual int OsiGlpkSolverInterface::getNumRows () const [virtual]`

Get number of rows.

Implements [OsiSolverInterface](#).

8.16.4.28 `virtual int OsiGlpkSolverInterface::getNumElements () const [virtual]`

Get number of nonzero elements.

Implements [OsiSolverInterface](#).

8.16.4.29 `virtual const double* OsiGlpkSolverInterface::getColLower () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column lower bounds.

Implements [OsiSolverInterface](#).

8.16.4.30 `virtual const double* OsiGlpkSolverInterface::getColUpper () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column upper bounds.

Implements [OsiSolverInterface](#).

8.16.4.31 `virtual const char* OsiGlpkSolverInterface::getRowSense () const [virtual]`

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

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

Implements [OsiSolverInterface](#).

8.16.4.32 `virtual const double* OsiGlpkSolverInterface::getRightHandSide () const [virtual]`

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

- if `rowsense()[i] == 'L'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'G'` then `rhs()[i] == rowlower()[i]`
- if `rowsense()[i] == 'R'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'N'` then `rhs()[i] == 0.0`

Implements [OsiSolverInterface](#).

8.16.4.33 `virtual const double* OsiGlpkSolverInterface::getRowRange () const [virtual]`

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

- if `rowsense()[i] == 'R'` then `rowrange()[i] == rowupper()[i] - rowlower()[i]`
- if `rowsense()[i] != 'R'` then `rowrange()[i]` is 0.0

Implements [OsiSolverInterface](#).

8.16.4.34 `virtual const double* OsiGlpkSolverInterface::getRowLower () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.16.4.35 `virtual const double* OsiGlpkSolverInterface::getRowUpper () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.16.4.36 `virtual const double* OsiGlpkSolverInterface::getObjCoefficients () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of objective function coefficients.

Implements [OsiSolverInterface](#).

8.16.4.37 `virtual double OsiGlpkSolverInterface::getObjSense () const [virtual]`

Get objective function sense (1 for min (default), -1 for max)

Implements [OsiSolverInterface](#).

8.16.4.38 `virtual bool OsiGlpkSolverInterface::isContinuous (int colNumber) const [virtual]`

Return true if column is continuous.

Implements [OsiSolverInterface](#).

8.16.4.39 `virtual const CoinPackedMatrix* OsiGlpkSolverInterface::getMatrixByRow () const [virtual]`

Get pointer to row-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.16.4.40 `virtual const CoinPackedMatrix* OsiGlpkSolverInterface::getMatrixByCol () const [virtual]`

Get pointer to column-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.16.4.41 `virtual double OsiGlpkSolverInterface::getInfinity () const [virtual]`

Get solver's value for infinity.

Implements [OsiSolverInterface](#).

8.16.4.42 `virtual const double* OsiGlpkSolverInterface::getColSolution () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of primal solution vector.

Implements [OsiSolverInterface](#).

8.16.4.43 `virtual const double* OsiGlpkSolverInterface::getRowPrice () const [virtual]`

Get pointer to array[[getNumRows\(\)](#)] of dual prices.

Implements [OsiSolverInterface](#).

8.16.4.44 `virtual const double* OsiGlpkSolverInterface::getReducedCost () const [virtual]`

Get a pointer to array[[getNumCols\(\)](#)] of reduced costs.

Implements [OsiSolverInterface](#).

8.16.4.45 `virtual const double* OsiGlpkSolverInterface::getRowActivity () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.16.4.46 `virtual double OsiGlpkSolverInterface::getObjValue () const [virtual]`

Get objective function value.

Implements [OsiSolverInterface](#).

8.16.4.47 `virtual int OsiGlpkSolverInterface::getIterationCount () const [virtual]`

Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver).

Implements [OsiSolverInterface](#).

8.16.4.48 `virtual std::vector<double*> OsiGlpkSolverInterface::getDualRays (int maxNumRays, bool fullRay = false) const [virtual]`

Get as many dual rays as the solver can provide.

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

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.16.4.49 `virtual std::vector<double*> OsiGlpkSolverInterface::getPrimalRays (int maxNumRays) const` [virtual]

Get as many primal rays as the solver can provide.

(In case of proven dual infeasibility there should be at least one.)

The first `getNumRows()` ray components will always be associated with the row duals (as returned by `getRowPrice()`). If `fullRay` is true, the final `getNumCols()` entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

NOTE for implementers of solver interfaces:

The double pointers in the vector should point to arrays of length `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[]`.

Implements [OsiSolverInterface](#).

8.16.4.50 `virtual void OsiGlpkSolverInterface::setObjCoeff (int elementIndex, double elementValue)` [virtual]

Set an objective function coefficient.

Implements [OsiSolverInterface](#).

8.16.4.51 `virtual void OsiGlpkSolverInterface::setColLower (int elementIndex, double elementValue)` [virtual]

Set a single column lower bound

Use `-COIN_DBL_MAX` for -infinity.

Implements [OsiSolverInterface](#).

8.16.4.52 `virtual void OsiGlpkSolverInterface::setColUpper (int elementIndex, double elementValue)` [virtual]

Set a single column upper bound

Use `COIN_DBL_MAX` for infinity.

Implements [OsiSolverInterface](#).

8.16.4.53 `virtual void OsiGlpkSolverInterface::setColBounds (int elementIndex, double lower, double upper)` [virtual]

Set a single column lower and upper bound

The default implementation just invokes `setColLower()` and `setColUpper()`

Reimplemented from [OsiSolverInterface](#).

8.16.4.54 `virtual void OsiGlpkSolverInterface::setColSetBounds (const int * indexFirst, const int * indexLast, const double * boundList)` [virtual]

Set the bounds on a number of columns simultaneously

The default implementation just invokes `setColLower()` and `setColUpper()` over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the variables whose <i>either</i> bound changes
-------------------------------	--

<i>boundList</i>	the new lower/upper bound pairs for the variables
------------------	---

Reimplemented from [OsiSolverInterface](#).

8.16.4.55 `virtual void OsiGlpkSolverInterface::setRowLower (int elementIndex, double elementValue) [virtual]`

Set a single row lower bound

Use -COIN_DBL_MAX for -infinity.

Implements [OsiSolverInterface](#).

8.16.4.56 `virtual void OsiGlpkSolverInterface::setRowUpper (int elementIndex, double elementValue) [virtual]`

Set a single row upper bound

Use COIN_DBL_MAX for infinity.

Implements [OsiSolverInterface](#).

8.16.4.57 `virtual void OsiGlpkSolverInterface::setRowBounds (int elementIndex, double lower, double upper) [virtual]`

Set a single row lower and upper bound

The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#)

Reimplemented from [OsiSolverInterface](#).

8.16.4.58 `virtual void OsiGlpkSolverInterface::setRowType (int index, char sense, double rightHandSide, double range) [virtual]`

Set the type of a single row

Implements [OsiSolverInterface](#).

8.16.4.59 `virtual void OsiGlpkSolverInterface::setRowSetBounds (const int * indexFirst, const int * indexLast, const double * boundList) [virtual]`

Set the bounds on a number of rows simultaneously

The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#) over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the constraints whose <i>either</i> bound changes
<i>boundList</i>	the new lower/upper bound pairs for the constraints

Reimplemented from [OsiSolverInterface](#).

8.16.4.60 `virtual void OsiGlpkSolverInterface::setRowSetTypes (const int * indexFirst, const int * indexLast, const char * senseList, const double * rhsList, const double * rangeList) [virtual]`

Set the type of a number of rows simultaneously

The default implementation just invokes [setRowType\(\)](#) over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the constraints whose any characteristics changes
<i>senseList</i>	the new senses
<i>rhsList</i>	the new right hand sides
<i>rangeList</i>	the new ranges

Reimplemented from [OsiSolverInterface](#).

8.16.4.61 `virtual void OsiGlpkSolverInterface::setContinuous (int index) [virtual]`

Set the index-th variable to be a continuous variable.

Implements [OsiSolverInterface](#).

8.16.4.62 `virtual void OsiGlpkSolverInterface::setInteger (int index) [virtual]`

Set the index-th variable to be an integer variable.

Implements [OsiSolverInterface](#).

8.16.4.63 `virtual void OsiGlpkSolverInterface::setContinuous (const int * indices, int len) [virtual]`

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

Reimplemented from [OsiSolverInterface](#).

8.16.4.64 `virtual void OsiGlpkSolverInterface::setInteger (const int * indices, int len) [virtual]`

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

Reimplemented from [OsiSolverInterface](#).

8.16.4.65 `virtual void OsiGlpkSolverInterface::setObjSense (double s) [virtual]`

Set objective function sense (1 for min (default), -1 for max,)

Implements [OsiSolverInterface](#).

8.16.4.66 `virtual void OsiGlpkSolverInterface::setColSolution (const double * colsol) [virtual]`

Set the primal solution column values.

`colsol[numcols()]` is an array of values of the problem column variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of `colsol()` until changed by another call to `setColsol()` or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

Implements [OsiSolverInterface](#).

8.16.4.67 `virtual void OsiGlpkSolverInterface::setRowPrice (const double * rowprice) [virtual]`

Set dual solution vector.

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

Implements [OsiSolverInterface](#).

8.16.4.68 `virtual void OsiGlpkSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [virtual]`

Add a column (primal variable) to the problem.

Implements [OsiSolverInterface](#).

8.16.4.69 `virtual void OsiGlpkSolverInterface::addCols (const int numcols, const CoinPackedVectorBase *const * cols, const double * collb, const double * colub, const double * obj) [virtual]`

Add a set of columns (primal variables) to the problem.

The default implementation simply makes repeated calls to [addCol\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.16.4.70 `virtual void OsiGlpkSolverInterface::deleteCols (const int num, const int * colIndices) [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](#).

8.16.4.71 `virtual void OsiGlpkSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.16.4.72 `virtual void OsiGlpkSolverInterface::addRow (const CoinPackedVectorBase & vec, const char rowSEN, const double rowrhs, const double rowrng) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.16.4.73 `virtual void OsiGlpkSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const double * rowlb, const double * rowub) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.16.4.74 `virtual void OsiGlpkSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const char * rowSEN, const double * rowrhs, const double * rowrng) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.16.4.75 `virtual void OsiGlpkSolverInterface::deleteRows (const int num, const int * rowIndices) [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](#).

8.16.4.76 `virtual void OsiGlpkSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub) [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](#).

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

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

For default values see the previous method.

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

Implements [OsiSolverInterface](#).

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

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

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

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

Implements [OsiSolverInterface](#).

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

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

For default values see the previous method.

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

Implements [OsiSolverInterface](#).

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

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

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

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

8.16.4.82 `virtual int OsiGlpkSolverInterface::readMps (const char * filename, const char * extension = "mps")` [virtual]

Read an mps file from the given filename.

Reimplemented from [OsiSolverInterface](#).

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

Write the problem into an mps file of the given filename.

If objSense is non zero then -1.0 forces the code to write a maximization objective and +1.0 to write a minimization one. If 0.0 then solver can do what it wants

Implements [OsiSolverInterface](#).

8.16.4.84 `void OsiGlpkSolverInterface::setObjName (std::string name)` [virtual]

Set the objective function name.

Reimplemented from [OsiSolverInterface](#).

8.16.4.85 `void OsiGlpkSolverInterface::setRowName (int ndx, std::string name)` [virtual]

Set a row name.

Quietly does nothing if the name discipline ([OsiNameDiscipline](#)) is auto. Quietly fails if the row index is invalid.

Reimplemented from [OsiSolverInterface](#).

8.16.4.86 `void OsiGlpkSolverInterface::setColName (int ndx, std::string name)` [virtual]

Set a column name.

Quietly does nothing if the name discipline ([OsiNameDiscipline](#)) is auto. Quietly fails if the column index is invalid.

Reimplemented from [OsiSolverInterface](#).

8.16.4.87 `LPX* OsiGlpkSolverInterface::getModelPtr ()`

Get pointer to GLPK model.

8.16.4.88 `static void OsiGlpkSolverInterface::incrementInstanceCounter ()` [inline],[static]

GLPK has a context which must be freed after all GLPK LPs (or MIPs) are freed.

It is automatically created when the first LP is created. This method:

- Increments by 1 the number of uses of the GLPK environment.

Definition at line 674 of file OsiGlpkSolverInterface.hpp.

8.16.4.89 `static void OsiGlpkSolverInterface::decrementInstanceCounter () [static]`

GLPK has a context which must be freed after all GLPK LPs (or MIPs) are freed.

This method:

- Decrements by 1 the number of uses of the GLPK environment.
- Deletes the GLPK environment when the number of uses is change to 0 from 1.

8.16.4.90 `static unsigned int OsiGlpkSolverInterface::getNumInstances () [inline],[static]`

Return the number of LP/MIP instances of instantiated objects using the GLPK environment.

Definition at line 686 of file OsiGlpkSolverInterface.hpp.

8.16.4.91 `virtual OsiSolverInterface* OsiGlpkSolverInterface::clone (bool copyData = true) const [virtual]`

Clone.

Implements [OsiSolverInterface](#).

8.16.4.92 `OsiGlpkSolverInterface& OsiGlpkSolverInterface::operator= (const OsiGlpkSolverInterface & rhs)`

Assignment operator.

8.16.4.93 `virtual void OsiGlpkSolverInterface::reset () [virtual]`

Resets as if default constructor.

Reimplemented from [OsiSolverInterface](#).

8.16.4.94 `virtual void OsiGlpkSolverInterface::applyRowCut (const OsiRowCut & rc) [protected],[virtual]`

Apply a row cut. Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.16.4.95 `virtual void OsiGlpkSolverInterface::applyColCut (const OsiColCut & cc) [protected],[virtual]`

Apply a column cut (bound adjustment).

Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.16.4.96 `LPX* OsiGlpkSolverInterface::getMutableModelPtr () const [protected]`

Pointer to the model.

8.16.4.97 `void OsiGlpkSolverInterface::gutsOfCopy (const OsiGlpkSolverInterface & source) [private]`

The real work of a copy constructor (used by copy and assignment)

8.16.4.98 `void OsiGlpkSolverInterface::gutsOfConstructor () [private]`

The real work of the constructor.

8.16.4.99 void OsiGlpkSolverInterface::gutsOfDestructor () [private]

The real work of the destructor.

8.16.4.100 void OsiGlpkSolverInterface::freeCachedColRim () [private]

free cached column rim vectors

8.16.4.101 void OsiGlpkSolverInterface::freeCachedRowRim () [private]

free cached row rim vectors

8.16.4.102 void OsiGlpkSolverInterface::freeCachedResults () [private]

free cached result vectors

8.16.4.103 void OsiGlpkSolverInterface::freeCachedMatrix () [private]

free cached matrices

8.16.4.104 void OsiGlpkSolverInterface::freeCachedData (int *keepCached* = **KEEPCACHED_NONE**) [private]

free all cached data (except specified entries, see getLpPtr())

8.16.4.105 void OsiGlpkSolverInterface::freeAllMemory () [private]

free all allocated memory

8.16.4.106 void OsiGlpkSolverInterface::printBounds () [private]

Just for testing purposes.

8.16.4.107 void OsiGlpkSolverInterface::fillColBounds () const [private]

Fill cached column bounds.

8.16.5 Friends And Related Function Documentation

8.16.5.1 void OsiGlpkSolverInterfaceUnitTest (const std::string & *mpsDir*, const std::string & *netlibDir*) [friend]

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

8.16.6 Member Data Documentation

8.16.6.1 LPX* OsiGlpkSolverInterface::lp_ [mutable], [private]

GLPK model represented by this class instance.

Definition at line 770 of file OsiGlpkSolverInterface.hpp.

8.16.6.2 unsigned int OsiGlpkSolverInterface::numInstances_ [static], [private]

number of GLPK instances currently in use (counts only those created by OsiGlpk)

Definition at line 773 of file OsiGlpkSolverInterface.hpp.

8.16.6.3 `int OsiGlpkSolverInterface::bbWasLast_ [private]`

Definition at line 778 of file OsiGlpkSolverInterface.hpp.

8.16.6.4 `int OsiGlpkSolverInterface::maxIteration_ [private]`

simplex iteration limit (per call to solver)

Definition at line 782 of file OsiGlpkSolverInterface.hpp.

8.16.6.5 `int OsiGlpkSolverInterface::hotStartMaxIteration_ [private]`

simplex iteration limit (for hot start)

Definition at line 784 of file OsiGlpkSolverInterface.hpp.

8.16.6.6 `int OsiGlpkSolverInterface::nameDisc_ [private]`

OSI name discipline.

Definition at line 786 of file OsiGlpkSolverInterface.hpp.

8.16.6.7 `double OsiGlpkSolverInterface::dualObjectiveLimit_ [private]`

dual objective limit (measure of badness; stop if we're worse)

Definition at line 790 of file OsiGlpkSolverInterface.hpp.

8.16.6.8 `double OsiGlpkSolverInterface::primalObjectiveLimit_ [private]`

primal objective limit (measure of goodness; stop if we're better)

Definition at line 792 of file OsiGlpkSolverInterface.hpp.

8.16.6.9 `double OsiGlpkSolverInterface::dualTolerance_ [private]`

dual feasibility tolerance

Definition at line 794 of file OsiGlpkSolverInterface.hpp.

8.16.6.10 `double OsiGlpkSolverInterface::primalTolerance_ [private]`

primal feasibility tolerance

Definition at line 796 of file OsiGlpkSolverInterface.hpp.

8.16.6.11 `double OsiGlpkSolverInterface::objOffset_ [private]`

constant offset for objective function

Definition at line 798 of file OsiGlpkSolverInterface.hpp.

8.16.6.12 `std::string OsiGlpkSolverInterface::probName_ [private]`

Problem name.

Definition at line 802 of file OsiGlpkSolverInterface.hpp.

8.16.6.13 `void* OsiGlpkSolverInterface::info_[OsiLastHintParam] [mutable], [private]`

Array for info blocks associated with hints.

Definition at line 805 of file OsiGlpkSolverInterface.hpp.

8.16.6.14 `int OsiGlpkSolverInterface::hotStartCStatSize_ [private]`

Hotstart information.

size of column status and value arrays

Definition at line 811 of file OsiGlpkSolverInterface.hpp.

8.16.6.15 `int* OsiGlpkSolverInterface::hotStartCStat_ [private]`

column status array

Definition at line 813 of file OsiGlpkSolverInterface.hpp.

8.16.6.16 `double* OsiGlpkSolverInterface::hotStartCVal_ [private]`

primal variable values

Definition at line 815 of file OsiGlpkSolverInterface.hpp.

8.16.6.17 `double* OsiGlpkSolverInterface::hotStartCDualVal_ [private]`

dual variable values

Definition at line 817 of file OsiGlpkSolverInterface.hpp.

8.16.6.18 `int OsiGlpkSolverInterface::hotStartRStatSize_ [private]`

size of row status and value arrays

Definition at line 820 of file OsiGlpkSolverInterface.hpp.

8.16.6.19 `int* OsiGlpkSolverInterface::hotStartRStat_ [private]`

row status array

Definition at line 822 of file OsiGlpkSolverInterface.hpp.

8.16.6.20 `double* OsiGlpkSolverInterface::hotStartRVal_ [private]`

row slack values

Definition at line 824 of file OsiGlpkSolverInterface.hpp.

8.16.6.21 `double* OsiGlpkSolverInterface::hotStartRDualVal_ [private]`

row dual values

Definition at line 826 of file OsiGlpkSolverInterface.hpp.

8.16.6.22 `bool OsiGlpkSolverInterface::isIterationLimitReached_ [private]`

glpk stopped on iteration limit

Definition at line 830 of file OsiGlpkSolverInterface.hpp.

8.16.6.23 `bool OsiGlpkSolverInterface::isTimeLimitReached_ [private]`

glpk stopped on time limit

Definition at line 832 of file OsiGlpkSolverInterface.hpp.

8.16.6.24 `bool OsiGlpkSolverInterface::isAbandoned_ [private]`

glpk abandoned the problem

Definition at line 834 of file OsiGlpkSolverInterface.hpp.

8.16.6.25 `bool OsiGlpkSolverInterface::isObjLowerLimitReached_ [private]`

glpk stopped on lower objective limit

When minimising, this is the primal limit; when maximising, the dual limit.

Definition at line 840 of file OsiGlpkSolverInterface.hpp.

8.16.6.26 `bool OsiGlpkSolverInterface::isObjUpperLimitReached_ [private]`

glpk stopped on upper objective limit

When minimising, this is the dual limit; when maximising, the primal limit.

Definition at line 846 of file OsiGlpkSolverInterface.hpp.

8.16.6.27 `bool OsiGlpkSolverInterface::isPrimInfeasible_ [private]`

glpk declared the problem primal infeasible

Definition at line 848 of file OsiGlpkSolverInterface.hpp.

8.16.6.28 `bool OsiGlpkSolverInterface::isDualInfeasible_ [private]`

glpk declared the problem dual infeasible

Definition at line 850 of file OsiGlpkSolverInterface.hpp.

8.16.6.29 `bool OsiGlpkSolverInterface::isFeasible_ [private]`

glpk declared the problem feasible

Definition at line 852 of file OsiGlpkSolverInterface.hpp.

8.16.6.30 `int OsiGlpkSolverInterface::iter_used_ [mutable], [private]`

Number of iterations.

Definition at line 858 of file OsiGlpkSolverInterface.hpp.

8.16.6.31 `double* OsiGlpkSolverInterface::obj_ [mutable], [private]`

Pointer to objective vector.

Definition at line 861 of file OsiGlpkSolverInterface.hpp.

8.16.6.32 `double* OsiGlpkSolverInterface::collower_ [mutable], [private]`

Pointer to dense vector of variable lower bounds.

Definition at line 864 of file OsiGlpkSolverInterface.hpp.

8.16.6.33 `double* OsiGlpkSolverInterface::colupper_ [mutable], [private]`

Pointer to dense vector of variable upper bounds.

Definition at line 867 of file OsiGlpkSolverInterface.hpp.

8.16.6.34 `char* OsiGlpkSolverInterface::ctype_` [mutable],[private]

Pointer to dense vector of variable types (continous, binary, integer)

Definition at line 870 of file OsiGlpkSolverInterface.hpp.

8.16.6.35 `char* OsiGlpkSolverInterface::rowsense_` [mutable],[private]

Pointer to dense vector of row sense indicators.

Definition at line 873 of file OsiGlpkSolverInterface.hpp.

8.16.6.36 `double* OsiGlpkSolverInterface::rhs_` [mutable],[private]

Pointer to dense vector of row right-hand side values.

Definition at line 876 of file OsiGlpkSolverInterface.hpp.

8.16.6.37 `double* OsiGlpkSolverInterface::rowrange_` [mutable],[private]

Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)

Definition at line 879 of file OsiGlpkSolverInterface.hpp.

8.16.6.38 `double* OsiGlpkSolverInterface::rowlower_` [mutable],[private]

Pointer to dense vector of row lower bounds.

Definition at line 882 of file OsiGlpkSolverInterface.hpp.

8.16.6.39 `double* OsiGlpkSolverInterface::rowupper_` [mutable],[private]

Pointer to dense vector of row upper bounds.

Definition at line 885 of file OsiGlpkSolverInterface.hpp.

8.16.6.40 `double* OsiGlpkSolverInterface::colsol_` [mutable],[private]

Pointer to primal solution vector.

Definition at line 888 of file OsiGlpkSolverInterface.hpp.

8.16.6.41 `double* OsiGlpkSolverInterface::rowsol_` [mutable],[private]

Pointer to dual solution vector.

Definition at line 891 of file OsiGlpkSolverInterface.hpp.

8.16.6.42 `double* OsiGlpkSolverInterface::redcost_` [mutable],[private]

Pointer to reduced cost vector.

Definition at line 894 of file OsiGlpkSolverInterface.hpp.

8.16.6.43 `double* OsiGlpkSolverInterface::rowact_` [mutable],[private]

Pointer to row activity (slack) vector.

Definition at line 897 of file OsiGlpkSolverInterface.hpp.

8.16.6.44 `CoinPackedMatrix* OsiGlpkSolverInterface::matrixByRow_ [mutable],[private]`

Pointer to row-wise copy of problem matrix coefficients.

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

8.16.6.45 `CoinPackedMatrix* OsiGlpkSolverInterface::matrixByCol_ [mutable],[private]`

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 903 of file `OsiGlpkSolverInterface.hpp`.

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

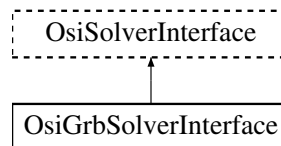
- </home/ted/COIN/trunk/Osi/src/OsiGlpk/OsiGlpkSolverInterface.hpp>

8.17 OsiGrbSolverInterface Class Reference

Gurobi Solver Interface.

```
#include <OsiGrbSolverInterface.hpp>
```

Inheritance diagram for `OsiGrbSolverInterface`:



Public Member Functions

- virtual void `setObjSense` (double s)
Set objective function sense (1 for min (default), -1 for max,)
- virtual void `setColSolution` (const double *colsol)
Set the primal solution column values.
- virtual void `setRowPrice` (const double *rowprice)
Set dual solution vector.
- const char * `getCtype` () const
return a vector of variable types (continous, binary, integer)
- virtual `OsiSolverInterface::ApplyCutsReturnCode` `applyCuts` (const `OsiCuts` &cs, double effectivenessLb=0.0)
Apply a collection of cuts.

Solve methods

- virtual void `initialSolve` ()
Solve initial LP relaxation.
- virtual void `resolve` ()
Resolve an LP relaxation after problem modification.
- virtual void `branchAndBound` ()
Invoke solver's built-in enumeration algorithm.

Parameter set/get methods

The set methods return true if the parameter was set to the given value, false otherwise.

There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the volume algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.

The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.

- bool [setIntParam](#) ([OsiIntParam](#) key, int value)
Set an integer parameter.
- bool [setDbParam](#) ([OsiDbParam](#) key, double value)
Set a double parameter.
- bool [setStrParam](#) ([OsiStrParam](#) key, const std::string &value)
Set a string parameter.
- bool [setHintParam](#) ([OsiHintParam](#) key, bool yesNo=true, [OsiHintStrength](#) strength=[OsiHintTry](#), void *!=NULL)
Set a hint parameter.
- bool [getIntParam](#) ([OsiIntParam](#) key, int &value) const
Get an integer parameter.
- bool [getDbParam](#) ([OsiDbParam](#) key, double &value) const
Get a double parameter.
- bool [getStrParam](#) ([OsiStrParam](#) key, std::string &value) const
Get a string parameter.
- bool [getHintParam](#) ([OsiHintParam](#) key, bool &yesNo, [OsiHintStrength](#) &strength, void *&otherInformation) const
Get a hint parameter (all information)
- bool [getHintParam](#) ([OsiHintParam](#) key, bool &yesNo, [OsiHintStrength](#) &strength) const
Get a hint parameter (sense and strength only)
- bool [getHintParam](#) ([OsiHintParam](#) key, bool &yesNo) const
Get a hint parameter (sense only)
- void [setMipStart](#) (bool value)
- bool [getMipStart](#) () const

Methods returning info on how the solution process terminated

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

WarmStart related methods

- CoinWarmStart * [getEmptyWarmStart](#) () const
Get an empty warm start object.

- virtual CoinWarmStart * [getWarmStart](#) () const
Get warmstarting information.
- virtual bool [setWarmStart](#) (const CoinWarmStart *warmstart)
Set warmstarting information.

Hotstart related methods (primarily used in strong branching).

The user can create a hotstart (a snapshot) of the optimization process then reoptimize over and over again always starting from there.

NOTE: between hotstarted optimizations only bound changes are allowed.

- virtual void [markHotStart](#) ()
Create a hotstart point of the optimization process.
- virtual void [solveFromHotStart](#) ()
Optimize starting from the hotstart.
- virtual void [unmarkHotStart](#) ()
Delete the snapshot.

Methods related to querying the input data

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

Methods related to querying the solution

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

Changing bounds on variables and constraints

- virtual void [setObjCoeff](#) (int elementIndex, double elementValue)
Set an objective function coefficient.
- virtual void [setObjCoeffSet](#) (const int *indexFirst, const int *indexLast, const double *coeffList)
Set a a set of objective function coefficients.
- virtual void [setColLower](#) (int elementIndex, double elementValue)
*Set a single column lower bound
Use -COIN_DBL_MAX for -infinity.*
- virtual void [setColUpper](#) (int elementIndex, double elementValue)
*Set a single column upper bound
Use COIN_DBL_MAX for infinity.*
- virtual void [setColBounds](#) (int elementIndex, double lower, double upper)
*Set a single column lower and upper bound
The default implementation just invokes [setColLower](#) () and [setColUpper](#) ()*
- virtual void [setColSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
*Set the bounds on a number of columns simultaneously
The default implementation just invokes [setColLower](#) () and [setColUpper](#) () over and over again.*
- virtual void [setRowLower](#) (int elementIndex, double elementValue)
*Set a single row lower bound
Use -COIN_DBL_MAX for -infinity.*
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)
*Set a single row upper bound
Use COIN_DBL_MAX for infinity.*
- virtual void [setRowBounds](#) (int elementIndex, double lower, double upper)
*Set a single row lower and upper bound
The default implementation just invokes [setRowLower](#) () and [setRowUpper](#) ()*
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)
Set the type of a single row
- virtual void [setRowSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
*Set the bounds on a number of rows simultaneously
The default implementation just invokes [setRowLower](#) () and [setRowUpper](#) () over and over again.*
- virtual void [setRowSetTypes](#) (const int *indexFirst, const int *indexLast, const char *senseList, const double *rhsList, const double *rangeList)
*Set the type of a number of rows simultaneously
The default implementation just invokes [setRowType](#) () and over and over again.*

Integrality related changing methods

- virtual void `setContinuous` (int index)
Set the index-th variable to be a continuous variable.
- virtual void `setInteger` (int index)
Set the index-th variable to be an integer variable.
- virtual void `setContinuous` (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be continuous variables.
- virtual void `setInteger` (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be integer variables.

Naming methods

- virtual void `setRowName` (int ndx, std::string name)
Set a row name.
- virtual void `setColName` (int ndx, std::string name)
Set a column name.

Methods to expand a problem.

Note that if a column is added then by default it will correspond to a continuous variable.

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

Methods to input a problem

- virtual void `loadProblem` (const CoinPackedMatrix &matrix, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub)
Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).
- virtual void `assignProblem` (CoinPackedMatrix *&matrix, double *&collb, double *&colub, double *&obj, double *&rowlb, double *&rowub)
Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).
- virtual void `loadProblem` (const CoinPackedMatrix &matrix, const double *collb, const double *colub, const double *obj, const char *rowsen, const double *rowrhs, const double *rowrng)
Load in an problem by copying the arguments (the constraints on the rows are given by sense/rhs/range triplets).

- virtual void [assignProblem](#) (CoinPackedMatrix *&matrix, double *&collb, double *&colub, double *&obj, char *&rowlsen, double *&rowrhs, double *&rowrng)
Load in a 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 int *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 int *start, const int *index, const double *value, const double *collb, const double *colub, const double *obj, const char *rowlsen, const double *rowrhs, const double *rowrng)
Just like the other [loadProblem\(\)](#) methods except that the matrix is given in a standard column major ordered format (without gaps).
- virtual int [readMps](#) (const char *filename, const char *extension="mps")
Read an mps file from the given filename.
- virtual void [writeMps](#) (const char *filename, const char *extension="mps", double objSense=0.0) const
Write the problem into an mps file of the given filename.

Constructors and destructor

- [OsiGrbSolverInterface](#) (bool use_local_env=false)
Default Constructor.
- [OsiGrbSolverInterface](#) (GRBEnv *localgrbenv)
Constructor that takes a gurobi environment and assumes membership.
- virtual [OsiSolverInterface](#) * [clone](#) (bool copyData=true) const
Clone.
- [OsiGrbSolverInterface](#) (const [OsiGrbSolverInterface](#) &)
Copy constructor.
- [OsiGrbSolverInterface](#) & [operator=](#) (const [OsiGrbSolverInterface](#) &rhs)
Assignment operator.
- virtual [~OsiGrbSolverInterface](#) ()
Destructor.
- virtual void [reset](#) ()
Resets as if default constructor.

OsiSimplexInterface methods

Gurobi adds a slack with coeff +1 in "<=" and "=" constraints, with coeff -1 in ">=", slack being non negative.

We switch in order to get a "Clp tableau" where all the slacks have coefficient +1 in the original tableau.

If a slack for ">=" is non basic, invB is not changed; column of the slack in the optimal tableau is flipped.

If a slack for ">=" is basic, corresp. row of invB is flipped; whole row of the optimal tableau is flipped; then whole column for the slack in opt tableau is flipped.

Ranged rows are not supported. It might work, but no guarantee is given.

- virtual int [canDoSimplexInterface](#) () const
Returns 1 if can just do getBlnc etc 2 if has all OsiSimplex methods and 0 if it has none.
- virtual void [enableSimplexInterface](#) (int doingPrimal)
Useless function, defined only for compatibility with OsiSimplexInterface.
- virtual void [disableSimplexInterface](#) ()
Useless function, defined only for compatibility with OsiSimplexInterface.
- virtual void [enableFactorization](#) () const
Useless function, defined only for compatibility with OsiSimplexInterface.
- virtual void [disableFactorization](#) () const
Useless function, defined only for compatibility with OsiSimplexInterface.

- virtual bool [basisIsAvailable](#) () const
Returns true if a basis is available.
- virtual void [getBasisStatus](#) (int *cstat, int *rstat) const
Returns a basis status of the structural/artificial variables At present as warm start i.e 0: free, 1: basic, 2: upper, 3: lower.
- void [switchToLP](#) ()
*Get indices of the pivot variable in each row
(order of indices corresponds to the order of elements in a vector returned by [getBlvAcol\(\)](#) and [getBlvCol\(\)](#)).*
- void [switchToMIP](#) ()
switches Gurobi to prob type MIP

Static Public Member Functions

Static instance counter methods

- static void [incrementInstanceCounter](#) ()
Gurobi has a context which must be created prior to all other Gurobi calls.
- static void [decrementInstanceCounter](#) ()
Gurobi has a context which should be deleted after Gurobi calls.
- static void [setEnvironment](#) (GRBEnv *globalenv)
sets the global gurobi environment to a user given one
- static unsigned int [getNumInstances](#) ()
Return the number of instances of instantiated objects using Gurobi services.

Protected Member Functions

Protected methods

- virtual void [applyRowCut](#) (const [OsiRowCut](#) &rc)
Apply a row cut. Return true if cut was applied.
- virtual void [applyColCut](#) (const [OsiColCut](#) &cc)
Apply a column cut (bound adjustment).

Private Member Functions

Private static class functions

- void [resizeColSpace](#) (int minsize)
resizes coltype_, colmap_O2G, colmap_G2O vectors to be able to store at least minsize elements
- void [freeColSpace](#) ()
frees colsize_ vector
- void [resizeAuxColSpace](#) (int minsize)
resizes colmap_G2O vector to be able to store at least minsize (auxiliary) elements
- void [resizeAuxColIndSpace](#) ()
resizes auxcolind vector to current number of rows and inits values to -1

Private methods

- [GRBmodel](#) * [getMutableLpPtr](#) () const
Get LP Pointer for const methods.
- void [gutsOfCopy](#) (const [OsiGrbSolverInterface](#) &source)
The real work of a copy constructor (used by copy and assignment)
- void [gutsOfConstructor](#) ()

- *The real work of the constructor.*
- void [gutsOfDestructor](#) ()
- *The real work of the destructor.*
- void [freeCachedColRim](#) ()
- *free cached column rim vectors*
- void [freeCachedRowRim](#) ()
- *free cached row rim vectors*
- void [freeCachedResults](#) ()
- *free cached result vectors*
- void [freeCachedMatrix](#) ()
- *free cached matrices*
- void [freeCachedData](#) (int keepCached=[KEEPCACHED_NONE](#))
- *free all cached data (except specified entries, see [getLpPtr\(\)](#))*
- void [freeAllMemory](#) ()
- *free all allocated memory*
- void [convertToRangedRow](#) (int rowidx, double rhs, double range)
- *converts a normal row into a ranged row by adding an auxiliary variable*
- void [convertToNormalRow](#) (int rowidx, char sense, double rhs)
- *converts a ranged row into a normal row by removing its auxiliary variable*

Private Attributes

Private member data

- [GRBEnv](#) * [localenv_](#)
- *Gurobi environment used only by this class instance.*
- [GRBmodel](#) * [lp_](#)
- *Gurobi model represented by this class instance.*
- int * [hotStartCStat_](#)
- *Hotstart information.*
- int [hotStartCStatSize_](#)
- int * [hotStartRStat_](#)
- int [hotStartRStatSize_](#)
- int [hotStartMaxIteration_](#)
- int [nameDisc_](#)
- *OSI name discipline.*

Cached information derived from the Gurobi model

- double * [obj_](#)
- *Pointer to objective vector.*
- double * [collower_](#)
- *Pointer to dense vector of variable lower bounds.*
- double * [colupper_](#)
- *Pointer to dense vector of variable upper bounds.*
- char * [rowsense_](#)
- *Pointer to dense vector of row sense indicators.*
- double * [rhs_](#)
- *Pointer to dense vector of row right-hand side values.*
- double * [rowrange_](#)
- *Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)*
- double * [rowlower_](#)
- *Pointer to dense vector of row lower bounds.*
- double * [rowupper_](#)

- *Pointer to dense vector of row upper bounds.*
- double * [colsol_](#)
Pointer to primal solution vector.
- double * [rowsol_](#)
Pointer to dual solution vector.
- double * [redcost_](#)
Pointer to reduced cost vector.
- double * [rowact_](#)
Pointer to row activity (slack) vector.
- CoinPackedMatrix * [matrixByRow_](#)
Pointer to row-wise copy of problem matrix coefficients.
- CoinPackedMatrix * [matrixByCol_](#)
Pointer to row-wise copy of problem matrix coefficients.

Additional information needed for storing MIP problems and handling ranged rows

- bool [probtymip_](#)
Stores whether we currently see the problem as a MIP.
- bool [domipstart](#)
Whether to pass a column solution to CPLEX before starting MIP solve (copymipstart)
- int [colspace_](#)
Size of allocated memory for coltype_, colmap_O2G, and (with offset auxcolspace) colmap_G2O.
- char * [coltype_](#)
Pointer to dense vector of variable types (continuous, binary, integer)
- int [nauxcols](#)
Number of auxiliary columns in Gurobi model for handling of ranged rows.
- int [auxcolspace](#)
Size of allocated memory for colmap_G2O that exceeds colspace_.
- int * [colmap_O2G](#)
Maps variable indices from Osi to Gurobi Is NULL if there are no ranged rows! (assume identity mapping then)
- int * [colmap_G2O](#)
Maps variable indices from Gurobi to Osi A negative value indicates that a variable is an auxiliary variable that was added to handle a ranged row -colmap_G2O[i]-1 gives the index of the ranged row in this case.
- int [auxcolindspace](#)
Current length of auxcolind array.
- int * [auxcolind](#)
Gives for each row the index of the corresponding auxiliary variable, if it is a ranged row.

Static Private Attributes

Private static class data

- static [GRBenv](#) * [globalenv_](#)
Gurobi environment pointer.
- static bool [globalenv_is_ours](#)
whether OsiGrb has created the global environment (and thus should free it)
- static unsigned int [numInstances_](#)
Number of instances using the global Gurobi environment.

Friends

- void [OsiGrbSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiGrbSolverInterface](#) class.

Gurobi specific public interfaces

- enum `keepCachedFlag` {
`KEEPCACHED_NONE` = 0, `KEEPCACHED_COLUMN` = 1, `KEEPCACHED_ROW` = 2, `KEEPCACHED_MATRIX` = 4,
`KEEPCACHED_RESULTS` = 8, `KEEPCACHED_PROBLEM` = `KEEPCACHED_COLUMN` | `KEEPCACHED_ROW` | `KEEPCACHED_MATRIX`, `KEEPCACHED_ALL` = `KEEPCACHED_PROBLEM` | `KEEPCACHED_RESULTS`,
`FREECACHED_COLUMN` = `KEEPCACHED_PROBLEM` & ~`KEEPCACHED_COLUMN`,
`FREECACHED_ROW` = `KEEPCACHED_PROBLEM` & ~`KEEPCACHED_ROW`, `FREECACHED_MATRIX` = `KEEPCACHED_PROBLEM` & ~`KEEPCACHED_MATRIX`, `FREECACHED_RESULTS` = `KEEPCACHED_ALL` & ~`KEEPCACHED_RESULTS` }

Get pointer to Gurobi model and free all specified cached data entries (combined with logical or-operator '|'):

- `GRBmodel * getLpPtr` (int keepCached=`KEEPCACHED_NONE`)
- `GRBenv * getEnvironmentPtr` () const

Method to access Gurobi environment pointer.

- bool `isDemoLicense` () const

Return whether the current Gurobi environment runs in demo mode.

Additional Inherited Members

8.17.1 Detailed Description

Gurobi Solver Interface.

Instantiation of `OsiGrbSolverInterface` for Gurobi

Definition at line 29 of file `OsiGrbSolverInterface.hpp`.

8.17.2 Member Enumeration Documentation

8.17.2.1 enum `OsiGrbSolverInterface::keepCachedFlag`

Get pointer to Gurobi model and free all specified cached data entries (combined with logical or-operator '|'):

Enumerator

- `KEEPCACHED_NONE`** discard all cached data (default)
- `KEEPCACHED_COLUMN`** column information: objective values, lower and upper bounds, variable types
- `KEEPCACHED_ROW`** row information: right hand sides, ranges and senses, lower and upper bounds for row
- `KEEPCACHED_MATRIX`** problem matrix: matrix ordered by column and by row
- `KEEPCACHED_RESULTS`** LP solution: primal and dual solution, reduced costs, row activities.
- `KEEPCACHED_PROBLEM`** only discard cached LP solution
- `KEEPCACHED_ALL`** keep all cached data (similar to `getMutableLpPtr()`)
- `FREECACHED_COLUMN`** free only cached column and LP solution information
- `FREECACHED_ROW`** free only cached row and LP solution information
- `FREECACHED_MATRIX`** free only cached matrix and LP solution information
- `FREECACHED_RESULTS`** free only cached LP solution information

Definition at line 546 of file `OsiGrbSolverInterface.hpp`.

8.17.3 Constructor & Destructor Documentation

8.17.3.1 OsiGrbSolverInterface::OsiGrbSolverInterface (bool *use_local_env* = false)

Default Constructor.

8.17.3.2 OsiGrbSolverInterface::OsiGrbSolverInterface (GRBEnv * *localgrbenv*)

Constructor that takes a gurobi environment and assumes membership.

8.17.3.3 OsiGrbSolverInterface::OsiGrbSolverInterface (const OsiGrbSolverInterface &)

Copy constructor.

8.17.3.4 virtual OsiGrbSolverInterface::~OsiGrbSolverInterface () [virtual]

Destructor.

8.17.4 Member Function Documentation

8.17.4.1 virtual void OsiGrbSolverInterface::initialSolve () [virtual]

Solve initial LP relaxation.

Implements [OsiSolverInterface](#).

8.17.4.2 virtual void OsiGrbSolverInterface::resolve () [virtual]

Resolve an LP relaxation after problem modification.

Implements [OsiSolverInterface](#).

8.17.4.3 virtual void OsiGrbSolverInterface::branchAndBound () [virtual]

Invoke solver's built-in enumeration algorithm.

Implements [OsiSolverInterface](#).

8.17.4.4 bool OsiGrbSolverInterface::setIntParam (OsiIntParam *key*, int *value*) [virtual]

Set an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.17.4.5 bool OsiGrbSolverInterface::setDbiParam (OsiDbiParam *key*, double *value*) [virtual]

Set a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.17.4.6 bool OsiGrbSolverInterface::setStrParam (OsiStrParam *key*, const std::string & *value*) [virtual]

Set a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.17.4.7 `bool OsiGrbSolverInterface::setHintParam (OsiHintParam key, bool yesNo = true, OsiHintStrength strength = OsiHintTry, void * =NULL) [virtual]`

Set a hint parameter.

The `otherInformation` parameter can be used to pass in an arbitrary block of information which is interpreted by the OSI and the underlying solver. Users are cautioned that this hook is solver-specific.

Implementors: The default implementation completely ignores `otherInformation` and always throws an exception for `OsiForceDo`. This is almost certainly not the behaviour you want; you really should override this method.

Reimplemented from [OsiSolverInterface](#).

8.17.4.8 `bool OsiGrbSolverInterface::getIntParam (OsiIntParam key, int & value) const [virtual]`

Get an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.17.4.9 `bool OsiGrbSolverInterface::getDbParam (OsiDbParam key, double & value) const [virtual]`

Get a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.17.4.10 `bool OsiGrbSolverInterface::getStrParam (OsiStrParam key, std::string & value) const [virtual]`

Get a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.17.4.11 `bool OsiGrbSolverInterface::getHintParam (OsiHintParam key, bool & yesNo, OsiHintStrength & strength, void *& otherInformation) const [virtual]`

Get a hint parameter (all information)

Return all available information for the hint: sense, strength, and any extra information associated with the hint.

Implementors: The default implementation will always set `otherInformation` to `NULL`. This is almost certainly not the behaviour you want; you really should override this method.

Reimplemented from [OsiSolverInterface](#).

8.17.4.12 `bool OsiGrbSolverInterface::getHintParam (OsiHintParam key, bool & yesNo, OsiHintStrength & strength) const [virtual]`

Get a hint parameter (sense and strength only)

Return only the sense and strength of the hint.

Reimplemented from [OsiSolverInterface](#).

8.17.4.13 `bool OsiGrbSolverInterface::getHintParam (OsiHintParam key, bool & yesNo) const [virtual]`

Get a hint parameter (sense only)

Return only the sense (true/false) of the hint.

Reimplemented from [OsiSolverInterface](#).

8.17.4.14 `void OsiGrbSolverInterface::setMipStart (bool value) [inline]`

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

8.17.4.15 `bool OsiGrbSolverInterface::getMipStart () const` `[inline]`

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

8.17.4.16 `virtual bool OsiGrbSolverInterface::isAbandoned () const` `[virtual]`

Are there a numerical difficulties?

Implements [OsiSolverInterface](#).

8.17.4.17 `virtual bool OsiGrbSolverInterface::isProvenOptimal () const` `[virtual]`

Is optimality proven?

Implements [OsiSolverInterface](#).

8.17.4.18 `virtual bool OsiGrbSolverInterface::isProvenPrimalInfeasible () const` `[virtual]`

Is primal infeasibility proven?

Implements [OsiSolverInterface](#).

8.17.4.19 `virtual bool OsiGrbSolverInterface::isProvenDualInfeasible () const` `[virtual]`

Is dual infeasibility proven?

Implements [OsiSolverInterface](#).

8.17.4.20 `virtual bool OsiGrbSolverInterface::isPrimalObjectiveLimitReached () const` `[virtual]`

Is the given primal objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.17.4.21 `virtual bool OsiGrbSolverInterface::isDualObjectiveLimitReached () const` `[virtual]`

Is the given dual objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.17.4.22 `virtual bool OsiGrbSolverInterface::isIterationLimitReached () const` `[virtual]`

Iteration limit reached?

Implements [OsiSolverInterface](#).

8.17.4.23 `CoinWarmStart* OsiGrbSolverInterface::getEmptyWarmStart () const` `[virtual]`

Get an empty warm start object.

This routine returns an empty `CoinWarmStartBasis` object. Its purpose is to provide a way to give a client a warm start basis object of the appropriate type, which can be resized and modified as desired.

Implements [OsiSolverInterface](#).

8.17.4.24 `virtual CoinWarmStart* OsiGrbSolverInterface::getWarmStart () const` `[virtual]`

Get warmstarting information.

Implements [OsiSolverInterface](#).

8.17.4.25 `virtual bool OsiGrbSolverInterface::setWarmStart (const CoinWarmStart * warmstart) [virtual]`

Set warmstarting information.

Return true/false depending on whether the warmstart information was accepted or not.

Implements [OsiSolverInterface](#).

8.17.4.26 `virtual void OsiGrbSolverInterface::markHotStart () [virtual]`

Create a hotstart point of the optimization process.

Reimplemented from [OsiSolverInterface](#).

8.17.4.27 `virtual void OsiGrbSolverInterface::solveFromHotStart () [virtual]`

Optimize starting from the hotstart.

Reimplemented from [OsiSolverInterface](#).

8.17.4.28 `virtual void OsiGrbSolverInterface::unmarkHotStart () [virtual]`

Delete the snapshot.

Reimplemented from [OsiSolverInterface](#).

8.17.4.29 `virtual int OsiGrbSolverInterface::getNumCols () const [virtual]`

Get number of columns.

Implements [OsiSolverInterface](#).

8.17.4.30 `virtual int OsiGrbSolverInterface::getNumRows () const [virtual]`

Get number of rows.

Implements [OsiSolverInterface](#).

8.17.4.31 `virtual int OsiGrbSolverInterface::getNumElements () const [virtual]`

Get number of nonzero elements.

Implements [OsiSolverInterface](#).

8.17.4.32 `virtual const double* OsiGrbSolverInterface::getColLower () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column lower bounds.

Implements [OsiSolverInterface](#).

8.17.4.33 `virtual const double* OsiGrbSolverInterface::getColUpper () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column upper bounds.

Implements [OsiSolverInterface](#).

8.17.4.34 `virtual const char* OsiGrbSolverInterface::getRowSense () const [virtual]`

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

- 'L': \leq constraint

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

Implements [OsiSolverInterface](#).

8.17.4.35 `virtual const double* OsiGrbSolverInterface::getRightHandSide () const [virtual]`

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

- if `rowsense()[i] == 'L'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'G'` then `rhs()[i] == rowlower()[i]`
- if `rowsense()[i] == 'R'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'N'` then `rhs()[i] == 0.0`

Implements [OsiSolverInterface](#).

8.17.4.36 `virtual const double* OsiGrbSolverInterface::getRowRange () const [virtual]`

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

- if `rowsense()[i] == 'R'` then `rowrange()[i] == rowupper()[i] - rowlower()[i]`
- if `rowsense()[i] != 'R'` then `rowrange()[i]` is 0.0

Implements [OsiSolverInterface](#).

8.17.4.37 `virtual const double* OsiGrbSolverInterface::getRowLower () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.17.4.38 `virtual const double* OsiGrbSolverInterface::getRowUpper () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.17.4.39 `virtual const double* OsiGrbSolverInterface::getObjCoefficients () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of objective function coefficients.

Implements [OsiSolverInterface](#).

8.17.4.40 `virtual double OsiGrbSolverInterface::getObjSense () const [virtual]`

Get objective function sense (1 for min (default), -1 for max)

Implements [OsiSolverInterface](#).

8.17.4.41 `virtual bool OsiGrbSolverInterface::isContinuous (int colNumber) const [virtual]`

Return true if column is continuous.

Implements [OsiSolverInterface](#).

8.17.4.42 `virtual const CoinPackedMatrix* OsiGrbSolverInterface::getMatrixByRow () const [virtual]`

Get pointer to row-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.17.4.43 `virtual const CoinPackedMatrix* OsiGrbSolverInterface::getMatrixByCol () const [virtual]`

Get pointer to column-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.17.4.44 `virtual double OsiGrbSolverInterface::getInfinity () const [virtual]`

Get solver's value for infinity.

Implements [OsiSolverInterface](#).

8.17.4.45 `virtual const double* OsiGrbSolverInterface::getColSolution () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of primal solution vector.

Implements [OsiSolverInterface](#).

8.17.4.46 `virtual const double* OsiGrbSolverInterface::getRowPrice () const [virtual]`

Get pointer to array[[getNumRows\(\)](#)] of dual prices.

Implements [OsiSolverInterface](#).

8.17.4.47 `virtual const double* OsiGrbSolverInterface::getReducedCost () const [virtual]`

Get a pointer to array[[getNumCols\(\)](#)] of reduced costs.

Implements [OsiSolverInterface](#).

8.17.4.48 `virtual const double* OsiGrbSolverInterface::getRowActivity () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.17.4.49 `virtual double OsiGrbSolverInterface::getObjValue () const [virtual]`

Get objective function value.

Implements [OsiSolverInterface](#).

8.17.4.50 `virtual int OsiGrbSolverInterface::getIterationCount () const [virtual]`

Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver).

Implements [OsiSolverInterface](#).

8.17.4.51 `virtual std::vector<double*> OsiGrbSolverInterface::getDualRays (int maxNumRays, bool fullRay = false) const` [virtual]

Get as many dual rays as the solver can provide.

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

The first `getNumRows()` ray components will always be associated with the row duals (as returned by `getRowPrice()`). If `fullRay` is true, the final `getNumCols()` entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.17.4.52 `virtual std::vector<double*> OsiGrbSolverInterface::getPrimalRays (int maxNumRays) const` [virtual]

Get as many primal rays as the solver can provide.

(In case of proven dual infeasibility there should be at least one.)

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.17.4.53 `virtual void OsiGrbSolverInterface::setObjCoeff (int elementIndex, double elementValue)` [virtual]

Set an objective function coefficient.

Implements [OsiSolverInterface](#).

8.17.4.54 `virtual void OsiGrbSolverInterface::setObjCoeffSet (const int * indexFirst, const int * indexLast, const double * coeffList)` [virtual]

Set a a set of objective function coefficients.

Reimplemented from [OsiSolverInterface](#).

8.17.4.55 `virtual void OsiGrbSolverInterface::setColLower (int elementIndex, double elementValue)` [virtual]

Set a single column lower bound

Use `-COIN_DBL_MAX` for -infinity.

Implements [OsiSolverInterface](#).

8.17.4.56 `virtual void OsiGrbSolverInterface::setColUpper (int elementIndex, double elementValue)` [virtual]

Set a single column upper bound

Use `COIN_DBL_MAX` for infinity.

Implements [OsiSolverInterface](#).

8.17.4.57 `virtual void OsiGrbSolverInterface::setColBounds (int elementIndex, double lower, double upper)` [virtual]

Set a single column lower and upper bound

The default implementation just invokes `setColLower ()` and `setColUpper ()`

Reimplemented from [OsiSolverInterface](#).

8.17.4.58 `virtual void OsiGrbSolverInterface::setColSetBounds (const int * indexFirst, const int * indexLast, const double * boundList)` [virtual]

Set the bounds on a number of columns simultaneously

The default implementation just invokes `setColLower ()` and `setColUpper ()` over and over again.

Parameters

<code><code>[<i>indexfirst</i>,<i>index-</i> <i>Last</i>]</code></code>	contains the indices of the constraints whose either bound changes
<code><i>boundList</i></code>	the new lower/upper bound pairs for the variables

Reimplemented from [OsiSolverInterface](#).

8.17.4.59 `virtual void OsiGrbSolverInterface::setRowLower (int elementIndex, double elementValue)` [virtual]

Set a single row lower bound

Use -COIN_DBL_MAX for -infinity.

Implements [OsiSolverInterface](#).

8.17.4.60 `virtual void OsiGrbSolverInterface::setRowUpper (int elementIndex, double elementValue)` [virtual]

Set a single row upper bound

Use COIN_DBL_MAX for infinity.

Implements [OsiSolverInterface](#).

8.17.4.61 `virtual void OsiGrbSolverInterface::setRowBounds (int elementIndex, double lower, double upper)` [virtual]

Set a single row lower and upper bound

The default implementation just invokes `setRowLower ()` and `setRowUpper ()`

Reimplemented from [OsiSolverInterface](#).

8.17.4.62 `virtual void OsiGrbSolverInterface::setRowType (int index, char sense, double rightHandSide, double range)` [virtual]

Set the type of a single row

Implements [OsiSolverInterface](#).

8.17.4.63 `virtual void OsiGrbSolverInterface::setRowSetBounds (const int * indexFirst, const int * indexLast, const double * boundList)` [virtual]

Set the bounds on a number of rows simultaneously

The default implementation just invokes `setRowLower ()` and `setRowUpper ()` over and over again.

Parameters

<code><code>[indexfirst,index-Last]</code></code>	contains the indices of the constraints whose either bound changes
<code>boundList</code>	the new lower/upper bound pairs for the constraints

Reimplemented from [OsiSolverInterface](#).

8.17.4.64 `virtual void OsiGrbSolverInterface::setRowSetTypes (const int * indexFirst, const int * indexLast, const char * senseList, const double * rhsList, const double * rangeList)` [virtual]

Set the type of a number of rows simultaneously

The default implementation just invokes [setRowType\(\)](#) and over and over again.

Parameters

<code><code>[indexfirst,index-Last]</code></code>	contains the indices of the constraints whose type changes
<code>senseList</code>	the new senses
<code>rhsList</code>	the new right hand sides
<code>rangeList</code>	the new ranges

Reimplemented from [OsiSolverInterface](#).

8.17.4.65 `virtual void OsiGrbSolverInterface::setContinuous (int index)` [virtual]

Set the index-th variable to be a continuous variable.

Implements [OsiSolverInterface](#).

8.17.4.66 `virtual void OsiGrbSolverInterface::setInteger (int index)` [virtual]

Set the index-th variable to be an integer variable.

Implements [OsiSolverInterface](#).

8.17.4.67 `virtual void OsiGrbSolverInterface::setContinuous (const int * indices, int len)` [virtual]

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

Reimplemented from [OsiSolverInterface](#).

8.17.4.68 `virtual void OsiGrbSolverInterface::setInteger (const int * indices, int len)` [virtual]

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

Reimplemented from [OsiSolverInterface](#).

8.17.4.69 `virtual void OsiGrbSolverInterface::setRowName (int ndx, std::string name)` [virtual]

Set a row name.

Reimplemented from [OsiSolverInterface](#).

8.17.4.70 `virtual void OsiGrbSolverInterface::setColName (int ndx, std::string name)` [virtual]

Set a column name.

Reimplemented from [OsiSolverInterface](#).

8.17.4.71 `virtual void OsiGrbSolverInterface::setObjSense (double s) [virtual]`

Set objective function sense (1 for min (default), -1 for max,)

Implements [OsiSolverInterface](#).

8.17.4.72 `virtual void OsiGrbSolverInterface::setColSolution (const double * colsol) [virtual]`

Set the primal solution column values.

`colsol[numcols()]` is an array of values of the problem column variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of `colsol()` until changed by another call to `setColsol()` or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

Implements [OsiSolverInterface](#).

8.17.4.73 `virtual void OsiGrbSolverInterface::setRowPrice (const double * rowprice) [virtual]`

Set dual solution vector.

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

Implements [OsiSolverInterface](#).

8.17.4.74 `virtual void OsiGrbSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [virtual]`

Add a column (primal variable) to the problem.

Implements [OsiSolverInterface](#).

8.17.4.75 `virtual void OsiGrbSolverInterface::addCols (const int numcols, const CoinPackedVectorBase *const * cols, const double * collb, const double * colub, const double * obj) [virtual]`

Add a set of columns (primal variables) to the problem.

The default implementation simply makes repeated calls to [addCol\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.17.4.76 `virtual void OsiGrbSolverInterface::deleteCols (const int num, const int * colIndices) [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](#).

8.17.4.77 `virtual void OsiGrbSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.17.4.78 `virtual void OsiGrbSolverInterface::addRow (const CoinPackedVectorBase & vec, const char rowsen, const double rowrhs, const double rowrng) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.17.4.79 `virtual void OsiGrbSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const double * rowlb, const double * rowub) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.17.4.80 `virtual void OsiGrbSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const char * rowsen, const double * rowrhs, const double * rowrng) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.17.4.81 `virtual void OsiGrbSolverInterface::deleteRows (const int num, const int * rowIndices) [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](#).

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

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

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

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

Implements [OsiSolverInterface](#).

8.17.4.83 `virtual void OsiGrbSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, double *& rowlb, double *& rowub) [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](#).

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

Load in 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](#).

8.17.4.85 `virtual void OsiGrbSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, char *& rowsen, double *& rowrhs, double *& rowrng) [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](#).

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

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

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

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

8.17.4.88 `virtual int OsiGrbSolverInterface::readMps (const char * filename, const char * extension = "mps") [virtual]`

Read an mps file from the given filename.

Reimplemented from [OsiSolverInterface](#).

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

Write the problem into an mps file of the given filename.

If *objSense* is non zero then -1.0 forces the code to write a maximization objective and +1.0 to write a minimization one. If 0.0 then solver can do what it wants

Implements [OsiSolverInterface](#).

8.17.4.90 **GRBmodel*** OsiGrbSolverInterface::getLpPtr (int *keepCached* = **KEEPCACHED_NONE**)

8.17.4.91 **GRBEnv*** OsiGrbSolverInterface::getEnvironmentPtr () const

Method to access Gurobi environment pointer.

8.17.4.92 **bool** OsiGrbSolverInterface::isDemoLicense () const

Return whether the current Gurobi environment runs in demo mode.

8.17.4.93 **const char*** OsiGrbSolverInterface::getCtype () const

return a vector of variable types (continous, binary, integer)

8.17.4.94 **static void** OsiGrbSolverInterface::incrementInstanceCounter () [static]

Gurobi has a context which must be created prior to all other Gurobi calls.

This method:

- Increments by 1 the number of uses of the Gurobi environment.
- Creates the Gurobi context when the number of uses is change to 1 from 0.

8.17.4.95 **static void** OsiGrbSolverInterface::decrementInstanceCounter () [static]

Gurobi has a context which should be deleted after Gurobi calls.

This method:

- Decrements by 1 the number of uses of the Gurobi environment.
- Deletes the Gurobi context when the number of uses is change to 0 from 1.

8.17.4.96 **static void** OsiGrbSolverInterface::setEnvironment (**GRBEnv*** *globalenv*) [static]

sets the global gurobi environment to a user given one

8.17.4.97 **static unsigned int** OsiGrbSolverInterface::getNumInstances () [static]

Return the number of instances of instantiated objects using Gurobi services.

8.17.4.98 **virtual OsiSolverInterface*** OsiGrbSolverInterface::clone (**bool** *copyData* = **true**) const [virtual]

Clone.

Implements [OsiSolverInterface](#).

8.17.4.99 **OsiGrbSolverInterface&** OsiGrbSolverInterface::operator= (**const OsiGrbSolverInterface&** *rhs*)

Assignment operator.

8.17.4.100 **virtual void** OsiGrbSolverInterface::reset () [virtual]

Resets as if default constructor.

Reimplemented from [OsiSolverInterface](#).

8.17.4.101 `virtual int OsiGrbSolverInterface::canDoSimplexInterface () const [virtual]`

Returns 1 if can just do getBlv etc 2 if has all OsiSimplex methods and 0 if it has none.

Reimplemented from [OsiSolverInterface](#).

8.17.4.102 `virtual void OsiGrbSolverInterface::enableSimplexInterface (int doingPrimal) [inline],[virtual]`

Useless function, defined only for compatibility with OsiSimplexInterface.

Definition at line 665 of file OsiGrbSolverInterface.hpp.

8.17.4.103 `virtual void OsiGrbSolverInterface::disableSimplexInterface () [inline],[virtual]`

Useless function, defined only for compatibility with OsiSimplexInterface.

Reimplemented from [OsiSolverInterface](#).

Definition at line 672 of file OsiGrbSolverInterface.hpp.

8.17.4.104 `virtual void OsiGrbSolverInterface::enableFactorization () const [inline],[virtual]`

Useless function, defined only for compatibility with OsiSimplexInterface.

Reimplemented from [OsiSolverInterface](#).

Definition at line 679 of file OsiGrbSolverInterface.hpp.

8.17.4.105 `virtual void OsiGrbSolverInterface::disableFactorization () const [inline],[virtual]`

Useless function, defined only for compatibility with OsiSimplexInterface.

Reimplemented from [OsiSolverInterface](#).

Definition at line 686 of file OsiGrbSolverInterface.hpp.

8.17.4.106 `virtual bool OsiGrbSolverInterface::basisIsAvailable () const [virtual]`

Returns true if a basis is available.

Reimplemented from [OsiSolverInterface](#).

8.17.4.107 `virtual void OsiGrbSolverInterface::getBasisStatus (int * cstat, int * rstat) const [virtual]`

Returns a basis status of the structural/artificial variables At present as warm start i.e 0: free, 1: basic, 2: upper, 3: lower.

Reimplemented from [OsiSolverInterface](#).

8.17.4.108 `void OsiGrbSolverInterface::switchToLP ()`

Get indices of the pivot variable in each row

(order of indices corresponds to the order of elements in a vector returned by [getBlvACol\(\)](#) and [getBlvCol\(\)](#)).

switches Gurobi to prob type LP

8.17.4.109 `void OsiGrbSolverInterface::switchToMIP ()`

switches Gurobi to prob type MIP

8.17.4.110 **virtual OsiSolverInterface::ApplyCutsReturnCode OsiGrbSolverInterface::applyCuts (const OsiCuts & cs, double effectivenessLb = 0.0)** [virtual]

Apply a collection of cuts.

Only cuts which have an `effectiveness >= effectivenessLb` are applied.

- `ReturnCode.getNumineffective()` – number of cuts which were not applied because they had an `effectiveness < effectivenessLb`
- `ReturnCode.getNuminconsistent()` – number of invalid cuts
- `ReturnCode.getNuminconsistentWrtIntegerModel()` – number of cuts that are invalid with respect to this integer model
- `ReturnCode.getNuminfeasible()` – number of cuts that would make this integer model infeasible
- `ReturnCode.getNumApplied()` – number of integer cuts which were applied to the integer model
- `cs.size() == getNumineffective() + getNuminconsistent() + getNuminconsistentWrtIntegerModel() + getNuminfeasible() + getNumApplied()`

Reimplemented from [OsiSolverInterface](#).

8.17.4.111 **virtual void OsiGrbSolverInterface::applyRowCut (const OsiRowCut & rc)** [protected],[virtual]

Apply a row cut. Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.17.4.112 **virtual void OsiGrbSolverInterface::applyColCut (const OsiColCut & cc)** [protected],[virtual]

Apply a column cut (bound adjustment).

Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.17.4.113 **void OsiGrbSolverInterface::resizeColSpace (int minsize)** [private]

resizes `coltype_`, `colmap_O2G`, `colmap_G2O` vectors to be able to store at least `minsize` elements

8.17.4.114 **void OsiGrbSolverInterface::freeColSpace ()** [private]

frees `colsize_` vector

8.17.4.115 **void OsiGrbSolverInterface::resizeAuxColSpace (int minsize)** [private]

resizes `colmap_G2O` vector to be able to store at least `minsize` (auxiliary) elements

8.17.4.116 **void OsiGrbSolverInterface::resizeAuxColIndSpace ()** [private]

resizes `auxcolind` vector to current number of rows and inits values to -1

8.17.4.117 **GRBmodel* OsiGrbSolverInterface::getMutableLpPtr () const** [private]

Get LP Pointer for const methods.

8.17.4.118 **void OsiGrbSolverInterface::gutsOfCopy (const OsiGrbSolverInterface & source)** [private]

The real work of a copy constructor (used by copy and assignment)

8.17.4.119 `void OsiGrbSolverInterface::gutsOfConstructor () [private]`

The real work of the constructor.

8.17.4.120 `void OsiGrbSolverInterface::gutsOfDestructor () [private]`

The real work of the destructor.

8.17.4.121 `void OsiGrbSolverInterface::freeCachedColRim () [private]`

free cached column rim vectors

8.17.4.122 `void OsiGrbSolverInterface::freeCachedRowRim () [private]`

free cached row rim vectors

8.17.4.123 `void OsiGrbSolverInterface::freeCachedResults () [private]`

free cached result vectors

8.17.4.124 `void OsiGrbSolverInterface::freeCachedMatrix () [private]`

free cached matrices

8.17.4.125 `void OsiGrbSolverInterface::freeCachedData (int keepCached = KEEPCACHED_NONE) [private]`

free all cached data (except specified entries, see [getLpPtr\(\)](#))

8.17.4.126 `void OsiGrbSolverInterface::freeAllMemory () [private]`

free all allocated memory

8.17.4.127 `void OsiGrbSolverInterface::convertToRangedRow (int rowidx, double rhs, double range) [private]`

converts a normal row into a ranged row by adding an auxiliary variable

8.17.4.128 `void OsiGrbSolverInterface::convertToNormalRow (int rowidx, char sense, double rhs) [private]`

converts a ranged row into a normal row by removing its auxiliary variable

8.17.5 Friends And Related Function Documentation

8.17.5.1 `void OsiGrbSolverInterfaceUnitTest (const std::string & mpsDir, const std::string & netlibDir) [friend]`

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

8.17.6 Member Data Documentation

8.17.6.1 `GRBenv* OsiGrbSolverInterface::globalenv_ [static], [private]`

Gurobi environment pointer.

Definition at line 784 of file `OsiGrbSolverInterface.hpp`.

8.17.6.2 `bool OsiGrbSolverInterface::globalenv_is_ours` `[static], [private]`

whether OsiGrb has created the global environment (and thus should free it)

Definition at line 786 of file OsiGrbSolverInterface.hpp.

8.17.6.3 `unsigned int OsiGrbSolverInterface::numInstances_` `[static], [private]`

Number of instances using the global Gurobi environment.

Definition at line 788 of file OsiGrbSolverInterface.hpp.

8.17.6.4 `GRBEnv* OsiGrbSolverInterface::localenv_` `[mutable], [private]`

Gurobi environment used only by this class instance.

Definition at line 835 of file OsiGrbSolverInterface.hpp.

8.17.6.5 `GRBmodel* OsiGrbSolverInterface::lp_` `[mutable], [private]`

Gurobi model represented by this class instance.

Definition at line 838 of file OsiGrbSolverInterface.hpp.

8.17.6.6 `int* OsiGrbSolverInterface::hotStartCStat_` `[private]`

Hotstart information.

Definition at line 841 of file OsiGrbSolverInterface.hpp.

8.17.6.7 `int OsiGrbSolverInterface::hotStartCStatSize_` `[private]`

Definition at line 842 of file OsiGrbSolverInterface.hpp.

8.17.6.8 `int* OsiGrbSolverInterface::hotStartRStat_` `[private]`

Definition at line 843 of file OsiGrbSolverInterface.hpp.

8.17.6.9 `int OsiGrbSolverInterface::hotStartRStatSize_` `[private]`

Definition at line 844 of file OsiGrbSolverInterface.hpp.

8.17.6.10 `int OsiGrbSolverInterface::hotStartMaxIteration_` `[private]`

Definition at line 845 of file OsiGrbSolverInterface.hpp.

8.17.6.11 `int OsiGrbSolverInterface::nameDisc_` `[private]`

OSI name discipline.

Definition at line 848 of file OsiGrbSolverInterface.hpp.

8.17.6.12 `double* OsiGrbSolverInterface::obj_` `[mutable], [private]`

Pointer to objective vector.

Definition at line 853 of file OsiGrbSolverInterface.hpp.

8.17.6.13 `double* OsiGrbSolverInterface::collower_` `[mutable], [private]`

Pointer to dense vector of variable lower bounds.

Definition at line 856 of file OsiGrbSolverInterface.hpp.

8.17.6.14 `double* OsiGrbSolverInterface::colupper_ [mutable], [private]`

Pointer to dense vector of variable lower bounds.

Definition at line 859 of file OsiGrbSolverInterface.hpp.

8.17.6.15 `char* OsiGrbSolverInterface::rowsense_ [mutable], [private]`

Pointer to dense vector of row sense indicators.

Definition at line 862 of file OsiGrbSolverInterface.hpp.

8.17.6.16 `double* OsiGrbSolverInterface::rhs_ [mutable], [private]`

Pointer to dense vector of row right-hand side values.

Definition at line 865 of file OsiGrbSolverInterface.hpp.

8.17.6.17 `double* OsiGrbSolverInterface::rowrange_ [mutable], [private]`

Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)

Definition at line 868 of file OsiGrbSolverInterface.hpp.

8.17.6.18 `double* OsiGrbSolverInterface::rowlower_ [mutable], [private]`

Pointer to dense vector of row lower bounds.

Definition at line 871 of file OsiGrbSolverInterface.hpp.

8.17.6.19 `double* OsiGrbSolverInterface::rowupper_ [mutable], [private]`

Pointer to dense vector of row upper bounds.

Definition at line 874 of file OsiGrbSolverInterface.hpp.

8.17.6.20 `double* OsiGrbSolverInterface::colsol_ [mutable], [private]`

Pointer to primal solution vector.

Definition at line 877 of file OsiGrbSolverInterface.hpp.

8.17.6.21 `double* OsiGrbSolverInterface::rowsol_ [mutable], [private]`

Pointer to dual solution vector.

Definition at line 880 of file OsiGrbSolverInterface.hpp.

8.17.6.22 `double* OsiGrbSolverInterface::redcost_ [mutable], [private]`

Pointer to reduced cost vector.

Definition at line 883 of file OsiGrbSolverInterface.hpp.

8.17.6.23 `double* OsiGrbSolverInterface::rowact_ [mutable], [private]`

Pointer to row activity (slack) vector.

Definition at line 886 of file OsiGrbSolverInterface.hpp.

8.17.6.24 `CoinPackedMatrix* OsiGrbSolverInterface::matrixByRow_` `[mutable], [private]`

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 889 of file OsiGrbSolverInterface.hpp.

8.17.6.25 `CoinPackedMatrix* OsiGrbSolverInterface::matrixByCol_` `[mutable], [private]`

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 892 of file OsiGrbSolverInterface.hpp.

8.17.6.26 `bool OsiGrbSolverInterface::probtypemip_` `[mutable], [private]`

Stores whether we currently see the problem as a MIP.

Definition at line 898 of file OsiGrbSolverInterface.hpp.

8.17.6.27 `bool OsiGrbSolverInterface::domipstart` `[private]`

Whether to pass a column solution to CPLEX before starting MIP solve (copymipstart)

Definition at line 901 of file OsiGrbSolverInterface.hpp.

8.17.6.28 `int OsiGrbSolverInterface::colspace_` `[private]`

Size of allocated memory for coltype_, colmap_O2G, and (with offset auxcolspace) colmap_G2O.

Definition at line 904 of file OsiGrbSolverInterface.hpp.

8.17.6.29 `char* OsiGrbSolverInterface::coltype_` `[private]`

Pointer to dense vector of variable types (continous, binary, integer)

Definition at line 907 of file OsiGrbSolverInterface.hpp.

8.17.6.30 `int OsiGrbSolverInterface::nauxcols` `[private]`

Number of auxiliary columns in Gurobi model for handling of ranged rows.

Definition at line 910 of file OsiGrbSolverInterface.hpp.

8.17.6.31 `int OsiGrbSolverInterface::auxcolspace` `[private]`

Size of allocated memory for colmap_G2O that exceeds colspace_.

Definition at line 913 of file OsiGrbSolverInterface.hpp.

8.17.6.32 `int* OsiGrbSolverInterface::colmap_O2G` `[private]`

Maps variable indices from Osi to Gurobi Is NULL if there are no ranged rows! (assume identity mapping then)

Definition at line 917 of file OsiGrbSolverInterface.hpp.

8.17.6.33 `int* OsiGrbSolverInterface::colmap_G2O` `[private]`

Maps variable indices from Gurobi to Osi A negative value indicates that a variable is an auxiliary variable that was added to handle a ranged row -colmap_G2O[i]-1 gives the index of the ranged row in this case.

Is NULL if there are no ranged rows! (assume identity mapping then)

Definition at line 923 of file OsiGrbSolverInterface.hpp.

8.17.6.34 `int OsiGrbSolverInterface::auxcolindspace` [private]

Current length of auxcolind array.

Definition at line 926 of file `OsiGrbSolverInterface.hpp`.

8.17.6.35 `int* OsiGrbSolverInterface::auxcolind` [private]

Gives for each row the index of the corresponding auxiliary variable, if it is a ranged row.

Otherwise, gives -1. Is NULL if there are no ranged rows! (assume -1 for each row then)

Definition at line 931 of file `OsiGrbSolverInterface.hpp`.

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

- </home/ted/COIN/trunk/Osi/src/OsiGrb/OsiGrbSolverInterface.hpp>

8.18 OsiHotInfo Class Reference

This class contains the result of strong branching on a variable. When created it stores enough information for strong branching.

```
#include <OsiChooseVariable.hpp>
```

Public Member Functions

- `OsiHotInfo ()`
Default Constructor.
- `OsiHotInfo (OsiSolverInterface *solver, const OsiBranchingInformation *info, const OsiObject *const *objects, int whichObject)`
Constructor from useful information.
- `OsiHotInfo (const OsiHotInfo &)`
Copy constructor.
- `OsiHotInfo & operator= (const OsiHotInfo &rhs)`
Assignment operator.
- `virtual OsiHotInfo * clone () const`
Clone.
- `virtual ~OsiHotInfo ()`
Destructor.
- `int updateInformation (const OsiSolverInterface *solver, const OsiBranchingInformation *info, OsiChooseVariable *choose)`
Fill in useful information after strong branch.
- `double originalObjectiveValue () const`
Original objective value.
- `double upChange () const`
Up change - invalid if n-way.
- `double downChange () const`
Down change - invalid if n-way.
- `void setUpChange (double value)`
Set up change - invalid if n-way.
- `void setDownChange (double value)`

- Set down change - invalid if n-way.*
 - double [change](#) (int k) const
- Change on way k.*
 - int [upIterationCount](#) () const
- Up iteration count - invalid if n-way.*
 - int [downIterationCount](#) () const
- Down iteration count - invalid if n-way.*
 - int [iterationCount](#) (int k) const
- Iteration count on way k.*
 - int [upStatus](#) () const
- Up status - invalid if n-way.*
 - int [downStatus](#) () const
- Down status - invalid if n-way.*
 - void [setUpStatus](#) (int value)
- Set up status - invalid if n-way.*
 - void [setDownStatus](#) (int value)
- Set down status - invalid if n-way.*
 - int [status](#) (int k) const
- Status on way k.*
 - [OsiBranchingObject](#) * [branchingObject](#) () const
- Branching object.*
 - int [whichObject](#) () const

Protected Attributes

- double [originalObjectiveValue_](#)
- Original objective value.*
- double * [changes_](#)
- Objective changes.*
- int * [iterationCounts_](#)
- Iteration counts.*
- int * [statuses_](#)
- Status -1 - not done 0 - feasible and finished 1 - infeasible 2 - not finished.*
- [OsiBranchingObject](#) * [branchingObject_](#)
- Branching object.*
- int [whichObject_](#)
- Which object on list.*

8.18.1 Detailed Description

This class contains the result of strong branching on a variable When created it stores enough information for strong branching.

Definition at line 432 of file OsiChooseVariable.hpp.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 OsiHotInfo::OsiHotInfo ()

Default Constructor.

8.18.2.2 OsiHotInfo::OsiHotInfo (OsiSolverInterface * *solver*, const OsiBranchingInformation * *info*, const OsiObject *const * *objects*, int *whichObject*)

Constructor from useful information.

8.18.2.3 OsiHotInfo::OsiHotInfo (const OsiHotInfo &)

Copy constructor.

8.18.2.4 virtual OsiHotInfo::~OsiHotInfo () [virtual]

Destructor.

8.18.3 Member Function Documentation

8.18.3.1 OsiHotInfo& OsiHotInfo::operator= (const OsiHotInfo & *rhs*)

Assignment operator.

8.18.3.2 virtual OsiHotInfo* OsiHotInfo::clone () const [virtual]

Clone.

8.18.3.3 int OsiHotInfo::updateInformation (const OsiSolverInterface * *solver*, const OsiBranchingInformation * *info*, OsiChooseVariable * *choose*)

Fill in useful information after strong branch.

Return status

8.18.3.4 double OsiHotInfo::originalObjectiveValue () const [inline]

Original objective value.

Definition at line 463 of file OsiChooseVariable.hpp.

8.18.3.5 double OsiHotInfo::upChange () const [inline]

Up change - invalid if n-way.

Definition at line 466 of file OsiChooseVariable.hpp.

8.18.3.6 double OsiHotInfo::downChange () const [inline]

Down change - invalid if n-way.

Definition at line 469 of file OsiChooseVariable.hpp.

8.18.3.7 void OsiHotInfo::setUpChange (double *value*) [inline]

Set up change - invalid if n-way.

Definition at line 472 of file OsiChooseVariable.hpp.

8.18.3.8 `void OsiHotInfo::setDownChange (double value) [inline]`

Set down change - invalid if n-way.

Definition at line 475 of file OsiChooseVariable.hpp.

8.18.3.9 `double OsiHotInfo::change (int k) const [inline]`

Change on way *k*.

Definition at line 478 of file OsiChooseVariable.hpp.

8.18.3.10 `int OsiHotInfo::upIterationCount () const [inline]`

Up iteration count - invalid if n-way.

Definition at line 482 of file OsiChooseVariable.hpp.

8.18.3.11 `int OsiHotInfo::downIterationCount () const [inline]`

Down iteration count - invalid if n-way.

Definition at line 485 of file OsiChooseVariable.hpp.

8.18.3.12 `int OsiHotInfo::iterationCount (int k) const [inline]`

Iteration count on way *k*.

Definition at line 488 of file OsiChooseVariable.hpp.

8.18.3.13 `int OsiHotInfo::upStatus () const [inline]`

Up status - invalid if n-way.

Definition at line 492 of file OsiChooseVariable.hpp.

8.18.3.14 `int OsiHotInfo::downStatus () const [inline]`

Down status - invalid if n-way.

Definition at line 495 of file OsiChooseVariable.hpp.

8.18.3.15 `void OsiHotInfo::setUpStatus (int value) [inline]`

Set up status - invalid if n-way.

Definition at line 498 of file OsiChooseVariable.hpp.

8.18.3.16 `void OsiHotInfo::setDownStatus (int value) [inline]`

Set down status - invalid if n-way.

Definition at line 501 of file OsiChooseVariable.hpp.

8.18.3.17 `int OsiHotInfo::status (int k) const [inline]`

Status on way *k*.

Definition at line 504 of file OsiChooseVariable.hpp.

8.18.3.18 **OsiBranchingObject*** OsiHotInfo::branchingObject () const [inline]

Branching object.

Definition at line 507 of file OsiChooseVariable.hpp.

8.18.3.19 **int** OsiHotInfo::whichObject () const [inline]

Definition at line 509 of file OsiChooseVariable.hpp.

8.18.4 Member Data Documentation

8.18.4.1 **double** OsiHotInfo::originalObjectiveValue_ [protected]

Original objective value.

Definition at line 515 of file OsiChooseVariable.hpp.

8.18.4.2 **double*** OsiHotInfo::changes_ [protected]

Objective changes.

Definition at line 517 of file OsiChooseVariable.hpp.

8.18.4.3 **int*** OsiHotInfo::iterationCounts_ [protected]

Iteration counts.

Definition at line 519 of file OsiChooseVariable.hpp.

8.18.4.4 **int*** OsiHotInfo::statuses_ [protected]

Status -1 - not done 0 - feasible and finished 1 - infeasible 2 - not finished.

Definition at line 526 of file OsiChooseVariable.hpp.

8.18.4.5 **OsiBranchingObject*** OsiHotInfo::branchingObject_ [protected]

Branching object.

Definition at line 528 of file OsiChooseVariable.hpp.

8.18.4.6 **int** OsiHotInfo::whichObject_ [protected]

Which object on list.

Definition at line 530 of file OsiChooseVariable.hpp.

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

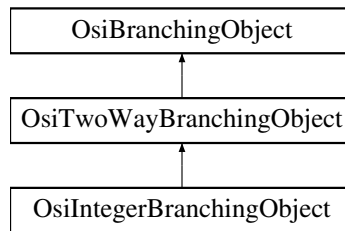
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiChooseVariable.hpp](#)

8.19 OsiIntegerBranchingObject Class Reference

Simple branching object for an integer variable.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for OsiIntegerBranchingObject:



Public Member Functions

- `OsiIntegerBranchingObject ()`
Default constructor.
- `OsiIntegerBranchingObject (OsiSolverInterface *solver, const OsiSimpleInteger *originalObject, int way, double value)`
Create a standard floor/ceiling branch object.
- `OsiIntegerBranchingObject (OsiSolverInterface *solver, const OsiSimpleInteger *originalObject, int way, double value, double downUpperBound, double upLowerBound)`
Create a standard floor/ceiling branch object.
- `OsiIntegerBranchingObject (const OsiIntegerBranchingObject &)`
Copy constructor.
- `OsiIntegerBranchingObject & operator= (const OsiIntegerBranchingObject &rhs)`
Assignment operator.
- `virtual OsiBranchingObject * clone () const`
Clone.
- `virtual ~OsiIntegerBranchingObject ()`
Destructor.
- `virtual double branch (OsiSolverInterface *solver)`
Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.
- `virtual void print (const OsiSolverInterface *solver=NULL)`
Print something about branch - only if log level high.

Protected Attributes

- `double down_ [2]`
Lower [0] and upper [1] bounds for the down arm (way_ = -1)
- `double up_ [2]`
Lower [0] and upper [1] bounds for the up arm (way_ = 1)

8.19.1 Detailed Description

Simple branching object for an integer variable.

This object can specify a two-way branch on an integer variable. For each arm of the branch, the upper and lower bounds on the variable can be independently specified. 0 -> down, 1-> up.

Definition at line 607 of file OsiBranchingObject.hpp.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 `OsiIntegerBranchingObject::OsiIntegerBranchingObject ()`

Default constructor.

8.19.2.2 `OsiIntegerBranchingObject::OsiIntegerBranchingObject (OsiSolverInterface * solver, const OsiSimpleInteger * originalObject, int way, double value)`

Create a standard floor/ceiling branch object.

Specifies a simple two-way branch. Let `value = x*`. One arm of the branch will be $lb \leq x \leq \text{floor}(x*)$, the other $\text{ceil}(x*) \leq x \leq ub$. Specify `way = -1` to set the object state to perform the down arm first, `way = 1` for the up arm.

8.19.2.3 `OsiIntegerBranchingObject::OsiIntegerBranchingObject (OsiSolverInterface * solver, const OsiSimpleInteger * originalObject, int way, double value, double downUpperBound, double upLowerBound)`

Create a standard floor/ceiling branch object.

Specifies a simple two-way branch in a more flexible way. One arm of the branch will be $lb \leq x \leq \text{downUpperBound}$, the other $\text{upLowerBound} \leq x \leq ub$. Specify `way = -1` to set the object state to perform the down arm first, `way = 1` for the up arm.

8.19.2.4 `OsiIntegerBranchingObject::OsiIntegerBranchingObject (const OsiIntegerBranchingObject &)`

Copy constructor.

8.19.2.5 `virtual OsiIntegerBranchingObject::~~OsiIntegerBranchingObject ()` [virtual]

Destructor.

8.19.3 Member Function Documentation

8.19.3.1 `OsiIntegerBranchingObject& OsiIntegerBranchingObject::operator= (const OsiIntegerBranchingObject & rhs)`

Assignment operator.

8.19.3.2 `virtual OsiBranchingObject* OsiIntegerBranchingObject::clone () const` [virtual]

Clone.

Implements [OsiBranchingObject](#).

8.19.3.3 `virtual double OsiIntegerBranchingObject::branch (OsiSolverInterface * solver)` [virtual]

Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.

state. Returns change in guessed objective on next branch

Implements [OsiTwoWayBranchingObject](#).

8.19.3.4 `virtual void OsiIntegerBranchingObject::print (const OsiSolverInterface * solver=NULL)` [virtual]

Print something about branch - only if log level high.

8.19.4 Member Data Documentation

8.19.4.1 `double OsiIntegerBranchingObject::down_[2]` [protected]

Lower [0] and upper [1] bounds for the down arm (`way_ = -1`)

Definition at line 661 of file `OsiBranchingObject.hpp`.

8.19.4.2 `double OsiIntegerBranchingObject::up_[2]` [protected]

Lower [0] and upper [1] bounds for the up arm (`way_ = 1`)

Definition at line 663 of file `OsiBranchingObject.hpp`.

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

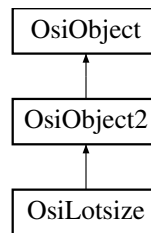
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp](#)

8.20 OsiLotsize Class Reference

Lotsize class.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for `OsiLotsize`:



Public Member Functions

- `OsiLotsize ()`
- `OsiLotsize (const OsiSolverInterface *solver, int iColumn, int numberPoints, const double *points, bool range=false)`
- `OsiLotsize (const OsiLotsize &)`
- `virtual OsiObject * clone () const`
Clone.
- `OsiLotsize & operator= (const OsiLotsize &rhs)`
- `virtual ~OsiLotsize ()`
- `virtual double infeasibility (const OsiBranchingInformation *info, int &whichWay) const`
Infeasibility - large is 0.5.
- `virtual double feasibleRegion (OsiSolverInterface *solver, const OsiBranchingInformation *info) const`
Set bounds to contain the current solution.
- `virtual OsiBranchingObject * createBranch (OsiSolverInterface *solver, const OsiBranchingInformation *info, int way) const`
Creates a branching object.
- `void setColumnNumber (int value)`
Set solver column number.
- `virtual int columnNumber () const`

- Column number if single column object -1 otherwise, so returns ≥ 0 Used by heuristics.*
- virtual void **resetBounds** (const **OsiSolverInterface** *solver)
 - Reset original upper and lower bound values from the solver.*
- bool **findRange** (double value, double integerTolerance) const
 - Finds range of interest so value is feasible in range range_ or infeasible between hi[range_] and lo[range_+1].*
- virtual void **floorCeiling** (double &floorLotsize, double &ceilingLotsize, double value, double tolerance) const
 - Returns floor and ceiling.*
- double **originalLowerBound** () const
 - Original bounds.*
- double **originalUpperBound** () const
- int **rangeType** () const
 - Type - 1 points, 2 ranges.*
- int **numberRanges** () const
 - Number of points.*
- double * **bound** () const
 - Ranges.*
- virtual void **resetSequenceEtc** (int numberColumns, const int *originalColumns)
 - Change column numbers after preprocessing.*
- virtual double **upEstimate** () const
 - Return "up" estimate (default 1.0e-5)*
- virtual double **downEstimate** () const
 - Return "down" estimate (default 1.0e-5)*
- virtual bool **canHandleShadowPrices** () const
 - Return true if knows how to deal with Pseudo Shadow Prices.*
- virtual bool **canDoHeuristics** () const
 - Return true if object can take part in normal heuristics.*

Private Attributes

- int **columnNumber_**
 - data*
- int **rangeType_**
 - Type - 1 points, 2 ranges.*
- int **numberRanges_**
 - Number of points.*
- double **largestGap_**
- double * **bound_**
 - Ranges.*
- int **range_**
 - Current range.*

Additional Inherited Members

8.20.1 Detailed Description

Lotsize class.

Definition at line 827 of file OsiBranchingObject.hpp.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 OsiLotsize::OsiLotsize ()

8.20.2.2 OsiLotsize::OsiLotsize (const OsiSolverInterface * *solver*, int *iColumn*, int *numberPoints*, const double * *points*, bool *range* = false)

8.20.2.3 OsiLotsize::OsiLotsize (const OsiLotsize &)

8.20.2.4 virtual OsiLotsize::~OsiLotsize () [virtual]

8.20.3 Member Function Documentation

8.20.3.1 virtual OsiObject* OsiLotsize::clone () const [virtual]

Clone.

Implements [OsiObject](#).8.20.3.2 OsiLotsize& OsiLotsize::operator= (const OsiLotsize & *rhs*)8.20.3.3 virtual double OsiLotsize::infeasibility (const OsiBranchingInformation * *info*, int & *whichWay*) const [virtual]

Infeasibility - large is 0.5.

Implements [OsiObject](#).8.20.3.4 virtual double OsiLotsize::feasibleRegion (OsiSolverInterface * *solver*, const OsiBranchingInformation * *info*) const [virtual]

Set bounds to contain the current solution.

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

Implements [OsiObject](#).8.20.3.5 virtual OsiBranchingObject* OsiLotsize::createBranch (OsiSolverInterface * *solver*, const OsiBranchingInformation * *info*, int *way*) const [virtual]

Creates a branching object.

The preferred direction is set by *way*, 0 for down, 1 for up.Reimplemented from [OsiObject](#).8.20.3.6 void OsiLotsize::setColumnNumber (int *value*) [inline]

Set solver column number.

Definition at line 874 of file OsiBranchingObject.hpp.

8.20.3.7 virtual int OsiLotsize::columnNumber () const [virtual]

Column number if single column object -1 otherwise, so returns ≥ 0 Used by heuristics.Reimplemented from [OsiObject](#).

8.20.3.8 `virtual void OsiLotsize::resetBounds (const OsiSolverInterface * solver) [virtual]`

Reset original upper and lower bound values from the solver.

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

Reimplemented from [OsiObject](#).

8.20.3.9 `bool OsiLotsize::findRange (double value, double integerTolerance) const`

Finds range of interest so value is feasible in range range_ or infeasible between hi[range_] and lo[range_+1].

Returns true if feasible.

8.20.3.10 `virtual void OsiLotsize::floorCeiling (double & floorLotsize, double & ceilingLotsize, double value, double tolerance) const [virtual]`

Returns floor and ceiling.

8.20.3.11 `double OsiLotsize::originalLowerBound () const [inline]`

Original bounds.

Definition at line 900 of file OsiBranchingObject.hpp.

8.20.3.12 `double OsiLotsize::originalUpperBound () const [inline]`

Definition at line 902 of file OsiBranchingObject.hpp.

8.20.3.13 `int OsiLotsize::rangeType () const [inline]`

Type - 1 points, 2 ranges.

Definition at line 905 of file OsiBranchingObject.hpp.

8.20.3.14 `int OsiLotsize::numberRanges () const [inline]`

Number of points.

Definition at line 908 of file OsiBranchingObject.hpp.

8.20.3.15 `double* OsiLotsize::bound () const [inline]`

Ranges.

Definition at line 911 of file OsiBranchingObject.hpp.

8.20.3.16 `virtual void OsiLotsize::resetSequenceEtc (int numberColumns, const int * originalColumns) [virtual]`

Change column numbers after preprocessing.

Reimplemented from [OsiObject](#).

8.20.3.17 `virtual double OsiLotsize::upEstimate () const [virtual]`

Return "up" estimate (default 1.0e-5)

Reimplemented from [OsiObject](#).

8.20.3.18 `virtual double OsiLotsize::downEstimate () const [virtual]`

Return "down" estimate (default 1.0e-5)

Reimplemented from [OsiObject](#).

8.20.3.19 `virtual bool OsiLotsize::canHandleShadowPrices () const [inline],[virtual]`

Return true if knows how to deal with Pseudo Shadow Prices.

Reimplemented from [OsiObject](#).

Definition at line 922 of file [OsiBranchingObject.hpp](#).

8.20.3.20 `virtual bool OsiLotsize::canDoHeuristics () const [inline],[virtual]`

Return true if object can take part in normal heuristics.

Reimplemented from [OsiObject](#).

Definition at line 926 of file [OsiBranchingObject.hpp](#).

8.20.4 Member Data Documentation

8.20.4.1 `int OsiLotsize::columnNumber_ [private]`

data

Column number in model

Definition at line 933 of file [OsiBranchingObject.hpp](#).

8.20.4.2 `int OsiLotsize::rangeType_ [private]`

Type - 1 points, 2 ranges.

Definition at line 935 of file [OsiBranchingObject.hpp](#).

8.20.4.3 `int OsiLotsize::numberRanges_ [private]`

Number of points.

Definition at line 937 of file [OsiBranchingObject.hpp](#).

8.20.4.4 `double OsiLotsize::largestGap_ [private]`

Definition at line 939 of file [OsiBranchingObject.hpp](#).

8.20.4.5 `double* OsiLotsize::bound_ [private]`

Ranges.

Definition at line 941 of file [OsiBranchingObject.hpp](#).

8.20.4.6 `int OsiLotsize::range_ [mutable],[private]`

Current range.

Definition at line 943 of file [OsiBranchingObject.hpp](#).

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

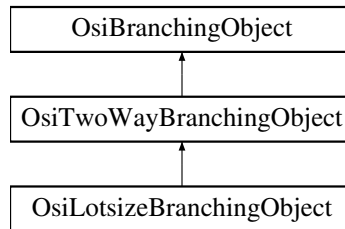
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp](#)

8.21 OsiLotsizeBranchingObject Class Reference

Lotsize branching object.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for OsiLotsizeBranchingObject:



Public Member Functions

- [OsiLotsizeBranchingObject](#) ()
Default constructor.
- [OsiLotsizeBranchingObject](#) ([OsiSolverInterface](#) *solver, const [OsiLotsize](#) *originalObject, int way, double value)
Create a lotsize floor/ceiling branch object.
- [OsiLotsizeBranchingObject](#) (const [OsiLotsizeBranchingObject](#) &)
Copy constructor.
- [OsiLotsizeBranchingObject](#) & operator= (const [OsiLotsizeBranchingObject](#) &rhs)
Assignment operator.
- virtual [OsiBranchingObject](#) * clone () const
Clone.
- virtual ~[OsiLotsizeBranchingObject](#) ()
Destructor.
- virtual double branch ([OsiSolverInterface](#) *solver)
Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.
- virtual void print (const [OsiSolverInterface](#) *solver=NULL)
Print something about branch - only if log level high.

Protected Attributes

- double down_ [2]
Lower [0] and upper [1] bounds for the down arm (way_ = -1)
- double up_ [2]
Lower [0] and upper [1] bounds for the up arm (way_ = 1)

8.21.1 Detailed Description

Lotsize branching object.

This object can specify a two-way branch on an integer variable. For each arm of the branch, the upper and lower bounds on the variable can be independently specified.

Variable_ holds the index of the integer variable in the integerVariable_ array of the model.

Definition at line 957 of file OsiBranchingObject.hpp.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 OsiLotsizeBranchingObject::OsiLotsizeBranchingObject ()

Default constructor.

8.21.2.2 OsiLotsizeBranchingObject::OsiLotsizeBranchingObject (OsiSolverInterface * *solver*, const OsiLotsize * *originalObject*, int *way*, double *value*)

Create a lotsize floor/ceiling branch object.

Specifies a simple two-way branch. Let `value = x*`. One arm of the branch will be `lb <= x <= valid range below(x*)`, the other valid range `above(x*) <= x <= ub`. Specify `way = -1` to set the object state to perform the down arm first, `way = 1` for the up arm.

8.21.2.3 OsiLotsizeBranchingObject::OsiLotsizeBranchingObject (const OsiLotsizeBranchingObject &)

Copy constructor.

8.21.2.4 virtual OsiLotsizeBranchingObject::~~OsiLotsizeBranchingObject () [virtual]

Destructor.

8.21.3 Member Function Documentation

8.21.3.1 OsiLotsizeBranchingObject& OsiLotsizeBranchingObject::operator= (const OsiLotsizeBranchingObject & *rhs*)

Assignment operator.

8.21.3.2 virtual OsiBranchingObject* OsiLotsizeBranchingObject::clone () const [virtual]

Clone.

Implements [OsiBranchingObject](#).

8.21.3.3 virtual double OsiLotsizeBranchingObject::branch (OsiSolverInterface * *solver*) [virtual]

Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.

state. Returns change in guessed objective on next branch

Implements [OsiTwoWayBranchingObject](#).

8.21.3.4 virtual void OsiLotsizeBranchingObject::print (const OsiSolverInterface * *solver* = NULL) [virtual]

Print something about branch - only if log level high.

8.21.4 Member Data Documentation

8.21.4.1 double OsiLotsizeBranchingObject::down_[2] [protected]

Lower [0] and upper [1] bounds for the down arm (`way_ = -1`)

Definition at line 1001 of file `OsiBranchingObject.hpp`.

8.21.4.2 double OsiLotsizeBranchingObject::up_[2] [protected]

Lower [0] and upper [1] bounds for the up arm (way_ = 1)

Definition at line 1003 of file OsiBranchingObject.hpp.

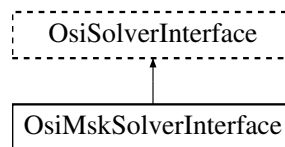
The documentation for this class was generated from the following file:

- </home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp>

8.22 OsiMskSolverInterface Class Reference

```
#include <OsiMskSolverInterface.hpp>
```

Inheritance diagram for OsiMskSolverInterface:



Public Member Functions

- virtual void [setObjSense](#) (double s)
Set objective function sense (1 for min (default), -1 for max,)
- virtual void [setColSolution](#) (const double *colsol)
Set the primal solution column values.
- virtual void [setRowPrice](#) (const double *rowprice)
Set dual solution vector.
- const char * [getCtype](#) () const
return a vector of variable types (continous, binary, integer)

Solve methods

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

Parameter set/get methods

The set methods return true if the parameter was set to the given value, false otherwise.

There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the volume algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.

The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.

- bool [setIntParam](#) ([OsiIntParam](#) key, int value)

- *Set an integer parameter.*
- bool [setDbiParam](#) ([OsiDbiParam](#) key, double value)
- *Set a double parameter.*
- bool [setStrParam](#) ([OsiStrParam](#) key, const std::string &value)
- *Set a string parameter.*
- bool [getIntParam](#) ([OsiIntParam](#) key, int &value) const
- *Get an integer parameter.*
- bool [getDbiParam](#) ([OsiDbiParam](#) key, double &value) const
- *Get a double parameter.*
- bool [getStrParam](#) ([OsiStrParam](#) key, std::string &value) const
- *Get a string parameter.*

Methods returning info on how the solution process terminated

- virtual bool [isAbandoned](#) () const
- *Are there a numerical difficulties?*
- virtual bool [isProvenOptimal](#) () const
- *Is optimality proven?*
- virtual bool [isProvenPrimalInfeasible](#) () const
- *Is primal infeasibility proven?*
- virtual bool [isProvenDualInfeasible](#) () const
- *Is dual infeasibility proven?*
- virtual bool [isPrimalObjectiveLimitReached](#) () const
- *Is the given primal objective limit reached?*
- virtual bool [isDualObjectiveLimitReached](#) () const
- *Is the given dual objective limit reached?*
- virtual bool [isIterationLimitReached](#) () const
- *Iteration limit reached?*
- virtual bool [isLicenseError](#) () const
- *Has there been a license problem?*
- int [getRescode](#) () const
- *Get rescode return of last Mosek optimizer call.*

WarmStart related methods

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

Hotstart related methods (primarily used in strong branching).

The user can create a hotstart (a snapshot) of the optimization process then reoptimize over and over again always starting from there.

NOTE: between hotstarted optimizations only bound changes are allowed.

- virtual void [markHotStart](#) ()
- *Create a hotstart point of the optimization process.*
- virtual void [solveFromHotStart](#) ()
- *Optimize starting from the hotstart.*
- virtual void [unmarkHotStart](#) ()
- *Delete the snapshot.*

Methods related to querying the input data

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

Methods related to querying the solution

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

Changing bounds on variables and constraints

- virtual void [setObjCoeff](#) (int elementIndex, double elementValue)
Set an objective function coefficient.
- virtual void [setObjCoeffSet](#) (const int *indexFirst, const int *indexLast, const double *coeffList)
Set a a set of objective function coefficients.
- virtual void [setColLower](#) (int elementIndex, double elementValue)
Set a single column lower bound
Use -COIN_DBL_MAX for -infinity.
- virtual void [setColUpper](#) (int elementIndex, double elementValue)
Set a single column upper bound
Use COIN_DBL_MAX for infinity.
- virtual void [setColBounds](#) (int elementIndex, double lower, double upper)
Set a single column lower and upper bound
The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#)
- virtual void [setColSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a number of columns simultaneously
The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#) over and over again.
- virtual void [setRowLower](#) (int elementIndex, double elementValue)
Set a single row lower bound
Use -COIN_DBL_MAX for -infinity.
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)
Set a single row upper bound
Use COIN_DBL_MAX for infinity.
- virtual void [setRowBounds](#) (int elementIndex, double lower, double upper)
Set a single row lower and upper bound
The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#)
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)
Set the type of a single row
- virtual void [setRowSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a number of rows simultaneously
The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#) over and over again.
- virtual void [setRowSetTypes](#) (const int *indexFirst, const int *indexLast, const char *senseList, const double *rhsList, const double *rangeList)
Set the type of a number of rows simultaneously
The default implementation just invokes [setRowType\(\)](#) and over and over again.

Integrality related changing methods

- virtual void [setContinuous](#) (int index)
Set the index-th variable to be a continuous variable.
- virtual void [setInteger](#) (int index)
Set the index-th variable to be an integer variable.
- virtual void [setContinuous](#) (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be continuous variables.
- virtual void [setInteger](#) (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be integer variables.

Methods to expand a problem.

Note that if a column is added then by default it will correspond to a continuous variable.

- virtual void [addCol](#) (const CoinPackedVectorBase &vec, const double collb, const double colub, const double obj)
Add a column (primal variable) to the problem.
- virtual void [addCols](#) (const int numcols, const CoinPackedVectorBase *const *cols, const double *collb, const double *colub, const double *obj)
Add a set of columns (primal variables) 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)
Add a row (constraint) to the problem.
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase *const *rows, const double *rowlb, const double *rowub)
Add a set of rows (constraints) to the problem.
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase *const *rows, const char *rowsen, const double *rowrhs, const double *rowrng)
Add a set of rows (constraints) to the problem.
- virtual void [deleteRows](#) (const int num, const int *rowIndices)
Delete a set of rows (constraints) from the problem.

Methods to input a problem

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

Message handling

- void [passInMessageHandler](#) (CoinMessageHandler *handler)
Pass in a message handler It is the client's responsibility to destroy a message handler installed by this routine; it will not be destroyed when the solver interface is destroyed.

Constructors and destructor

- [OsiMskSolverInterface](#) (MSKenv_t mskenv=NULL)

Default Constructor optional argument `mskenv` can be used to reach in an initialized user environment `OsiMsk` assumes membership of `mskenv`, so it will be freed when the last instantiation of `OsiMsk` is deleted.

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

Static Public Member Functions

Static instance counter methods

- static void [incrementInstanceCounter](#) ()
MOSEK has a context which must be created prior to all other MOSEK calls.
- static void [decrementInstanceCounter](#) ()
MOSEK has a context which should be deleted after MOSEK calls.
- static unsigned int [getNumInstances](#) ()
Return the number of instances of instantiated objects using MOSEK services.

Public Attributes

Private member data

- [MSKtask_t](#) [task_](#)
MOSEK model represented by this class instance.
- int * [hotStartCStat_](#)
Hotstart information.
- int [hotStartCStatSize_](#)
- int * [hotStartRStat_](#)
- int [hotStartRStatSize_](#)
- int [hotStartMaxIteration_](#)

Cached information derived from the MOSEK model

- double * [obj_](#)
Pointer to objective vector.
- double * [collower_](#)
Pointer to dense vector of variable lower bounds.
- double * [colupper_](#)
Pointer to dense vector of variable lower bounds.
- char * [rowsense_](#)
Pointer to dense vector of row sense indicators.
- double * [rhs_](#)
Pointer to dense vector of row right-hand side values.
- double * [rowrange_](#)
Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)
- double * [rowlower_](#)
Pointer to dense vector of row lower bounds.
- double * [rowupper_](#)
Pointer to dense vector of row upper bounds.
- double * [colsol_](#)

- *Pointer to primal solution vector.*
- double * [rowsol_](#)
Pointer to dual solution vector.
- double * [redcost_](#)
Pointer to reduced cost vector.
- double * [rowact_](#)
Pointer to row activity (slack) vector.
- CoinPackedMatrix * [matrixByRow_](#)
Pointer to row-wise copy of problem matrix coefficients.
- CoinPackedMatrix * [matrixByCol_](#)
Pointer to row-wise copy of problem matrix coefficients.

Additional information needed for storing MIP problems

- char * [coltype_](#)
Pointer to dense vector of variable types (continuous, binary, integer)
- int [coltypesize_](#)
Size of allocated memory for coltype_.
- bool [probtymip_](#)
Stores whether MOSEK' prob type is currently set to MIP.

Protected Member Functions

Protected methods

- virtual void [applyRowCut](#) (const [OsiRowCut](#) &rc)
Apply a row cut. Return true if cut was applied.
- virtual void [applyColCut](#) (const [OsiColCut](#) &cc)
Apply a column cut (bound adjustment).

Private Member Functions

Private static class functions

- void [switchToLP](#) ()
switches MOSEK to prob type LP
- void [switchToMIP](#) ()
switches MOSEK to prob type MIP
- void [resizeColType](#) (int minsize)
resizes coltype_ vector to be able to store at least minsize elements
- void [freeColType](#) ()
frees colsize_ vector
- bool [definedSolution](#) (int solution) const
- int [solverUsed](#) () const

Static Private Attributes

- static unsigned int [numInstances_](#)
Number of live problem instances.

Private static class data

- static [MSKenv_t](#) [env_](#)
MOSEK environment pointer.

Friends

- void [OsiMskSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)

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

MOSEK specific public interfaces

- enum [keepCachedFlag](#) {
[KEEPCACHED_NONE](#) = 0, [KEEPCACHED_COLUMN](#) = 1, [KEEPCACHED_ROW](#) = 2, [KEEPCACHED_MATRIX](#) = 4,
[KEEPCACHED_RESULTS](#) = 8, [KEEPCACHED_PROBLEM](#) = [KEEPCACHED_COLUMN](#) | [KEEPCACHED_ROW](#) | [KEEPCACHED_MATRIX](#), [KEEPCACHED_ALL](#) = [KEEPCACHED_PROBLEM](#) | [KEEPCACHED_RESULTS](#),
[FREECACHED_COLUMN](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_COLUMN](#),
[FREECACHED_ROW](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_ROW](#), [FREECACHED_MATRIX](#) = [KEEPCACHED_PROBLEM](#) & ~[KEEPCACHED_MATRIX](#), [FREECACHED_RESULTS](#) = [KEEPCACHED_ALL](#) & ~[KEEPCACHED_RESULTS](#) }

Get pointer to MOSEK model and free all specified cached data entries (combined with logical or-operator '|'):

- [MSKtask_t](#) [getLpPtr](#) (int keepCached=[KEEPCACHED_NONE](#))
- [MSKenv_t](#) [getEnvironmentPtr](#) ()

Method to access MOSEK environment pointer.

Private methods

- int [Mskerr](#)
- int [MSKsolverused_](#)
- double [ObjOffset_](#)
- int [InitialSolver](#)
- [MSKtask_t](#) [getMutableLpPtr](#) () const
Get task Pointer for const methods.
- void [gutsOfCopy](#) (const [OsiMskSolverInterface](#) &source)
The real work of a copy constructor (used by copy and assignment)
- void [gutsOfConstructor](#) ()
The real work of the constructor.
- void [gutsOfDestructor](#) ()
The real work of the destructor.
- void [freeCachedColRim](#) ()
free cached column rim vectors
- void [freeCachedRowRim](#) ()
free cached row rim vectors
- void [freeCachedResults](#) ()
free cached result vectors
- void [freeCachedMatrix](#) ()
free cached matrices
- void [freeCachedData](#) (int keepCached=[KEEPCACHED_NONE](#))
free all cached data (except specified entries, see [getLpPtr\(\)](#))
- void [freeAllMemory](#) ()
free all allocated memory

Additional Inherited Members

8.22.1 Detailed Description

Definition at line 23 of file OsiMskSolverInterface.hpp.

8.22.2 Member Enumeration Documentation

8.22.2.1 enum OsiMskSolverInterface::keepCachedFlag

Get pointer to MOSEK model and free all specified cached data entries (combined with logical or-operator '|'):

Enumerator

KEEPCACHED_NONE discard all cached data (default)
KEEPCACHED_COLUMN column information: objective values, lower and upper bounds, variable types
KEEPCACHED_ROW row information: right hand sides, ranges and senses, lower and upper bounds for row
KEEPCACHED_MATRIX problem matrix: matrix ordered by column and by row
KEEPCACHED_RESULTS LP solution: primal and dual solution, reduced costs, row activities.
KEEPCACHED_PROBLEM only discard cached LP solution
KEEPCACHED_ALL keep all cached data (similar to [getMutableLpPtr\(\)](#))
FREECACHED_COLUMN free only cached column and LP solution information
FREECACHED_ROW free only cached row and LP solution information
FREECACHED_MATRIX free only cached matrix and LP solution information
FREECACHED_RESULTS free only cached LP solution information

Definition at line 598 of file OsiMskSolverInterface.hpp.

8.22.3 Constructor & Destructor Documentation

8.22.3.1 OsiMskSolverInterface::OsiMskSolverInterface (MSKEnv_t mskenv = NULL)

Default Constructor optional argument mskenv can be used to reach in an initialized user environment OsiMsk assumes membership of mskenv, so it will be freed when the last instantiation of OsiMsk is deleted.

8.22.3.2 OsiMskSolverInterface::OsiMskSolverInterface (const OsiMskSolverInterface &)

Copy constructor.

8.22.3.3 virtual OsiMskSolverInterface::~~OsiMskSolverInterface () [virtual]

Destructor.

8.22.4 Member Function Documentation

8.22.4.1 virtual void OsiMskSolverInterface::initialSolve () [virtual]

Solve initial LP relaxation.

Implements [OsiSolverInterface](#).

8.22.4.2 `virtual void OsiMskSolverInterface::resolve () [virtual]`

Resolve an LP relaxation after problem modification.

Implements [OsiSolverInterface](#).

8.22.4.3 `virtual void OsiMskSolverInterface::branchAndBound () [virtual]`

Invoke solver's built-in enumeration algorithm.

Implements [OsiSolverInterface](#).

8.22.4.4 `bool OsiMskSolverInterface::setIntParam (OsiIntParam key, int value) [virtual]`

Set an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.22.4.5 `bool OsiMskSolverInterface::setDbiParam (OsiDbiParam key, double value) [virtual]`

Set a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.22.4.6 `bool OsiMskSolverInterface::setStrParam (OsiStrParam key, const std::string & value) [virtual]`

Set a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.22.4.7 `bool OsiMskSolverInterface::getIntParam (OsiIntParam key, int & value) const [virtual]`

Get an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.22.4.8 `bool OsiMskSolverInterface::getDbiParam (OsiDbiParam key, double & value) const [virtual]`

Get a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.22.4.9 `bool OsiMskSolverInterface::getStrParam (OsiStrParam key, std::string & value) const [virtual]`

Get a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.22.4.10 `virtual bool OsiMskSolverInterface::isAbandoned () const [virtual]`

Are there a numerical difficulties?

Implements [OsiSolverInterface](#).

8.22.4.11 `virtual bool OsiMskSolverInterface::isProvenOptimal () const [virtual]`

Is optimality proven?

Implements [OsiSolverInterface](#).

8.22.4.12 `virtual bool OsiMskSolverInterface::isProvenPrimalInfeasible () const [virtual]`

Is primal infeasibility proven?

Implements [OsiSolverInterface](#).

8.22.4.13 `virtual bool OsiMskSolverInterface::isProvenDualInfeasible () const [virtual]`

Is dual infeasibility proven?

Implements [OsiSolverInterface](#).

8.22.4.14 `virtual bool OsiMskSolverInterface::isPrimalObjectiveLimitReached () const [virtual]`

Is the given primal objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.22.4.15 `virtual bool OsiMskSolverInterface::isDualObjectiveLimitReached () const [virtual]`

Is the given dual objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.22.4.16 `virtual bool OsiMskSolverInterface::isIterationLimitReached () const [virtual]`

Iteration limit reached?

Implements [OsiSolverInterface](#).

8.22.4.17 `virtual bool OsiMskSolverInterface::isLicenseError () const [virtual]`

Has there been a license problem?

8.22.4.18 `int OsiMskSolverInterface::getRescode () const [inline]`

Get rescode return of last Mosek optimizer call.

Definition at line 91 of file `OsiMskSolverInterface.hpp`.

8.22.4.19 `CoinWarmStart* OsiMskSolverInterface::getEmptyWarmStart () const [virtual]`

Get an empty warm start object.

This routine returns an empty `CoinWarmStartBasis` object. Its purpose is to provide a way to give a client a warm start basis object of the appropriate type, which can be resized and modified as desired.

Implements [OsiSolverInterface](#).

8.22.4.20 `virtual CoinWarmStart* OsiMskSolverInterface::getWarmStart () const [virtual]`

Get warmstarting information.

Implements [OsiSolverInterface](#).

8.22.4.21 `virtual bool OsiMskSolverInterface::setWarmStart (const CoinWarmStart * warmstart) [virtual]`

Set warmstarting information.

Return true/false depending on whether the warmstart information was accepted or not.

Implements [OsiSolverInterface](#).

8.22.4.22 `virtual void OsiMskSolverInterface::markHotStart () [virtual]`

Create a hotstart point of the optimization process.

Reimplemented from [OsiSolverInterface](#).

8.22.4.23 `virtual void OsiMskSolverInterface::solveFromHotStart () [virtual]`

Optimize starting from the hotstart.

Reimplemented from [OsiSolverInterface](#).

8.22.4.24 `virtual void OsiMskSolverInterface::unmarkHotStart () [virtual]`

Delete the snapshot.

Reimplemented from [OsiSolverInterface](#).

8.22.4.25 `virtual int OsiMskSolverInterface::getNumCols () const [virtual]`

Get number of columns.

Implements [OsiSolverInterface](#).

8.22.4.26 `virtual int OsiMskSolverInterface::getNumRows () const [virtual]`

Get number of rows.

Implements [OsiSolverInterface](#).

8.22.4.27 `virtual int OsiMskSolverInterface::getNumElements () const [virtual]`

Get number of nonzero elements.

Implements [OsiSolverInterface](#).

8.22.4.28 `virtual const double* OsiMskSolverInterface::getColLower () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column lower bounds.

Implements [OsiSolverInterface](#).

8.22.4.29 `virtual const double* OsiMskSolverInterface::getColUpper () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column upper bounds.

Implements [OsiSolverInterface](#).

8.22.4.30 `virtual const char* OsiMskSolverInterface::getRowSense () const [virtual]`

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

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

Implements [OsiSolverInterface](#).

8.22.4.31 `virtual const double* OsiMskSolverInterface::getRightHandSide () const [virtual]`

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

- if `rowsense()[i] == 'L'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'G'` then `rhs()[i] == rowlower()[i]`
- if `rowsense()[i] == 'R'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'N'` then `rhs()[i] == 0.0`

Implements [OsiSolverInterface](#).

8.22.4.32 `virtual const double* OsiMskSolverInterface::getRowRange () const [virtual]`

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

- if `rowsense()[i] == 'R'` then `rowrange()[i] == rowupper()[i] - rowlower()[i]`
- if `rowsense()[i] != 'R'` then `rowrange()[i]` is 0.0

Implements [OsiSolverInterface](#).

8.22.4.33 `virtual const double* OsiMskSolverInterface::getRowLower () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.22.4.34 `virtual const double* OsiMskSolverInterface::getRowUpper () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.22.4.35 `virtual const double* OsiMskSolverInterface::getObjCoefficients () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of objective function coefficients.

Implements [OsiSolverInterface](#).

8.22.4.36 `virtual double OsiMskSolverInterface::getObjSense () const [virtual]`

Get objective function sense (1 for min (default), -1 for max)

Implements [OsiSolverInterface](#).

8.22.4.37 `virtual bool OsiMskSolverInterface::isContinuous (int colNumber) const [virtual]`

Return true if column is continuous.

Implements [OsiSolverInterface](#).

8.22.4.38 `virtual const CoinPackedMatrix* OsiMskSolverInterface::getMatrixByRow () const [virtual]`

Get pointer to row-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.22.4.39 `virtual const CoinPackedMatrix* OsiMskSolverInterface::getMatrixByCol () const [virtual]`

Get pointer to column-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.22.4.40 `virtual double OsiMskSolverInterface::getInfinity () const [virtual]`

Get solver's value for infinity.

Implements [OsiSolverInterface](#).

8.22.4.41 `virtual const double* OsiMskSolverInterface::getColSolution () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of primal solution vector.

Implements [OsiSolverInterface](#).

8.22.4.42 `virtual const double* OsiMskSolverInterface::getRowPrice () const [virtual]`

Get pointer to array[[getNumRows\(\)](#)] of dual prices.

Implements [OsiSolverInterface](#).

8.22.4.43 `virtual const double* OsiMskSolverInterface::getReducedCost () const [virtual]`

Get a pointer to array[[getNumCols\(\)](#)] of reduced costs.

Implements [OsiSolverInterface](#).

8.22.4.44 `virtual const double* OsiMskSolverInterface::getRowActivity () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.22.4.45 `virtual double OsiMskSolverInterface::getObjValue () const [virtual]`

Get objective function value.

Implements [OsiSolverInterface](#).

8.22.4.46 `virtual int OsiMskSolverInterface::getIterationCount () const [virtual]`

Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver).

Implements [OsiSolverInterface](#).

8.22.4.47 `virtual std::vector<double*> OsiMskSolverInterface::getDualRays (int maxNumRays, bool fullRay = false) const [virtual]`

Get as many dual rays as the solver can provide.

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

The first [getNumRows\(\)](#) ray components will always be associated with the row duals (as returned by [getRowPrice\(\)](#)). If `fullRay` is true, the final [getNumCols\(\)](#) entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.22.4.48 `virtual std::vector<double*> OsiMskSolverInterface::getPrimalRays (int maxNumRays) const` [virtual]

Get as many primal rays as the solver can provide.

(In case of proven dual infeasibility there should be at least one.)

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.22.4.49 `virtual void OsiMskSolverInterface::setObjCoeff (int elementIndex, double elementValue)` [virtual]

Set an objective function coefficient.

Implements [OsiSolverInterface](#).

8.22.4.50 `virtual void OsiMskSolverInterface::setObjCoeffSet (const int * indexFirst, const int * indexLast, const double * coeffList)` [virtual]

Set a a set of objective function coefficients.

Reimplemented from [OsiSolverInterface](#).

8.22.4.51 `virtual void OsiMskSolverInterface::setColLower (int elementIndex, double elementValue)` [virtual]

Set a single column lower bound

Use `-COIN_DBL_MAX` for -infinity.

Implements [OsiSolverInterface](#).

8.22.4.52 `virtual void OsiMskSolverInterface::setColUpper (int elementIndex, double elementValue)` [virtual]

Set a single column upper bound

Use `COIN_DBL_MAX` for infinity.

Implements [OsiSolverInterface](#).

8.22.4.53 `virtual void OsiMskSolverInterface::setColBounds (int elementIndex, double lower, double upper)` [virtual]

Set a single column lower and upper bound

The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#)

Reimplemented from [OsiSolverInterface](#).

8.22.4.54 `virtual void OsiMskSolverInterface::setColSetBounds (const int * indexFirst, const int * indexLast, const double * boundList)` [virtual]

Set the bounds on a number of columns simultaneously

The default implementation just invokes [setColLower\(\)](#) and [setColupper\(\)](#) over and over again.

Parameters

<code><code>[indexfirst,index-Last]</code></code>	contains the indices of the constraints whose either bound changes
<code>boundList</code>	the new lower/upper bound pairs for the variables

Reimplemented from [OsiSolverInterface](#).

8.22.4.55 `virtual void OsiMskSolverInterface::setRowLower (int elementIndex, double elementValue)` [virtual]

Set a single row lower bound

Use -COIN_DBL_MAX for -infinity.

Implements [OsiSolverInterface](#).

8.22.4.56 `virtual void OsiMskSolverInterface::setRowUpper (int elementIndex, double elementValue)` [virtual]

Set a single row upper bound

Use COIN_DBL_MAX for infinity.

Implements [OsiSolverInterface](#).

8.22.4.57 `virtual void OsiMskSolverInterface::setRowBounds (int elementIndex, double lower, double upper)` [virtual]

Set a single row lower and upper bound

The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#)

Reimplemented from [OsiSolverInterface](#).

8.22.4.58 `virtual void OsiMskSolverInterface::setRowType (int index, char sense, double rightHandSide, double range)` [virtual]

Set the type of a single row

Implements [OsiSolverInterface](#).

8.22.4.59 `virtual void OsiMskSolverInterface::setRowSetBounds (const int * indexFirst, const int * indexLast, const double * boundList)` [virtual]

Set the bounds on a number of rows simultaneously

The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#) over and over again.

Parameters

<code><code>[indexfirst,index-Last]</code></code>	contains the indices of the constraints whose either bound changes
<code>boundList</code>	the new lower/upper bound pairs for the constraints

Reimplemented from [OsiSolverInterface](#).

8.22.4.60 `virtual void OsiMskSolverInterface::setRowSetTypes (const int * indexFirst, const int * indexLast, const char * senseList, const double * rhsList, const double * rangeList)` [virtual]

Set the type of a number of rows simultaneously

The default implementation just invokes [setRowType\(\)](#) and over and over again.

Parameters

<code><code>[indexfirst, index-Last]</code></code>	contains the indices of the constraints whose type changes
<code>senseList</code>	the new senses
<code>rhsList</code>	the new right hand sides
<code>rangeList</code>	the new ranges

Reimplemented from [OsiSolverInterface](#).

8.22.4.61 `virtual void OsiMskSolverInterface::setContinuous (int index)` [virtual]

Set the index-th variable to be a continuous variable.

Implements [OsiSolverInterface](#).

8.22.4.62 `virtual void OsiMskSolverInterface::setInteger (int index)` [virtual]

Set the index-th variable to be an integer variable.

Implements [OsiSolverInterface](#).

8.22.4.63 `virtual void OsiMskSolverInterface::setContinuous (const int * indices, int len)` [virtual]

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

Reimplemented from [OsiSolverInterface](#).

8.22.4.64 `virtual void OsiMskSolverInterface::setInteger (const int * indices, int len)` [virtual]

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

Reimplemented from [OsiSolverInterface](#).

8.22.4.65 `virtual void OsiMskSolverInterface::setObjSense (double s)` [virtual]

Set objective function sense (1 for min (default), -1 for max,)

Implements [OsiSolverInterface](#).

8.22.4.66 `virtual void OsiMskSolverInterface::setColSolution (const double * colsol)` [virtual]

Set the primal solution column values.

`colsol[numcols()]` is an array of values of the problem column variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of `colsol()` until changed by another call to `setColsol()` or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

Implements [OsiSolverInterface](#).

8.22.4.67 `virtual void OsiMskSolverInterface::setRowPrice (const double * rowprice)` [virtual]

Set dual solution vector.

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

Implements [OsiSolverInterface](#).

8.22.4.68 `virtual void OsiMskSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [virtual]`

Add a column (primal variable) to the problem.

Implements [OsiSolverInterface](#).

8.22.4.69 `virtual void OsiMskSolverInterface::addCols (const int numcols, const CoinPackedVectorBase *const * cols, const double * collb, const double * colub, const double * obj) [virtual]`

Add a set of columns (primal variables) to the problem.

The default implementation simply makes repeated calls to [addCol\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.22.4.70 `virtual void OsiMskSolverInterface::deleteCols (const int num, const int * colIndices) [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](#).

8.22.4.71 `virtual void OsiMskSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.22.4.72 `virtual void OsiMskSolverInterface::addRow (const CoinPackedVectorBase & vec, const char rowsen, const double rowrhs, const double rowrng) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.22.4.73 `virtual void OsiMskSolverInterface::addRows (const int numRows, const CoinPackedVectorBase *const * rows, const double * rowlb, const double * rowub) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.22.4.74 `virtual void OsiMskSolverInterface::addRows (const int numRows, const CoinPackedVectorBase *const * rows, const char * rowsen, const double * rowrhs, const double * rowrng) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.22.4.75 `virtual void OsiMskSolverInterface::deleteRows (const int num, const int * rowIndices) [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](#).

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

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

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

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

Implements [OsiSolverInterface](#).

8.22.4.77 `virtual void OsiMskSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, double *& rowlb, double *& rowub) [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](#).

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

Load in 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](#).

8.22.4.79 `virtual void OsiMskSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& collb, double *& colub, double *& obj, char *& rowsen, double *& rowrhs, double *& rowrng) [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](#).

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

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

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

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

8.22.4.82 `virtual int OsiMskSolverInterface::readMps (const char * filename, const char * extension = "mps")` [virtual]

Read an mps file from the given filename.

Reimplemented from [OsiSolverInterface](#).

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

Write the problem into an mps file of the given filename.

If objSense is non zero then -1.0 forces the code to write a maximization objective and +1.0 to write a minimization one. If 0.0 then solver can do what it wants

Implements [OsiSolverInterface](#).

8.22.4.84 `void OsiMskSolverInterface::passInMessageHandler (CoinMessageHandler * handler)` [virtual]

Pass in a message handler It is the client's responsibility to destroy a message handler installed by this routine; it will not be destroyed when the solver interface is destroyed.

Reimplemented from [OsiSolverInterface](#).

8.22.4.85 `MSKtask_t OsiMskSolverInterface::getLpPtr (int keepCached = KEEPCACHED_NONE)`

8.22.4.86 `MSKenv_t OsiMskSolverInterface::getEnvironmentPtr ()`

Method to access MOSEK environment pointer.

8.22.4.87 `const char* OsiMskSolverInterface::getCtype () const`

return a vector of variable types (continous, binary, integer)

8.22.4.88 `static void OsiMskSolverInterface::incrementInstanceCounter ()` [static]

MOSEK has a context which must be created prior to all other MOSEK calls.

This method:

- Increments by 1 the number of uses of the MOSEK environment.
- Creates the MOSEK context when the number of uses is change to 1 from 0.

8.22.4.89 `static void OsiMskSolverInterface::decrementInstanceCounter () [static]`

MOSEK has a context which should be deleted after MOSEK calls.

This method:

- Decrements by 1 the number of uses of the MOSEK environment.
- Deletes the MOSEK context when the number of uses is change to 0 from 1.

8.22.4.90 `static unsigned int OsiMskSolverInterface::getNumInstances () [static]`

Return the number of instances of instantiated objects using MOSEK services.

8.22.4.91 `virtual OsiSolverInterface* OsiMskSolverInterface::clone (bool copyData =true) const [virtual]`

Clone.

Implements [OsiSolverInterface](#).

8.22.4.92 `OsiMskSolverInterface& OsiMskSolverInterface::operator= (const OsiMskSolverInterface & rhs)`

Assignment operator.

8.22.4.93 `virtual void OsiMskSolverInterface::applyRowCut (const OsiRowCut & rc) [protected],[virtual]`

Apply a row cut. Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.22.4.94 `virtual void OsiMskSolverInterface::applyColCut (const OsiColCut & cc) [protected],[virtual]`

Apply a column cut (bound adjustment).

Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.22.4.95 `void OsiMskSolverInterface::switchToLP () [private]`

switches MOSEK to prob type LP

8.22.4.96 `void OsiMskSolverInterface::switchToMIP () [private]`

switches MOSEK to prob type MIP

8.22.4.97 `void OsiMskSolverInterface::resizeColType (int minsize) [private]`

resizes coltype_ vector to be able to store at least minsize elements

8.22.4.98 `void OsiMskSolverInterface::freeColType () [private]`

frees colsize_ vector

8.22.4.99 `bool OsiMskSolverInterface::definedSolution (int solution) const [private]`

8.22.4.100 `int OsiMskSolverInterface::solverUsed () const [private]`

8.22.4.101 **MSKtask_t** OsiMskSolverInterface::getMutableLpPtr () const

Get task Pointer for const methods.

8.22.4.102 void OsiMskSolverInterface::gutsOfCopy (const OsiMskSolverInterface & *source*)

The real work of a copy constructor (used by copy and assignment)

8.22.4.103 void OsiMskSolverInterface::gutsOfConstructor ()

The real work of the constructor.

8.22.4.104 void OsiMskSolverInterface::gutsOfDestructor ()

The real work of the destructor.

8.22.4.105 void OsiMskSolverInterface::freeCachedColRim ()

free cached column rim vectors

8.22.4.106 void OsiMskSolverInterface::freeCachedRowRim ()

free cached row rim vectors

8.22.4.107 void OsiMskSolverInterface::freeCachedResults ()

free cached result vectors

8.22.4.108 void OsiMskSolverInterface::freeCachedMatrix ()

free cached matrices

8.22.4.109 void OsiMskSolverInterface::freeCachedData (int *keepCached* = **KEEPCACHED_NONE**)

free all cached data (except specified entries, see [getLpPtr\(\)](#))

8.22.4.110 void OsiMskSolverInterface::freeAllMemory ()

free all allocated memory

8.22.5 Friends And Related Function Documentation

8.22.5.1 void OsiMskSolverInterfaceUnitTest (const std::string & *mpsDir*, const std::string & *netlibDir*) [friend]

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

8.22.6 Member Data Documentation

8.22.6.1 **MSKenv_t** OsiMskSolverInterface::env_ [static],[private]

MOSEK environment pointer.

Definition at line 717 of file OsiMskSolverInterface.hpp.

8.22.6.2 `unsigned int OsiMskSolverInterface::numInstances_` `[static],[private]`

Number of live problem instances.

Definition at line 722 of file `OsiMskSolverInterface.hpp`.

8.22.6.3 `int OsiMskSolverInterface::Mskerr` `[private]`

Definition at line 727 of file `OsiMskSolverInterface.hpp`.

8.22.6.4 `int OsiMskSolverInterface::MSKsolverused_` `[private]`

Definition at line 728 of file `OsiMskSolverInterface.hpp`.

8.22.6.5 `double OsiMskSolverInterface::ObjOffset_` `[private]`

Definition at line 729 of file `OsiMskSolverInterface.hpp`.

8.22.6.6 `int OsiMskSolverInterface::InitialSolver` `[private]`

Definition at line 731 of file `OsiMskSolverInterface.hpp`.

8.22.6.7 `MSKtask_t OsiMskSolverInterface::task_` `[mutable]`

MOSEK model represented by this class instance.

Definition at line 766 of file `OsiMskSolverInterface.hpp`.

8.22.6.8 `int* OsiMskSolverInterface::hotStartCStat_`

Hotstart information.

Definition at line 769 of file `OsiMskSolverInterface.hpp`.

8.22.6.9 `int OsiMskSolverInterface::hotStartCStatSize_`

Definition at line 770 of file `OsiMskSolverInterface.hpp`.

8.22.6.10 `int* OsiMskSolverInterface::hotStartRStat_`

Definition at line 771 of file `OsiMskSolverInterface.hpp`.

8.22.6.11 `int OsiMskSolverInterface::hotStartRStatSize_`

Definition at line 772 of file `OsiMskSolverInterface.hpp`.

8.22.6.12 `int OsiMskSolverInterface::hotStartMaxIteration_`

Definition at line 773 of file `OsiMskSolverInterface.hpp`.

8.22.6.13 `double* OsiMskSolverInterface::obj_` `[mutable]`

Pointer to objective vector.

Definition at line 778 of file `OsiMskSolverInterface.hpp`.

8.22.6.14 `double* OsiMskSolverInterface::collower_` `[mutable]`

Pointer to dense vector of variable lower bounds.

Definition at line 781 of file OsiMskSolverInterface.hpp.

8.22.6.15 `double* OsiMskSolverInterface::colupper_` [mutable]

Pointer to dense vector of variable lower bounds.

Definition at line 784 of file OsiMskSolverInterface.hpp.

8.22.6.16 `char* OsiMskSolverInterface::rowsense_` [mutable]

Pointer to dense vector of row sense indicators.

Definition at line 787 of file OsiMskSolverInterface.hpp.

8.22.6.17 `double* OsiMskSolverInterface::rhs_` [mutable]

Pointer to dense vector of row right-hand side values.

Definition at line 790 of file OsiMskSolverInterface.hpp.

8.22.6.18 `double* OsiMskSolverInterface::rowrange_` [mutable]

Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)

Definition at line 793 of file OsiMskSolverInterface.hpp.

8.22.6.19 `double* OsiMskSolverInterface::rowlower_` [mutable]

Pointer to dense vector of row lower bounds.

Definition at line 796 of file OsiMskSolverInterface.hpp.

8.22.6.20 `double* OsiMskSolverInterface::rowupper_` [mutable]

Pointer to dense vector of row upper bounds.

Definition at line 799 of file OsiMskSolverInterface.hpp.

8.22.6.21 `double* OsiMskSolverInterface::colsol_` [mutable]

Pointer to primal solution vector.

Definition at line 802 of file OsiMskSolverInterface.hpp.

8.22.6.22 `double* OsiMskSolverInterface::rowsol_` [mutable]

Pointer to dual solution vector.

Definition at line 805 of file OsiMskSolverInterface.hpp.

8.22.6.23 `double* OsiMskSolverInterface::redcost_` [mutable]

Pointer to reduced cost vector.

Definition at line 808 of file OsiMskSolverInterface.hpp.

8.22.6.24 `double* OsiMskSolverInterface::rowact_` [mutable]

Pointer to row activity (slack) vector.

Definition at line 811 of file OsiMskSolverInterface.hpp.

8.22.6.25 `CoinPackedMatrix* OsiMskSolverInterface::matrixByRow_` [mutable]

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 814 of file `OsiMskSolverInterface.hpp`.

8.22.6.26 `CoinPackedMatrix* OsiMskSolverInterface::matrixByCol_` [mutable]

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 817 of file `OsiMskSolverInterface.hpp`.

8.22.6.27 `char* OsiMskSolverInterface::coltype_`

Pointer to dense vector of variable types (continuous, binary, integer)

Definition at line 823 of file `OsiMskSolverInterface.hpp`.

8.22.6.28 `int OsiMskSolverInterface::coltypesize_`

Size of allocated memory for `coltype_`.

Definition at line 826 of file `OsiMskSolverInterface.hpp`.

8.22.6.29 `bool OsiMskSolverInterface::probtymip_` [mutable]

Stores whether MOSEK' prob type is currently set to MIP.

Definition at line 829 of file `OsiMskSolverInterface.hpp`.

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

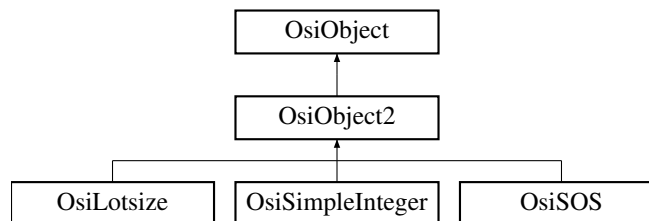
- </home/ted/COIN/trunk/Osi/src/OsiMsk/OsiMskSolverInterface.hpp>

8.23 OsiObject Class Reference

Abstract base class for 'objects'.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for `OsiObject`:



Public Member Functions

- [OsiObject\(\)](#)
Default Constructor.
- [OsiObject\(const OsiObject &\)](#)
Copy constructor.

- `OsiObject` & `operator=` (const `OsiObject` &rhs)
Assignment operator.
- virtual `OsiObject` * `clone` () const =0
Clone.
- virtual `~OsiObject` ()
Destructor.
- double `infeasibility` (const `OsiSolverInterface` *solver, int &`whichWay`) const
Infeasibility of the object.
- virtual double `infeasibility` (const `OsiBranchingInformation` *info, int &`whichWay`) const =0
- virtual double `checkInfeasibility` (const `OsiBranchingInformation` *info) const
- virtual double `feasibleRegion` (`OsiSolverInterface` *solver) const
For the variable(s) referenced by the object, look at the current solution and set bounds to match the solution.
- virtual double `feasibleRegion` (`OsiSolverInterface` *solver, const `OsiBranchingInformation` *info) const =0
For the variable(s) referenced by the object, look at the current solution and set bounds to match the solution.
- virtual `OsiBranchingObject` * `createBranch` (`OsiSolverInterface` *, const `OsiBranchingInformation` *, int) const
Create a branching object and indicate which way to branch first.
- virtual bool `canDoHeuristics` () const
Return true if object can take part in normal heuristics.
- virtual bool `canMoveToNearest` () const
Return true if object can take part in move to nearest heuristic.
- virtual int `columnNumber` () const
Column number if single column object -1 otherwise, Used by heuristics.
- int `priority` () const
Return Priority - note 1 is highest priority.
- void `setPriority` (int `priority`)
Set priority.
- virtual bool `boundBranch` () const
Return true if branch should only bound variables.
- virtual bool `canHandleShadowPrices` () const
Return true if knows how to deal with Pseudo Shadow Prices.
- int `numberWays` () const
Return maximum number of ways branch may have.
- void `setNumberWays` (int `numberWays`)
Set maximum number of ways branch may have.
- void `setWhichWay` (int way)
Return preferred way to branch.
- int `whichWay` () const
Return current preferred way to branch.
- virtual int `preferredWay` () const
Get pre-emptive preferred way of branching - -1 off, 0 down, 1 up (for 2-way)
- double `infeasibility` () const
Return infeasibility.
- virtual double `upEstimate` () const
Return "up" estimate (default 1.0e-5)
- virtual double `downEstimate` () const
Return "down" estimate (default 1.0e-5)
- virtual void `resetBounds` (const `OsiSolverInterface` *)

- Reset variable bounds to their original values.*
- virtual void `resetSequenceEtc` (int, const int *)
Change column numbers after preprocessing.
- virtual void `updateBefore` (const `OsiObject` *)
Updates stuff like pseudocosts before threads.
- virtual void `updateAfter` (const `OsiObject` *, const `OsiObject` *)
Updates stuff like pseudocosts after threads finished.

Protected Attributes

- double `infeasibility_`
data
- short `whichWay_`
Computed preferred way to branch.
- short `numberWays_`
Maximum number of ways on branch.
- int `priority_`
Priority.

8.23.1 Detailed Description

Abstract base class for 'objects'.

The branching model used in Osi is based on the idea of an *object*. In the abstract, an object is something that has a feasible region, can be evaluated for infeasibility, can be branched on (*i.e.*, there's some constructive action to be taken to move toward feasibility), and allows comparison of the effect of branching.

This class (`OsiObject`) is the base class for an object. To round out the branching model, the class `OsiBranchingObject` describes how to perform a branch, and the class `OsiBranchDecision` describes how to compare two `OsiBranchingObject`s.

To create a new type of object you need to provide three methods: `infeasibility()`, `feasibleRegion()`, and `createBranch()`, described below.

This base class is primarily virtual to allow for any form of structure. Any form of discontinuity is allowed.

As there is an overhead in getting information from solvers and because other useful information is available there is also an `OsiBranchingInformation` class which can contain pointers to information. If used it must at minimum contain pointers to current value of objective, maximum allowed objective and pointers to arrays for bounds and solution and direction of optimization. Also integer and primal tolerance.

Classes which inherit might have other information such as depth, number of solutions, pseudo-shadow prices etc etc. May be easier just to throw in here - as I keep doing

Definition at line 56 of file `OsiBranchingObject.hpp`.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 `OsiObject::OsiObject ()`

Default Constructor.

8.23.2.2 `OsiObject::OsiObject (const OsiObject &)`

Copy constructor.

8.23.2.3 `virtual OsiObject::~~OsiObject () [virtual]`

Destructor.

8.23.3 Member Function Documentation

8.23.3.1 `OsiObject& OsiObject::operator= (const OsiObject & rhs)`

Assignment operator.

8.23.3.2 `virtual OsiObject* OsiObject::clone () const [pure virtual]`

Clone.

Implemented in [OsiLotsize](#), [OsiSOS](#), and [OsiSimpleInteger](#).

8.23.3.3 `double OsiObject::infeasibility (const OsiSolverInterface * solver, int & whichWay) const`

Infeasibility of the object.

This is some measure of the infeasibility of the object. 0.0 indicates that the object is satisfied.

The preferred branching direction is returned in whichWay, where for normal two-way branching 0 is down, 1 is up

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

This should also set mutable infeasibility_ and whichWay_ This is for instant re-use for speed

Default for this just calls infeasibility with [OsiBranchingInformation](#) NOTE - Convention says that an infeasibility of COI-N_DBL_MAX means object has worked out it can't be satisfied!

8.23.3.4 `virtual double OsiObject::infeasibility (const OsiBranchingInformation * info, int & whichWay) const [pure virtual]`

Implemented in [OsiLotsize](#), [OsiSOS](#), and [OsiSimpleInteger](#).

8.23.3.5 `virtual double OsiObject::checkInfeasibility (const OsiBranchingInformation * info) const [virtual]`

8.23.3.6 `virtual double OsiObject::feasibleRegion (OsiSolverInterface * solver) const [virtual]`

For the variable(s) referenced by the object, look at the current solution and set bounds to match the solution.

Returns measure of how much it had to move solution to make feasible

8.23.3.7 `virtual double OsiObject::feasibleRegion (OsiSolverInterface * solver, const OsiBranchingInformation * info) const [pure virtual]`

For the variable(s) referenced by the object, look at the current solution and set bounds to match the solution.

Returns measure of how much it had to move solution to make feasible Faster version

Implemented in [OsiLotsize](#), [OsiSOS](#), and [OsiSimpleInteger](#).

8.23.3.8 `virtual OsiBranchingObject* OsiObject::createBranch (OsiSolverInterface *, const OsiBranchingInformation *, int) 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.)

Reimplemented in [OsiLotsize](#), [OsiSOS](#), and [OsiSimpleInteger](#).

Definition at line 119 of file OsiBranchingObject.hpp.

8.23.3.9 `virtual bool OsiObject::canDoHeuristics () const [inline],[virtual]`

Return true if object can take part in normal heuristics.

Reimplemented in [OsiLotsize](#), and [OsiSOS](#).

Definition at line 125 of file OsiBranchingObject.hpp.

8.23.3.10 `virtual bool OsiObject::canMoveToNearest () const [inline],[virtual]`

Return true if object can take part in move to nearest heuristic.

Definition at line 129 of file OsiBranchingObject.hpp.

8.23.3.11 `virtual int OsiObject::columnNumber () const [virtual]`

Column number if single column object -1 otherwise, Used by heuristics.

Reimplemented in [OsiLotsize](#), and [OsiSimpleInteger](#).

8.23.3.12 `int OsiObject::priority () const [inline]`

Return Priority - note 1 is highest priority.

Definition at line 136 of file OsiBranchingObject.hpp.

8.23.3.13 `void OsiObject::setPriority (int priority) [inline]`

Set priority.

Definition at line 139 of file OsiBranchingObject.hpp.

8.23.3.14 `virtual bool OsiObject::boundBranch () const [inline],[virtual]`

Return true if branch should only bound variables.

Definition at line 143 of file OsiBranchingObject.hpp.

8.23.3.15 `virtual bool OsiObject::canHandleShadowPrices () const [inline],[virtual]`

Return true if knows how to deal with Pseudo Shadow Prices.

Reimplemented in [OsiLotsize](#), [OsiSOS](#), and [OsiSimpleInteger](#).

Definition at line 146 of file OsiBranchingObject.hpp.

8.23.3.16 `int OsiObject::numberWays () const [inline]`

Return maximum number of ways branch may have.

Definition at line 149 of file OsiBranchingObject.hpp.

8.23.3.17 `void OsiObject::setNumberWays (int numberWays) [inline]`

Set maximum number of ways branch may have.

Definition at line 152 of file OsiBranchingObject.hpp.

8.23.3.18 `void OsiObject::setWhichWay (int way) [inline]`

Return preferred way to branch.

If two then way=0 means down and 1 means up, otherwise way points to preferred branch

Definition at line 158 of file OsiBranchingObject.hpp.

8.23.3.19 `int OsiObject::whichWay () const [inline]`

Return current preferred way to branch.

If two then way=0 means down and 1 means up, otherwise way points to preferred branch

Definition at line 164 of file OsiBranchingObject.hpp.

8.23.3.20 `virtual int OsiObject::preferredWay () const [inline],[virtual]`

Get pre-emptive preferred way of branching - -1 off, 0 down, 1 up (for 2-way)

Reimplemented in [OsiObject2](#).

Definition at line 167 of file OsiBranchingObject.hpp.

8.23.3.21 `double OsiObject::infeasibility () const [inline]`

Return infeasibility.

Definition at line 170 of file OsiBranchingObject.hpp.

8.23.3.22 `virtual double OsiObject::upEstimate () const [virtual]`

Return "up" estimate (default 1.0e-5)

Reimplemented in [OsiLotsize](#), [OsiSOS](#), and [OsiSimpleInteger](#).

8.23.3.23 `virtual double OsiObject::downEstimate () const [virtual]`

Return "down" estimate (default 1.0e-5)

Reimplemented in [OsiLotsize](#), [OsiSOS](#), and [OsiSimpleInteger](#).

8.23.3.24 `virtual void OsiObject::resetBounds (const OsiSolverInterface *) [inline],[virtual]`

Reset variable bounds to their original values.

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

Reimplemented in [OsiLotsize](#), and [OsiSimpleInteger](#).

Definition at line 180 of file OsiBranchingObject.hpp.

8.23.3.25 `virtual void OsiObject::resetSequenceEtc (int, const int *) [inline],[virtual]`

Change column numbers after preprocessing.

Reimplemented in [OsiLotsize](#), [OsiSOS](#), and [OsiSimpleInteger](#).

Definition at line 183 of file OsiBranchingObject.hpp.

8.23.3.26 `virtual void OsiObject::updateBefore (const OsiObject *) [inline],[virtual]`

Updates stuff like pseudocosts before threads.

Definition at line 185 of file OsiBranchingObject.hpp.

8.23.3.27 `virtual void OsiObject::updateAfter (const OsiObject *, const OsiObject *) [inline],[virtual]`

Updates stuff like pseudocosts after threads finished.

Definition at line 187 of file OsiBranchingObject.hpp.

8.23.4 Member Data Documentation

8.23.4.1 `double OsiObject::infeasibility_ [mutable],[protected]`

data

Computed infeasibility

Definition at line 193 of file OsiBranchingObject.hpp.

8.23.4.2 `short OsiObject::whichWay_ [mutable],[protected]`

Computed preferred way to branch.

Definition at line 195 of file OsiBranchingObject.hpp.

8.23.4.3 `short OsiObject::numberWays_ [protected]`

Maximum number of ways on branch.

Definition at line 197 of file OsiBranchingObject.hpp.

8.23.4.4 `int OsiObject::priority_ [protected]`

Priority.

Definition at line 199 of file OsiBranchingObject.hpp.

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

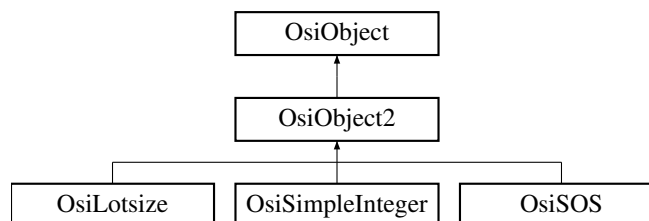
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp](#)

8.24 OsiObject2 Class Reference

Define a class to add a bit of complexity to [OsiObject](#) This assumes 2 way branching.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for OsiObject2:



Public Member Functions

- [OsiObject2](#) ()
Default Constructor.
- [OsiObject2](#) (const [OsiObject2](#) &)
Copy constructor.
- [OsiObject2](#) & [operator=](#) (const [OsiObject2](#) &rhs)
Assignment operator.
- virtual [~OsiObject2](#) ()
Destructor.
- void [setPreferredWay](#) (int value)
Set preferred way of branching - -1 off, 0 down, 1 up (for 2-way)
- virtual int [preferredWay](#) () const
Get preferred way of branching - -1 off, 0 down, 1 up (for 2-way)

Protected Attributes

- int [preferredWay_](#)
Preferred way of branching - -1 off, 0 down, 1 up (for 2-way)
- double [otherInfeasibility_](#)
"Infeasibility" on other way

8.24.1 Detailed Description

Define a class to add a bit of complexity to [OsiObject](#) This assumes 2 way branching.
Definition at line 206 of file OsiBranchingObject.hpp.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 [OsiObject2::OsiObject2](#) ()

Default Constructor.

8.24.2.2 [OsiObject2::OsiObject2](#) (const [OsiObject2](#) &)

Copy constructor.

8.24.2.3 virtual [OsiObject2::~~OsiObject2](#) () [virtual]

Destructor.

8.24.3 Member Function Documentation

8.24.3.1 [OsiObject2& OsiObject2::operator=](#) (const [OsiObject2](#) & rhs)

Assignment operator.

8.24.3.2 void OsiObject2::setPreferredWay (int value) [inline]

Set preferred way of branching - -1 off, 0 down, 1 up (for 2-way)

Definition at line 223 of file OsiBranchingObject.hpp.

8.24.3.3 virtual int OsiObject2::preferredWay () const [inline],[virtual]

Get preferred way of branching - -1 off, 0 down, 1 up (for 2-way)

Reimplemented from [OsiObject](#).

Definition at line 227 of file OsiBranchingObject.hpp.

8.24.4 Member Data Documentation

8.24.4.1 int OsiObject2::preferredWay_ [protected]

Preferred way of branching - -1 off, 0 down, 1 up (for 2-way)

Definition at line 231 of file OsiBranchingObject.hpp.

8.24.4.2 double OsiObject2::otherInfeasibility_ [mutable],[protected]

"Infeasibility" on other way

Definition at line 233 of file OsiBranchingObject.hpp.

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

- /home/ted/COIN/trunk/Osi/src/Osi/[OsiBranchingObject.hpp](#)

8.25 OsiPresolve Class Reference

OSI interface to COIN problem simplification capabilities.

```
#include <OsiPresolve.hpp>
```

Public Member Functions

- [OsiPresolve](#) ()
Default constructor (empty object)
- virtual [~OsiPresolve](#) ()
Virtual destructor.
- virtual [OsiSolverInterface](#) * [presolvedModel](#) ([OsiSolverInterface](#) &origModel, double feasibilityTolerance=0.0, bool keepIntegers=true, int numberPasses=5, const char *prohibited=NULL, bool doStatus=true, const char *row-Prohibited=NULL)
Create a new [OsiSolverInterface](#) loaded with the presolved problem.
- virtual void [postsolve](#) (bool updateStatus=true)
Restate the solution to the presolved problem in terms of the original problem and load it into the original model.
- [OsiSolverInterface](#) * [model](#) () const
Return a pointer to the presolved model.
- [OsiSolverInterface](#) * [originalModel](#) () const
Return a pointer to the original model.

- void `setOriginalModel` (`OsiSolverInterface *model`)
Set the pointer to the original model.
- const int * `originalColumns` () const
Return a pointer to the original columns.
- const int * `originalRows` () const
Return a pointer to the original rows.
- int `getNumRows` () const
Return number of rows in original model.
- int `getNumCols` () const
Return number of columns in original model.
- void `setNonLinearValue` (double value)
"Magic" number.
- double `nonLinearValue` () const
- void `setPresolveActions` (int action)
Fine control over presolve actions.

Protected Member Functions

- virtual const CoinPresolveAction * `presolve` (CoinPresolveMatrix *prob)
Apply presolve transformations to the problem.
- virtual void `postsolve` (CoinPostsolveMatrix &prob)
Reverse presolve transformations to recover the solution to the original problem.
- void `gutsOfDestroy` ()
Destroys queued postsolve actions.

Private Attributes

- `OsiSolverInterface * originalModel_`
Original model (solver interface loaded with the original problem).
- `OsiSolverInterface * presolvedModel_`
Presolved model (solver interface loaded with the presolved problem)
- double `nonLinearValue_`
"Magic" number.
- int * `originalColumn_`
Original column numbers.
- int * `originalRow_`
Original row numbers.
- const CoinPresolveAction * `paction_`
The list of transformations applied.
- int `ncols_`
Number of columns in original model.
- int `nrows_`
Number of rows in original model.
- CoinBigIndex `nelems_`
Number of nonzero matrix coefficients in the original model.
- int `presolveActions_`
Whether we want to skip dual part of presolve etc.
- int `numberPasses_`
Number of major passes.

8.25.1 Detailed Description

OSI interface to COIN problem simplification capabilities.

COIN provides a number of classes which implement problem simplification algorithms (CoinPresolveAction, CoinPrePostsolveMatrix, and derived classes). The model of operation is as follows:

- Create a copy of the original problem.
- Subject the copy to a series of transformations (the *presolve* methods) to produce a presolved model. Each transformation is also expected to provide a method to reverse the transformation (the *postsolve* method). The postsolve methods are collected in a linked list; the postsolve method for the final presolve transformation is at the head of the list.
- Hand the presolved problem to the solver for optimization.
- Apply the collected postsolve methods to the presolved problem and solution, restating the solution in terms of the original problem.

The COIN presolve algorithms are unaware of OSI. The [OsiPresolve](#) class takes care of the interface. Given an [OsiSolverInterface](#) `origModel`, it will take care of creating a clone properly loaded with the presolved problem and ready for optimization. After optimization, it will apply postsolve transformations and load the result back into `origModel`.

Assuming a problem has been loaded into an [OsiSolverInterface](#) `origModel`, a bare-bones application looks like this:

```
OsiPresolve pinfo ;
OsiSolverInterface *presolvedModel ;
// Return an OsiSolverInterface loaded with the presolved problem.
presolvedModel = pinfo.presolvedModel(*origModel, 1.0e-8, false, numberPasses) ;
presolvedModel->initialSolve() ;
// Restate the solution and load it back into origModel.
pinfo.postsolve(true) ;
delete presolvedModel ;
```

Definition at line 62 of file OsiPresolve.hpp.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 OsiPresolve::OsiPresolve ()

Default constructor (empty object)

8.25.2.2 virtual OsiPresolve::~~OsiPresolve () [virtual]

Virtual destructor.

8.25.3 Member Function Documentation

8.25.3.1 virtual OsiSolverInterface* OsiPresolve::presolvedModel (OsiSolverInterface & origModel, double feasibilityTolerance = 0.0, bool keepIntegers = true, int numberPasses = 5, const char * prohibited = NULL, bool doStatus = true, const char * rowProhibited = NULL) [virtual]

Create a new [OsiSolverInterface](#) loaded with the presolved problem.

This method implements the first two steps described in the class documentation. It clones `origModel` and applies presolve transformations, storing the resulting list of postsolve transformations. It returns a pointer to a new [OsiSolverInterface](#) loaded with the presolved problem, or NULL if the problem is infeasible or unbounded. If `keepIntegers` is

true then bounds may be tightened in the original. Bounds will be moved by up to `feasibilityTolerance` to try and stay feasible. When `doStatus` is true, the current solution will be transformed to match the presolved model.

This should be paired with `postsolve()`. It is up to the client to destroy the returned `OsiSolverInterface`, *after* calling `postsolve()`.

This method is virtual. Override this method if you need to customize the steps of creating a model to apply presolve transformations.

In some sense, a wrapper for `presolve(CoinPresolveMatrix*)`.

8.25.3.2 `virtual void OsiPresolve::postsolve (bool updateStatus = true) [virtual]`

Restate the solution to the presolved problem in terms of the original problem and load it into the original model.

`postsolve()` restates the solution in terms of the original problem and updates the original `OsiSolverInterface` supplied to `presolvedModel()`. If the problem has not been solved to optimality, there are no guarantees. If you are using an algorithm like simplex that has a concept of a basic solution, then set `updateStatus`

The advantage of going back to the original problem is that it will be exactly as it was, *i.e.*, 0.0 will not become 1.0e-19.

Note that if you modified the original problem after presolving, then you must “undo” these modifications before calling `postsolve()`.

In some sense, a wrapper for `postsolve(CoinPostsolveMatrix&)`.

8.25.3.3 `OsiSolverInterface* OsiPresolve::model () const`

Return a pointer to the presolved model.

8.25.3.4 `OsiSolverInterface* OsiPresolve::originalModel () const`

Return a pointer to the original model.

8.25.3.5 `void OsiPresolve::setOriginalModel (OsiSolverInterface * model)`

Set the pointer to the original model.

8.25.3.6 `const int* OsiPresolve::originalColumns () const`

Return a pointer to the original columns.

8.25.3.7 `const int* OsiPresolve::originalRows () const`

Return a pointer to the original rows.

8.25.3.8 `int OsiPresolve::getNumRows () const [inline]`

Return number of rows in original model.

Definition at line 133 of file `OsiPresolve.hpp`.

8.25.3.9 `int OsiPresolve::getNumCols () const [inline]`

Return number of columns in original model.

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

8.25.3.10 `void OsiPresolve::setNonLinearValue (double value) [inline]`

"Magic" number.

If this is non-zero then any elements with this value may change and so presolve is very limited in what can be done to the row and column. This is for non-linear problems.

Definition at line 144 of file OsiPresolve.hpp.

8.25.3.11 `double OsiPresolve::nonLinearValue () const [inline]`

Definition at line 146 of file OsiPresolve.hpp.

8.25.3.12 `void OsiPresolve::setPresolveActions (int action) [inline]`

Fine control over presolve actions.

Set/clear the following bits to allow or suppress actions:

- 0x01 allow duplicate column processing on integer columns and dual stuff on integers
- 0x02 switch off actions which can change +1 to something else (doubleton, tripleton, implied free)
- 0x04 allow transfer of costs from singletons and between integer variables (when advantageous)
- 0x08 do not allow $x+y+z=1$ transform
- 0x10 allow actions that don't easily unroll
- 0x20 allow dubious gub element reduction

GUB element reduction is only partially implemented in CoinPresolve (see `gubrow_action`) and will cause an abort at postsolve. It's not clear what's meant by 'dual stuff on integers'. – lh, 110605 –

Definition at line 166 of file OsiPresolve.hpp.

8.25.3.13 `virtual const CoinPresolveAction* OsiPresolve::presolve (CoinPresolveMatrix * prob) [protected],
[virtual]`

Apply presolve transformations to the problem.

Handles the core activity of applying presolve transformations.

If you want to apply the individual presolve routines differently, or perhaps add your own to the mix, define a derived class and override this method

8.25.3.14 `virtual void OsiPresolve::postsolve (CoinPostsolveMatrix & prob) [protected], [virtual]`

Reverse presolve transformations to recover the solution to the original problem.

Handles the core activity of applying postsolve transformations.

Postsolving is pretty generic; just apply the transformations in reverse order. You will probably only be interested in overriding this method if you want to add code to test for consistency while debugging new presolve techniques.

8.25.3.15 `void OsiPresolve::gutsOfDestroy () [protected]`

Destroys queued postsolve actions.

E.g., when `presolve()` determines the problem is infeasible, so that it will not be necessary to actually solve the presolved problem and convert the result back to the original problem.

8.25.4 Member Data Documentation

8.25.4.1 OsiSolverInterface* OsiPresolve::originalModel_ [private]

Original model (solver interface loaded with the original problem).

Must not be destroyed until after [postsolve\(\)](#).

Definition at line 174 of file OsiPresolve.hpp.

8.25.4.2 OsiSolverInterface* OsiPresolve::presolvedModel_ [private]

Presolved model (solver interface loaded with the presolved problem)

Must be destroyed by the client (using delete) after [postsolve\(\)](#).

Definition at line 180 of file OsiPresolve.hpp.

8.25.4.3 double OsiPresolve::nonLinearValue_ [private]

"Magic" number.

If this is non-zero then any elements with this value may change and so presolve is very limited in what can be done to the row and column. This is for non-linear problems. One could also allow for cases where sign of coefficient is known.

Definition at line 187 of file OsiPresolve.hpp.

8.25.4.4 int* OsiPresolve::originalColumn_ [private]

Original column numbers.

Definition at line 190 of file OsiPresolve.hpp.

8.25.4.5 int* OsiPresolve::originalRow_ [private]

Original row numbers.

Definition at line 193 of file OsiPresolve.hpp.

8.25.4.6 const CoinPresolveAction* OsiPresolve::paction_ [private]

The list of transformations applied.

Definition at line 196 of file OsiPresolve.hpp.

8.25.4.7 int OsiPresolve::ncols_ [private]

Number of columns in original model.

The problem will expand back to its former size as postsolve transformations are applied. It is efficient to allocate data structures for the final size of the problem rather than expand them as needed.

Definition at line 204 of file OsiPresolve.hpp.

8.25.4.8 int OsiPresolve::nrows_ [private]

Number of rows in original model.

Definition at line 207 of file OsiPresolve.hpp.

8.25.4.9 CoinBigIndex OsiPresolve::nelems_ [private]

Number of nonzero matrix coefficients in the original model.

Definition at line 210 of file OsiPresolve.hpp.

8.25.4.10 `int OsiPresolve::presolveActions_` [private]

Whether we want to skip dual part of presolve etc.

1 bit allows duplicate column processing on integer columns and dual stuff on integers 4 transfers costs to integer variables

Definition at line 217 of file `OsiPresolve.hpp`.

8.25.4.11 `int OsiPresolve::numberPasses_` [private]

Number of major passes.

Definition at line 219 of file `OsiPresolve.hpp`.

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

- `/home/ted/COIN/trunk/Osi/src/Osi/OsiPresolve.hpp`

8.26 OsiPseudoCosts Class Reference

This class is the placeholder for the pseudocosts used by `OsiChooseStrong`.

```
#include <OsiChooseVariable.hpp>
```

Public Member Functions

- `OsiPseudoCosts` ()
- virtual `~OsiPseudoCosts` ()
- `OsiPseudoCosts` (const `OsiPseudoCosts` &rhs)
- `OsiPseudoCosts` & `operator=` (const `OsiPseudoCosts` &rhs)
- int `numberBeforeTrusted` () const
Number of times before trusted.
- void `setNumberBeforeTrusted` (int value)
Set number of times before trusted.
- void `initialize` (int n)
Initialize the pseudocosts with n entries.
- int `numberObjects` () const
Give the number of objects for which pseudo costs are stored.
- virtual void `updateInformation` (const `OsiBranchingInformation` *info, int branch, `OsiHotInfo` *hotInfo)
Given a candidate fill in useful information e.g. estimates.
- virtual void `updateInformation` (int whichObject, int branch, double changeInObjective, double changeInValue, int status)
Given a branch fill in useful information e.g. estimates.

Accessor methods to pseudo costs data

- double * `upTotalChange` ()
- const double * `upTotalChange` () const
- double * `downTotalChange` ()
- const double * `downTotalChange` () const
- int * `upNumber` ()
- const int * `upNumber` () const
- int * `downNumber` ()
- const int * `downNumber` () const

Protected Attributes

- double * [upTotalChange_](#)
Total of all changes up.
- double * [downTotalChange_](#)
Total of all changes down.
- int * [upNumber_](#)
Number of times up.
- int * [downNumber_](#)
Number of times down.
- int [numberObjects_](#)
Number of objects (could be found from solver)
- int [numberBeforeTrusted_](#)
Number before we trust.

Private Member Functions

- void [gutsOfDelete](#) ()
- void [gutsOfCopy](#) (const [OsiPseudoCosts](#) &rhs)

8.26.1 Detailed Description

This class is the placeholder for the pseudocosts used by [OsiChooseStrong](#).
It can also be used by any other pseudocost based strong branching algorithm.
Definition at line 240 of file [OsiChooseVariable.hpp](#).

8.26.2 Constructor & Destructor Documentation

- 8.26.2.1 [OsiPseudoCosts::OsiPseudoCosts \(\)](#)
- 8.26.2.2 [virtual OsiPseudoCosts::~~OsiPseudoCosts \(\)](#) `[virtual]`
- 8.26.2.3 [OsiPseudoCosts::OsiPseudoCosts \(const \[OsiPseudoCosts\]\(#\) & rhs \)](#)

8.26.3 Member Function Documentation

- 8.26.3.1 [void OsiPseudoCosts::gutsOfDelete \(\)](#) `[private]`
- 8.26.3.2 [void OsiPseudoCosts::gutsOfCopy \(const \[OsiPseudoCosts\]\(#\) & rhs \)](#) `[private]`
- 8.26.3.3 [OsiPseudoCosts& OsiPseudoCosts::operator= \(const \[OsiPseudoCosts\]\(#\) & rhs \)](#)
- 8.26.3.4 [int OsiPseudoCosts::numberBeforeTrusted \(\) const](#) `[inline]`

Number of times before trusted.

Definition at line 267 of file [OsiChooseVariable.hpp](#).

8.26.3.5 `void OsiPseudoCosts::setNumberBeforeTrusted (int value) [inline]`

Set number of times before trusted.

Definition at line 270 of file OsiChooseVariable.hpp.

8.26.3.6 `void OsiPseudoCosts::initialize (int n)`

Initialize the pseudocosts with n entries.

8.26.3.7 `int OsiPseudoCosts::numberOfObjects () const [inline]`

Give the number of objects for which pseudo costs are stored.

Definition at line 275 of file OsiChooseVariable.hpp.

8.26.3.8 `double* OsiPseudoCosts::upTotalChange () [inline]`

Definition at line 280 of file OsiChooseVariable.hpp.

8.26.3.9 `const double* OsiPseudoCosts::upTotalChange () const [inline]`

Definition at line 281 of file OsiChooseVariable.hpp.

8.26.3.10 `double* OsiPseudoCosts::downTotalChange () [inline]`

Definition at line 283 of file OsiChooseVariable.hpp.

8.26.3.11 `const double* OsiPseudoCosts::downTotalChange () const [inline]`

Definition at line 284 of file OsiChooseVariable.hpp.

8.26.3.12 `int* OsiPseudoCosts::upNumber () [inline]`

Definition at line 286 of file OsiChooseVariable.hpp.

8.26.3.13 `const int* OsiPseudoCosts::upNumber () const [inline]`

Definition at line 287 of file OsiChooseVariable.hpp.

8.26.3.14 `int* OsiPseudoCosts::downNumber () [inline]`

Definition at line 289 of file OsiChooseVariable.hpp.

8.26.3.15 `const int* OsiPseudoCosts::downNumber () const [inline]`

Definition at line 290 of file OsiChooseVariable.hpp.

8.26.3.16 `virtual void OsiPseudoCosts::updateInformation (const OsiBranchingInformation * info, int branch, OsiHotInfo * hotInfo) [virtual]`

Given a candidate fill in useful information e.g. estimates.

8.26.3.17 `virtual void OsiPseudoCosts::updateInformation (int whichObject, int branch, double changeInObjective, double changeInValue, int status) [virtual]`

Given a branch fill in useful information e.g. estimates.

8.26.4 Member Data Documentation

8.26.4.1 `double* OsiPseudoCosts::upTotalChange_` [protected]

Total of all changes up.

Definition at line 244 of file OsiChooseVariable.hpp.

8.26.4.2 `double* OsiPseudoCosts::downTotalChange_` [protected]

Total of all changes down.

Definition at line 246 of file OsiChooseVariable.hpp.

8.26.4.3 `int* OsiPseudoCosts::upNumber_` [protected]

Number of times up.

Definition at line 248 of file OsiChooseVariable.hpp.

8.26.4.4 `int* OsiPseudoCosts::downNumber_` [protected]

Number of times down.

Definition at line 250 of file OsiChooseVariable.hpp.

8.26.4.5 `int OsiPseudoCosts::numberOfObjects_` [protected]

Number of objects (could be found from solver)

Definition at line 252 of file OsiChooseVariable.hpp.

8.26.4.6 `int OsiPseudoCosts::numberOfBeforeTrusted_` [protected]

Number before we trust.

Definition at line 254 of file OsiChooseVariable.hpp.

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

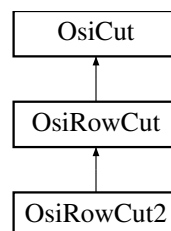
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiChooseVariable.hpp](#)

8.27 OsiRowCut Class Reference

Row Cut Class.

```
#include <OsiRowCut.hpp>
```

Inheritance diagram for OsiRowCut:



Public Member Functions

- void `sortIncrIndex` ()
Allow access row sorting function.

Row bounds

- `OsiRowCut_inline` double `lb` () const
Get lower bound.
- `OsiRowCut_inline` void `setLb` (double `lb`)
Set lower bound.
- `OsiRowCut_inline` double `ub` () const
Get upper bound.
- `OsiRowCut_inline` void `setUb` (double `ub`)
Set upper bound.

Row rhs, sense, range

- char `sense` () const
Get sense ('E', 'G', 'L', 'N', 'R')
- double `rhs` () const
Get right-hand side.
- double `range` () const
Get range (ub - lb for 'R' rows, 0 otherwise)

Row elements

- `OsiRowCut_inline` void `setRow` (int size, const int *colIndices, const double *elements, bool testForDuplicateIndex=`COIN_DEFAULT_VALUE_FOR_DUPLICATE`)
Set row elements.
- `OsiRowCut_inline` void `setRow` (const CoinPackedVector &v)
Set row elements from a packed vector.
- `OsiRowCut_inline` const CoinPackedVector & `row` () const
Get row elements.
- `OsiRowCut_inline` CoinPackedVector & `mutableRow` ()
Get row elements for changing.

Comparison operators

- `OsiRowCut_inline` bool `operator==` (const `OsiRowCut` &rhs) const
equal - true if lower bound, upper bound, row elements, and `OsiCut` are equal.
- `OsiRowCut_inline` bool `operator!=` (const `OsiRowCut` &rhs) const
not equal

Sanity checks on cut

- `OsiRowCut_inline` bool `consistent` () const
Returns true if the cut is consistent.
- `OsiRowCut_inline` bool `consistent` (const `OsiSolverInterface` &im) const
Returns true if cut is consistent with respect to the solver interface's model.
- `OsiRowCut_inline` bool `infeasible` (const `OsiSolverInterface` &im) const
Returns true if the row cut itself is infeasible and cannot be satisfied.
- virtual double `violated` (const double *solution) const

Returns infeasibility of the cut with respect to solution passed in i.e.

Arithmetic operators. Apply CoinPackedVector methods to the vector

- void `operator+=` (double value)
add value to every vector entry
- void `operator-=` (double value)
subtract value from every vector entry
- void `operator*=` (double value)
multiply every vector entry by value
- void `operator/=` (double value)
divide every vector entry by value

Constructors and destructors

- `OsiRowCut & operator=` (const `OsiRowCut` &rhs)
Assignment operator.
- `OsiRowCut` (const `OsiRowCut` &)
Copy constructor.
- virtual `OsiRowCut * clone` () const
Clone.
- `OsiRowCut` ()
Default Constructor.
- `OsiRowCut` (double cutlb, double cutub, int capacity, int size, int *&colIndices, double *&elements)
Ownership Constructor.
- virtual `~OsiRowCut` ()
Destructor.

Debug stuff

- virtual void `print` () const
Print cuts in collection.

Private Attributes

Private member data

- CoinPackedVector `row_`
Row elements.
- double `lb_`
Row lower bound.
- double `ub_`
Row upper bound.

Friends

- void `OsiRowCutUnitTest` (const `OsiSolverInterface` *baseSiP, const std::string &mpsDir)
A function that tests the methods in the `OsiRowCut` class.

Additional Inherited Members

8.27.1 Detailed Description

Row Cut Class.

A row cut has:

- a lower bound
- an upper bound
- a vector of row elements

Definition at line 29 of file OsiRowCut.hpp.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 OsiRowCut::OsiRowCut (const OsiRowCut &)

Copy constructor.

8.27.2.2 OsiRowCut::OsiRowCut ()

Default Constructor.

8.27.2.3 OsiRowCut::OsiRowCut (double *cutlb*, double *cutub*, int *capacity*, int *size*, int *& *colIndices*, double *& *elements*)

Ownership Constructor.

This constructor assumes ownership of the vectors passed as parameters for indices and elements. `colIndices` and `elements` will be NULL on return.

8.27.2.4 virtual OsiRowCut::~OsiRowCut () [virtual]

Destructor.

8.27.3 Member Function Documentation

8.27.3.1 OsiRowCut_inline double OsiRowCut::lb () const

Get lower bound.

8.27.3.2 OsiRowCut_inline void OsiRowCut::setLb (double *lb*)

Set lower bound.

8.27.3.3 OsiRowCut_inline double OsiRowCut::ub () const

Get upper bound.

8.27.3.4 OsiRowCut_inline void OsiRowCut::setUb (double *ub*)

Set upper bound.

8.27.3.5 `char OsiRowCut::sense () const`

Get sense ('E', 'G', 'L', 'N', 'R')

8.27.3.6 `double OsiRowCut::rhs () const`

Get right-hand side.

8.27.3.7 `double OsiRowCut::range () const`

Get range (ub - lb for 'R' rows, 0 otherwise)

8.27.3.8 `OsiRowCut_inline void OsiRowCut::setRow (int size, const int * colIndices, const double * elements, bool testForDuplicateIndex = COIN_DEFAULT_VALUE_FOR_DUPLICATE)`

Set row elements.

8.27.3.9 `OsiRowCut_inline void OsiRowCut::setRow (const CoinPackedVector & v)`

Set row elements from a packed vector.

8.27.3.10 `OsiRowCut_inline const CoinPackedVector& OsiRowCut::row () const`

Get row elements.

8.27.3.11 `OsiRowCut_inline CoinPackedVector& OsiRowCut::mutableRow ()`

Get row elements for changing.

8.27.3.12 `OsiRowCut_inline bool OsiRowCut::operator== (const OsiRowCut & rhs) const`

equal - true if lower bound, upper bound, row elements, and [OsiCut](#) are equal.

8.27.3.13 `OsiRowCut_inline bool OsiRowCut::operator!= (const OsiRowCut & rhs) const`

not equal

8.27.3.14 `OsiRowCut_inline bool OsiRowCut::consistent () const [virtual]`

Returns true if the cut is consistent.

This checks to ensure that:

- The row element vector does not have duplicate indices
- The row element vector indices are ≥ 0

Implements [OsiCut](#).

8.27.3.15 `OsiRowCut_inline bool OsiRowCut::consistent (const OsiSolverInterface & im) const [virtual]`

Returns true if cut is consistent with respect to the solver interface's model.

This checks to ensure that

- The row element vector indices are $<$ the number of columns in the model

Implements [OsiCut](#).

8.27.3.16 `OsiRowCut_inline bool OsiRowCut::infeasible (const OsiSolverInterface & im) const` [virtual]

Returns true if the row cut itself is infeasible and cannot be satisfied.

This checks whether

- the lower bound is strictly greater than the upper bound.

Implements [OsiCut](#).

8.27.3.17 `virtual double OsiRowCut::violated (const double * solution) const` [virtual]

Returns infeasibility of the cut with respect to solution passed in i.e.

is positive if cuts off that solution. solution is getNumCols() long..

Implements [OsiCut](#).

8.27.3.18 `void OsiRowCut::operator+=(double value)` [inline]

add *value* to every vector entry

Definition at line 132 of file OsiRowCut.hpp.

8.27.3.19 `void OsiRowCut::operator-= (double value)` [inline]

subtract *value* from every vector entry

Definition at line 136 of file OsiRowCut.hpp.

8.27.3.20 `void OsiRowCut::operator*= (double value)` [inline]

multiply every vector entry by *value*

Definition at line 140 of file OsiRowCut.hpp.

8.27.3.21 `void OsiRowCut::operator/= (double value)` [inline]

divide every vector entry by *value*

Definition at line 144 of file OsiRowCut.hpp.

8.27.3.22 `void OsiRowCut::sortIncrIndex ()` [inline]

Allow access row sorting function.

Definition at line 149 of file OsiRowCut.hpp.

8.27.3.23 `OsiRowCut& OsiRowCut::operator= (const OsiRowCut & rhs)`

Assignment operator.

8.27.3.24 `virtual OsiRowCut* OsiRowCut::clone () const` [virtual]

Clone.

Reimplemented in [OsiRowCut2](#).

8.27.3.25 `virtual void OsiRowCut::print () const` [virtual]

Print cuts in collection.

Reimplemented from [OsiCut](#).

8.27.4 Friends And Related Function Documentation

8.27.4.1 `void OsiRowCutUnitTest (const OsiSolverInterface * baseSiP, const std::string & mpsDir)` `[friend]`

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

8.27.5 Member Data Documentation

8.27.5.1 `CoinPackedVector OsiRowCut::row_` `[private]`

Row elements.

Definition at line 192 of file `OsiRowCut.hpp`.

8.27.5.2 `double OsiRowCut::lb_` `[private]`

Row lower bound.

Definition at line 194 of file `OsiRowCut.hpp`.

8.27.5.3 `double OsiRowCut::ub_` `[private]`

Row upper bound.

Definition at line 196 of file `OsiRowCut.hpp`.

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

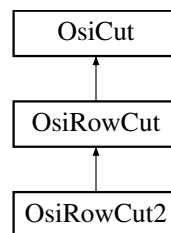
- `/home/ted/COIN/trunk/Osi/src/Osi/OsiRowCut.hpp`

8.28 OsiRowCut2 Class Reference

Row Cut Class which refers back to row which created it.

```
#include <OsiRowCut.hpp>
```

Inheritance diagram for `OsiRowCut2`:



Public Member Functions

Which row

- `int whichRow () const`
Get row.

- void `setWhichRow` (int `row`)
Set row.

Constructors and destructors

- `OsiRowCut2` & `operator=` (const `OsiRowCut2` &`rhs`)
Assignment operator.
- `OsiRowCut2` (const `OsiRowCut2` &)
Copy constructor.
- virtual `OsiRowCut` * `clone` () const
Clone.
- `OsiRowCut2` (int `row`== -1)
Default Constructor.
- virtual `~OsiRowCut2` ()
Destructor.

Private Attributes

Private member data

- int `whichRow_`
Which row.

Additional Inherited Members

8.28.1 Detailed Description

Row Cut Class which refers back to row which created it.

It may be useful to strengthen a row rather than add a cut. To do this we need to know which row is strengthened. This trivial extension to `OsiRowCut` does that.

Definition at line 290 of file `OsiRowCut.hpp`.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 `OsiRowCut2::OsiRowCut2 (const OsiRowCut2 &)`

Copy constructor.

8.28.2.2 `OsiRowCut2::OsiRowCut2 (int row = -1)`

Default Constructor.

8.28.2.3 `virtual OsiRowCut2::~OsiRowCut2 () [virtual]`

Destructor.

8.28.3 Member Function Documentation

8.28.3.1 `int OsiRowCut2::whichRow () const [inline]`

Get row.

Definition at line 297 of file `OsiRowCut.hpp`.

8.28.3.2 `void OsiRowCut2::setWhichRow (int row) [inline]`

Set row.

Definition at line 300 of file OsiRowCut.hpp.

8.28.3.3 `OsiRowCut2& OsiRowCut2::operator= (const OsiRowCut2 & rhs)`

Assignment operator.

8.28.3.4 `virtual OsiRowCut* OsiRowCut2::clone () const [virtual]`

Clone.

Reimplemented from [OsiRowCut](#).

8.28.4 Member Data Documentation

8.28.4.1 `int OsiRowCut2::whichRow_ [private]`

Which row.

Definition at line 328 of file OsiRowCut.hpp.

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

- [/home/ted/COIN/trunk/Osi/src/Osi/OsiRowCut.hpp](#)

8.29 OsiRowCutDebugger Class Reference

Validate cuts against a known solution.

```
#include <OsiRowCutDebugger.hpp>
```

Public Member Functions

Validate Row Cuts

Check that the specified cuts do not cut off the known solution.

- virtual int [validateCuts](#) (const [OsiCuts](#) &cs, int first, int last) const
Check that the set of cuts does not cut off the solution known to the debugger.
- virtual bool [invalidCut](#) (const [OsiRowCut](#) &rowcut) const
Check that the cut does not cut off the solution known to the debugger.
- bool [onOptimalPath](#) (const [OsiSolverInterface](#) &si) const
Returns true if the solution held in the solver is compatible with the known solution.

Activate the Debugger

The debugger is considered to be active when it holds a known solution.

- bool [activate](#) (const [OsiSolverInterface](#) &si, const char *model)
Activate a debugger using the name of a problem.
- bool [activate](#) (const [OsiSolverInterface](#) &si, const double *solution, bool keepContinuous=false)
Activate a debugger using a full solution array.
- bool [active](#) () const
Returns true if the debugger is active.

Query or Manipulate the Known Solution

- `const double * optimalSolution () const`
Return the known solution.
- `int numberColumns () const`
Return the number of columns in the known solution.
- `double optimalValue () const`
Return the value of the objective for the known solution.
- `void redoSolution (int numberColumns, const int *originalColumns)`
Edit the known solution to reflect column changes.
- `int printOptimalSolution (const OsiSolverInterface &si) const`
Print optimal solution (returns -1 bad debug, 0 on optimal, 1 not)

Constructors and Destructors

- `OsiRowCutDebugger ()`
Default constructor - no checking.
- `OsiRowCutDebugger (const OsiSolverInterface &si, const char *model)`
Constructor with name of model.
- `OsiRowCutDebugger (const OsiSolverInterface &si, const double *solution, bool enforceOptimality=false)`
Constructor with full solution.
- `OsiRowCutDebugger (const OsiRowCutDebugger &)`
Copy constructor.
- `OsiRowCutDebugger & operator= (const OsiRowCutDebugger &rhs)`
Assignment operator.
- `virtual ~OsiRowCutDebugger ()`
Destructor.

Private Attributes

Private member data

- `double knownValue_`
Value of known solution.
- `int numberColumns_`
Number of columns in known solution.
- `bool * integerVariable_`
array specifying integer variables
- `double * knownSolution_`
array specifying known solution

Friends

- `void OsiRowCutDebuggerUnitTest (const OsiSolverInterface *siP, const std::string &mpsDir)`
A function that tests the methods in the `OsiRowCutDebugger` class.

8.29.1 Detailed Description

Validate cuts against a known solution.

`OsiRowCutDebugger` provides a facility for validating cuts against a known solution for a problem. The debugger knows an optimal solution for many of the miplib3 problems. Check the source for `activate(const OsiSolverInterface&,const char*)` in `OsiRowCutDebugger.cpp` for the full set of known problems.

A full solution vector can be supplied as a parameter with ([activate\(const OsiSolverInterface&,const double*,bool\)](#)). Only the integer values need to be valid. The default behaviour is to solve an lp relaxation with the integer variables fixed to the specified values and use the optimal solution to fill in the continuous variables in the solution. The debugger can be instructed to preserve the continuous variables (useful when debugging solvers where the linear relaxation doesn't capture all the constraints).

Note that the solution must match the problem held in the solver interface. If you want to use the row cut debugger on a problem after applying presolve transformations, your solution must match the presolved problem. (But see [redoSolution\(\)](#).)

Definition at line 42 of file OsiRowCutDebugger.hpp.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 OsiRowCutDebugger::OsiRowCutDebugger ()

Default constructor - no checking.

8.29.2.2 OsiRowCutDebugger::OsiRowCutDebugger (const OsiSolverInterface & *si*, const char * *model*)

Constructor with name of model.

See [activate\(const OsiSolverInterface&,const char*\)](#).

8.29.2.3 OsiRowCutDebugger::OsiRowCutDebugger (const OsiSolverInterface & *si*, const double * *solution*, bool *enforceOptimality* = false)

Constructor with full solution.

See [activate\(const OsiSolverInterface&,const double*,bool\)](#).

8.29.2.4 OsiRowCutDebugger::OsiRowCutDebugger (const OsiRowCutDebugger &)

Copy constructor.

8.29.2.5 virtual OsiRowCutDebugger::~OsiRowCutDebugger () [virtual]

Destructor.

8.29.3 Member Function Documentation

8.29.3.1 virtual int OsiRowCutDebugger::validateCuts (const OsiCuts & *cs*, int *first*, int *last*) const [virtual]

Check that the set of cuts does not cut off the solution known to the debugger.

Check if any generated cuts cut off the solution known to the debugger! If so then print offending cuts. Return the number of invalid cuts.

8.29.3.2 virtual bool OsiRowCutDebugger::invalidCut (const OsiRowCut & *rowcut*) const [virtual]

Check that the cut does not cut off the solution known to the debugger.

Return true if cut is invalid

8.29.3.3 bool OsiRowCutDebugger::onOptimalPath (const OsiSolverInterface & *si*) const

Returns true if the solution held in the solver is compatible with the known solution.

More specifically, returns true if the known solution satisfies the column bounds held in the solver.

8.29.3.4 `bool OsiRowCutDebugger::activate (const OsiSolverInterface & si, const char * model)`

Activate a debugger using the name of a problem.

The debugger knows an optimal solution for most of miplib3. Check the source code for the full list. Returns true if the debugger is successfully activated.

8.29.3.5 `bool OsiRowCutDebugger::activate (const OsiSolverInterface & si, const double * solution, bool keepContinuous = false)`

Activate a debugger using a full solution array.

The solution must have one entry for every variable, but only the entries for integer values are used. By default the debugger will solve an lp relaxation with the integer variables fixed and fill in values for the continuous variables from this solution. If the debugger should preserve the given values for the continuous variables, set `keepContinuous` to `true`.

Returns true if debugger activates successfully.

8.29.3.6 `bool OsiRowCutDebugger::active () const`

Returns true if the debugger is active.

8.29.3.7 `const double* OsiRowCutDebugger::optimalSolution () const [inline]`

Return the known solution.

Definition at line 111 of file `OsiRowCutDebugger.hpp`.

8.29.3.8 `int OsiRowCutDebugger::numberColumns () const [inline]`

Return the number of columns in the known solution.

Definition at line 115 of file `OsiRowCutDebugger.hpp`.

8.29.3.9 `double OsiRowCutDebugger::optimalValue () const [inline]`

Return the value of the objective for the known solution.

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

8.29.3.10 `void OsiRowCutDebugger::redoSolution (int numberColumns, const int * originalColumns)`

Edit the known solution to reflect column changes.

Given a translation array `originalColumns[numberColumns]` which can translate current column indices to original column indices, this method will edit the solution held in the debugger so that it matches the current set of columns.

Useful when the original problem is preprocessed prior to cut generation. The debugger does keep a record of the changes.

8.29.3.11 `int OsiRowCutDebugger::printOptimalSolution (const OsiSolverInterface & si) const`

Print optimal solution (returns -1 bad debug, 0 on optimal, 1 not)

8.29.3.12 `OsiRowCutDebugger& OsiRowCutDebugger::operator= (const OsiRowCutDebugger & rhs)`

Assignment operator.

8.29.4 Friends And Related Function Documentation

8.29.4.1 `void OsiRowCutDebuggerUnitTest (const OsiSolverInterface * siP, const std::string & mpsDir)` `[friend]`

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

8.29.5 Member Data Documentation

8.29.5.1 `double OsiRowCutDebugger::knownValue_` `[private]`

Value of known solution.

Definition at line 171 of file [OsiRowCutDebugger.hpp](#).

8.29.5.2 `int OsiRowCutDebugger::numberColumns_` `[private]`

Number of columns in known solution.

This must match the number of columns reported by the solver.

Definition at line 177 of file [OsiRowCutDebugger.hpp](#).

8.29.5.3 `bool* OsiRowCutDebugger::integerVariable_` `[private]`

array specifying integer variables

Definition at line 180 of file [OsiRowCutDebugger.hpp](#).

8.29.5.4 `double* OsiRowCutDebugger::knownSolution_` `[private]`

array specifying known solution

Definition at line 183 of file [OsiRowCutDebugger.hpp](#).

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

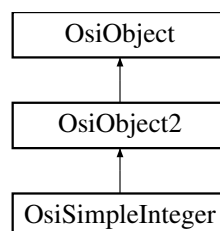
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiRowCutDebugger.hpp](#)

8.30 OsiSimpleInteger Class Reference

Define a single integer class.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for [OsiSimpleInteger](#):



Public Member Functions

- [OsiSimpleInteger](#) ()
Default Constructor.
- [OsiSimpleInteger](#) (const [OsiSolverInterface](#) *solver, int iColumn)
Useful constructor - passed solver index.
- [OsiSimpleInteger](#) (int iColumn, double lower, double upper)
Useful constructor - passed solver index and original bounds.
- [OsiSimpleInteger](#) (const [OsiSimpleInteger](#) &)
Copy constructor.
- virtual [OsiObject](#) * [clone](#) () const
Clone.
- [OsiSimpleInteger](#) & [operator=](#) (const [OsiSimpleInteger](#) &rhs)
Assignment operator.
- virtual [~OsiSimpleInteger](#) ()
Destructor.
- virtual double [infeasibility](#) (const [OsiBranchingInformation](#) *info, int &[whichWay](#)) const
Infeasibility - large is 0.5.
- virtual double [feasibleRegion](#) ([OsiSolverInterface](#) *solver, const [OsiBranchingInformation](#) *info) const
Set bounds to fix the variable at the current (integer) value.
- virtual [OsiBranchingObject](#) * [createBranch](#) ([OsiSolverInterface](#) *solver, const [OsiBranchingInformation](#) *info, int way) const
Creates a branching object.
- void [setColumnNumber](#) (int value)
Set solver column number.
- virtual int [columnNumber](#) () const
Column number if single column object -1 otherwise, so returns ≥ 0 Used by heuristics.
- double [originalLowerBound](#) () const
Original bounds.
- void [setOriginalLowerBound](#) (double value)
- double [originalUpperBound](#) () const
- void [setOriginalUpperBound](#) (double value)
- virtual void [resetBounds](#) (const [OsiSolverInterface](#) *solver)
Reset variable bounds to their original values.
- virtual void [resetSequenceEtc](#) (int numberColumns, const int *originalColumns)
Change column numbers after preprocessing.
- virtual double [upEstimate](#) () const
Return "up" estimate (default 1.0e-5)
- virtual double [downEstimate](#) () const
Return "down" estimate (default 1.0e-5)
- virtual bool [canHandleShadowPrices](#) () const
Return true if knows how to deal with Pseudo Shadow Prices.

Protected Attributes

- double [originalLower_](#)
data Original lower bound
- double [originalUpper_](#)
Original upper bound.
- int [columnNumber_](#)
Column number in solver.

8.30.1 Detailed Description

Define a single integer class.

Definition at line 511 of file OsiBranchingObject.hpp.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 OsiSimpleInteger::OsiSimpleInteger ()

Default Constructor.

8.30.2.2 OsiSimpleInteger::OsiSimpleInteger (const OsiSolverInterface * *solver*, int *iColumn*)

Useful constructor - passed solver index.

8.30.2.3 OsiSimpleInteger::OsiSimpleInteger (int *iColumn*, double *lower*, double *upper*)

Useful constructor - passed solver index and original bounds.

8.30.2.4 OsiSimpleInteger::OsiSimpleInteger (const OsiSimpleInteger &)

Copy constructor.

8.30.2.5 virtual OsiSimpleInteger::~OsiSimpleInteger () [virtual]

Destructor.

8.30.3 Member Function Documentation

8.30.3.1 virtual OsiObject* OsiSimpleInteger::clone () const [virtual]

Clone.

Implements [OsiObject](#).

8.30.3.2 OsiSimpleInteger& OsiSimpleInteger::operator= (const OsiSimpleInteger & *rhs*)

Assignment operator.

8.30.3.3 virtual double OsiSimpleInteger::infeasibility (const OsiBranchingInformation * *info*, int & *whichWay*) const [virtual]

Infeasibility - large is 0.5.

Implements [OsiObject](#).

8.30.3.4 `virtual double OsiSimpleInteger::feasibleRegion (OsiSolverInterface * solver, const OsiBranchingInformation * info) const` `[virtual]`

Set bounds to fix the variable at the current (integer) value.

Given an integer value, set the lower and upper bounds to fix the variable. Returns amount it had to move variable.

Implements [OsiObject](#).

8.30.3.5 `virtual OsiBranchingObject* OsiSimpleInteger::createBranch (OsiSolverInterface * solver, const OsiBranchingInformation * info, int way) const` `[virtual]`

Creates a branching object.

The preferred direction is set by `way`, 0 for down, 1 for up.

Reimplemented from [OsiObject](#).

8.30.3.6 `void OsiSimpleInteger::setColumnNumber (int value)` `[inline]`

Set solver column number.

Definition at line 556 of file `OsiBranchingObject.hpp`.

8.30.3.7 `virtual int OsiSimpleInteger::columnNumber () const` `[virtual]`

Column number if single column object -1 otherwise, so returns ≥ 0 Used by heuristics.

Reimplemented from [OsiObject](#).

8.30.3.8 `double OsiSimpleInteger::originalLowerBound () const` `[inline]`

Original bounds.

Definition at line 566 of file `OsiBranchingObject.hpp`.

8.30.3.9 `void OsiSimpleInteger::setOriginalLowerBound (double value)` `[inline]`

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

8.30.3.10 `double OsiSimpleInteger::originalUpperBound () const` `[inline]`

Definition at line 570 of file `OsiBranchingObject.hpp`.

8.30.3.11 `void OsiSimpleInteger::setOriginalUpperBound (double value)` `[inline]`

Definition at line 572 of file `OsiBranchingObject.hpp`.

8.30.3.12 `virtual void OsiSimpleInteger::resetBounds (const OsiSolverInterface * solver)` `[virtual]`

Reset variable bounds to their original values.

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

Reimplemented from [OsiObject](#).

8.30.3.13 `virtual void OsiSimpleInteger::resetSequenceEtc (int numberColumns, const int * originalColumns)` `[virtual]`

Change column numbers after preprocessing.

Reimplemented from [OsiObject](#).

8.30.3.14 `virtual double OsiSimpleInteger::upEstimate () const [virtual]`

Return "up" estimate (default 1.0e-5)

Reimplemented from [OsiObject](#).

8.30.3.15 `virtual double OsiSimpleInteger::downEstimate () const [virtual]`

Return "down" estimate (default 1.0e-5)

Reimplemented from [OsiObject](#).

8.30.3.16 `virtual bool OsiSimpleInteger::canHandleShadowPrices () const [inline],[virtual]`

Return true if knows how to deal with Pseudo Shadow Prices.

Reimplemented from [OsiObject](#).

Definition at line 588 of file `OsiBranchingObject.hpp`.

8.30.4 Member Data Documentation

8.30.4.1 `double OsiSimpleInteger::originalLower_ [protected]`

data Original lower bound

Definition at line 593 of file `OsiBranchingObject.hpp`.

8.30.4.2 `double OsiSimpleInteger::originalUpper_ [protected]`

Original upper bound.

Definition at line 595 of file `OsiBranchingObject.hpp`.

8.30.4.3 `int OsiSimpleInteger::columnNumber_ [protected]`

Column number in solver.

Definition at line 597 of file `OsiBranchingObject.hpp`.

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

- `/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp`

8.31 OsiSolverBranch Class Reference

Solver Branch Class.

```
#include <OsiSolverBranch.hpp>
```

Public Member Functions

Add and Get methods

- void [addBranch](#) (int iColumn, double value)
Add a simple branch (i.e. first sets ub of floor(value), second lb of ceil(value))
- void [addBranch](#) (int way, int numberTighterLower, const int *whichLower, const double *newLower, int numberTighterUpper, const int *whichUpper, const double *newUpper)

- *Add bounds - way = -1 is first , +1 is second.*
- void `addBranch` (int way, int numberColumns, const double *oldLower, const double *newLower, const double *oldUpper, const double *newUpper)
- *Add bounds - way = -1 is first , +1 is second.*
- void `applyBounds` (`OsiSolverInterface` &solver, int way) const
- *Apply bounds.*
- bool `feasibleOneWay` (const `OsiSolverInterface` &solver) const
- *Returns true if current solution satisfies one side of branch.*
- const int * `starts` () const
- *Starts.*
- const int * `which` () const
- *Which variables.*
- const double * `bounds` () const
- *Bounds.*

Constructors and destructors

- `OsiSolverBranch` ()
- *Default Constructor.*
- `OsiSolverBranch` (const `OsiSolverBranch` &rhs)
- *Copy constructor.*
- `OsiSolverBranch` & `operator=` (const `OsiSolverBranch` &rhs)
- *Assignment operator.*
- `~OsiSolverBranch` ()
- *Destructor.*

Private Attributes

Private member data

- int `start_` [5]
- *Start of lower first, upper first, lower second, upper second.*
- int * `indices_`
- *Column numbers (if >= numberColumns treat as rows)*
- double * `bound_`
- *New bounds.*

8.31.1 Detailed Description

Solver Branch Class.

This provides information on a branch as a set of tighter bounds on both ways

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

8.31.2 Constructor & Destructor Documentation

8.31.2.1 `OsiSolverBranch::OsiSolverBranch ()`

Default Constructor.

8.31.2.2 `OsiSolverBranch::OsiSolverBranch (const OsiSolverBranch & rhs)`

Copy constructor.

8.31.2.3 OsiSolverBranch::~~OsiSolverBranch ()

Destructor.

8.31.3 Member Function Documentation

8.31.3.1 void OsiSolverBranch::addBranch (int *iColumn*, double *value*)

Add a simple branch (i.e. first sets ub of floor(value), second lb of ceil(value))

8.31.3.2 void OsiSolverBranch::addBranch (int *way*, int *numberTighterLower*, const int * *whichLower*, const double * *newLower*, int *numberTighterUpper*, const int * *whichUpper*, const double * *newUpper*)

Add bounds - way == -1 is first , +1 is second.

8.31.3.3 void OsiSolverBranch::addBranch (int *way*, int *numberColumns*, const double * *oldLower*, const double * *newLower*, const double * *oldUpper*, const double * *newUpper*)

Add bounds - way == -1 is first , +1 is second.

8.31.3.4 void OsiSolverBranch::applyBounds (OsiSolverInterface & *solver*, int *way*) const

Apply bounds.

8.31.3.5 bool OsiSolverBranch::feasibleOneWay (const OsiSolverInterface & *solver*) const

Returns true if current solution satisfies one side of branch.

8.31.3.6 const int* OsiSolverBranch::starts () const [inline]

Starts.

Definition at line 38 of file OsiSolverBranch.hpp.

8.31.3.7 const int* OsiSolverBranch::which () const [inline]

Which variables.

Definition at line 41 of file OsiSolverBranch.hpp.

8.31.3.8 const double* OsiSolverBranch::bounds () const [inline]

Bounds.

Definition at line 44 of file OsiSolverBranch.hpp.

8.31.3.9 OsiSolverBranch& OsiSolverBranch::operator= (const OsiSolverBranch & *rhs*)

Assignment operator.

8.31.4 Member Data Documentation

8.31.4.1 int OsiSolverBranch::start_[5] [private]

Start of lower first, upper first, lower second, upper second.

Definition at line 69 of file OsiSolverBranch.hpp.

8.31.4.2 `int* OsiSolverBranch::indices_ [private]`

Column numbers (if \geq numberColumns treat as rows)

Definition at line 71 of file OsiSolverBranch.hpp.

8.31.4.3 `double* OsiSolverBranch::bound_ [private]`

New bounds.

Definition at line 73 of file OsiSolverBranch.hpp.

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

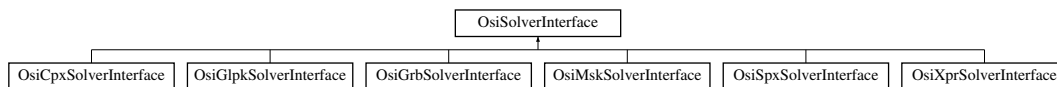
- </home/ted/COIN/trunk/Osi/src/Osi/OsiSolverBranch.hpp>

8.32 OsiSolverInterface Class Reference

Abstract Base Class for describing an interface to a solver.

```
#include <OsiSolverInterface.hpp>
```

Inheritance diagram for OsiSolverInterface:



Classes

- class [ApplyCutsReturnCode](#)
Internal class for obtaining status from the applyCuts method.

Public Types

- typedef `std::vector< std::string >` [OsiNameVec](#)
Data type for name vectors.

Public Member Functions

Solve methods

- virtual void [initialSolve](#) ()=0
Solve initial LP relaxation.
- virtual void [resolve](#) ()=0
Resolve an LP relaxation after problem modification.
- virtual void [branchAndBound](#) ()=0
Invoke solver's built-in enumeration algorithm.

Parameter set/get methods

The set methods return true if the parameter was set to the given value, false otherwise.

When a set method returns false, the original value (if any) should be unchanged. There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the volume algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.

The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.

Note

There is a default implementation of the set/get methods, namely to store/retrieve the given value using an array in the base class. A specific solver implementation can use this feature, for example, to store parameters that should be used later on. Implementors of a solver interface should overload these functions to provide the proper interface to and accurately reflect the capabilities of a specific solver.

The format for hints is slightly different in that a boolean specifies the sense of the hint and an enum specifies the strength of the hint. Hints should be initialised when a solver is instantiated. (See [OsiSolverParameters.hpp](#) for defined hint parameters and strength.) When specifying the sense of the hint, a value of true means to work with the hint, false to work against it. For example,

- `setHintParam(OsiDoScale, true, OsiHintTry)`
is a mild suggestion to the solver to scale the constraint system.
- `setHintParam(OsiDoScale, false, OsiForceDo)`
tells the solver to disable scaling, or throw an exception if it cannot comply.

As another example, a solver interface could use the value and strength of the `OsiDoReducePrint` hint to adjust the amount of information printed by the interface and/or solver. The extent to which a solver obeys hints is left to the solver. The value and strength returned by `getHintParam` will match the most recent call to `setHintParam`, and will not necessarily reflect the solver's ability to comply with the hint. If the hint strength is `OsiForceDo`, the solver is required to throw an exception if it cannot perform the specified action.

Note

As with the other set/get methods, there is a default implementation which maintains arrays in the base class for hint sense and strength. The default implementation does not store the `otherInformation` pointer, and always throws an exception for strength `OsiForceDo`. Implementors of a solver interface should override these functions to provide the proper interface to and accurately reflect the capabilities of a specific solver.

- virtual bool `setIntParam (OsiIntParam key, int value)`
Set an integer parameter.
- virtual bool `setDbiParam (OsiDbiParam key, double value)`
Set a double parameter.
- virtual bool `setStrParam (OsiStrParam key, const std::string &value)`
Set a string parameter.
- virtual bool `setHintParam (OsiHintParam key, bool yesNo=true, OsiHintStrength strength=OsiHintTry, void *N=NULL)`
Set a hint parameter.
- virtual bool `getIntParam (OsiIntParam key, int &value) const`
Get an integer parameter.
- virtual bool `getDbiParam (OsiDbiParam key, double &value) const`
Get a double parameter.
- virtual bool `getStrParam (OsiStrParam key, std::string &value) const`
Get a string parameter.
- virtual bool `getHintParam (OsiHintParam key, bool &yesNo, OsiHintStrength &strength, void *&otherInformation) const`
Get a hint parameter (all information)
- virtual bool `getHintParam (OsiHintParam key, bool &yesNo, OsiHintStrength &strength) const`
Get a hint parameter (sense and strength only)

- virtual bool `getHintParam` (`OsiHintParam` key, bool &yesNo) const
Get a hint parameter (sense only)
- void `copyParameters` (`OsiSolverInterface` &rhs)
Copy all parameters in this section from one solver to another.
- double `getIntegerTolerance` () const
Return the integrality tolerance of the underlying solver.

Methods returning info on how the solution process terminated

- virtual bool `isAbandoned` () const =0
Are there numerical difficulties?
- virtual bool `isProvenOptimal` () const =0
Is optimality proven?
- virtual bool `isProvenPrimalInfeasible` () const =0
Is primal infeasibility proven?
- virtual bool `isProvenDualInfeasible` () const =0
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 =0
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 =0
Get an empty warm start object.
- virtual `CoinWarmStart` * `getWarmStart` () const =0
Get warm start information.
- virtual `CoinWarmStart` * `getPointerToWarmStart` (bool &mustDelete)
Get warm start information.
- virtual bool `setWarmStart` (const `CoinWarmStart` *warmstart)=0
Set warm start information.

Hot start methods

Primarily used in strong branching.

The user can create a hot start object — a snapshot of the optimization process — then reoptimize over and over again, starting from the same point.

Note

- *Between hot started optimizations only bound changes are allowed.*
- *The copy constructor and assignment operator should NOT copy any hot start information.*
- *The default implementation simply extracts a warm start object in `markHotStart`, resets to the warm start object in `solveFromHotStart`, and deletes the warm start object in `unmarkHotStart`. Actual solver implementations are encouraged to do better.*
- virtual void `markHotStart` ()
Create a hot start snapshot of the optimization process.

- virtual void [solveFromHotStart](#) ()
Optimize starting from the hot start snapshot.
- virtual void [unmarkHotStart](#) ()
Delete the hot start snapshot.

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 =0
Get the number of columns.
- virtual int [getNumRows](#) () const =0
Get the number of rows.
- virtual int [getNumElements](#) () const =0
Get the number of nonzero elements.
- virtual int [getNumIntegers](#) () const
Get the number of integer variables.
- virtual const double * [getColLower](#) () const =0
Get a pointer to an array[getNumCols()] of column lower bounds.
- virtual const double * [getColUpper](#) () const =0
Get a pointer to an array[getNumCols()] of column upper bounds.
- virtual const char * [getRowSense](#) () const =0
Get a pointer to an array[getNumRows()] of row constraint senses.
- virtual const double * [getRightHandSide](#) () const =0
Get a pointer to an array[getNumRows()] of row right-hand sides.
- virtual const double * [getRowRange](#) () const =0
Get a pointer to an array[getNumRows()] of row ranges.
- virtual const double * [getRowLower](#) () const =0
Get a pointer to an array[getNumRows()] of row lower bounds.
- virtual const double * [getRowUpper](#) () const =0
Get a pointer to an array[getNumRows()] of row upper bounds.
- virtual const double * [getObjCoefficients](#) () const =0
Get a pointer to an array[getNumCols()] of objective function coefficients.
- virtual double [getObjSense](#) () const =0
Get the objective function sense.
- virtual bool [isContinuous](#) (int colIndex) const =0
Return true if the variable is continuous.
- virtual bool [isBinary](#) (int colIndex) const
Return true if the variable is binary.
- virtual bool [isInteger](#) (int colIndex) const
Return true if the variable is integer.
- virtual bool [isIntegerNonBinary](#) (int colIndex) const
Return true if the variable is general integer.
- virtual bool [isFreeBinary](#) (int colIndex) const
Return true if the variable is binary and not fixed.
- const char * [columnType](#) (bool refresh=false) const
Return an array[getNumCols()] of column types.
- virtual const char * [getColType](#) (bool refresh=false) const
Return an array[getNumCols()] of column types.
- virtual const CoinPackedMatrix * [getMatrixByRow](#) () const =0
Get a pointer to a row-wise copy of the matrix.
- virtual const CoinPackedMatrix * [getMatrixByCol](#) () const =0

- *Get a pointer to a column-wise copy of the matrix.*
- virtual CoinPackedMatrix * [getMutableMatrixByRow](#) () const
- *Get a pointer to a mutable row-wise copy of the matrix.*
- virtual CoinPackedMatrix * [getMutableMatrixByCol](#) () const
- *Get a pointer to a mutable column-wise copy of the matrix.*
- virtual double [getInfinity](#) () const =0
- *Get the solver's value for infinity.*

Solution query methods

- virtual const double * [getColSolution](#) () const =0
- *Get a pointer to an array[[getNumCols\(\)](#)] of primal variable values.*
- virtual const double * [getStrictColSolution](#) ()
- *Get a pointer to an array[[getNumCols\(\)](#)] of primal variable values guaranteed to be between the column lower and upper bounds.*
- virtual const double * [getRowPrice](#) () const =0
- *Get pointer to array[[getNumRows\(\)](#)] of dual variable values.*
- virtual const double * [getReducedCost](#) () const =0
- *Get a pointer to an array[[getNumCols\(\)](#)] of reduced costs.*
- virtual const double * [getRowActivity](#) () const =0
- *Get a pointer to array[[getNumRows\(\)](#)] of row activity levels.*
- virtual double [getObjValue](#) () const =0
- *Get the objective function value.*
- virtual int [getIterationCount](#) () const =0
- *Get the number of iterations it took to solve the problem (whatever 'iteration' means to the solver).*
- virtual std::vector< double * > [getDualRays](#) (int maxNumRays, bool fullRay=false) const =0
- *Get as many dual rays as the solver can provide.*
- virtual std::vector< double * > [getPrimalRays](#) (int maxNumRays) const =0
- *Get as many primal rays as the solver can provide.*
- virtual [OsiVectorInt](#) [getFractionalIndices](#) (const double etol=1.e-05) const
- *Get vector of indices of primal variables which are integer variables but have fractional values in the current solution.*

Methods to modify the objective, bounds, and solution

For functions which take a set of indices as parameters ([setObjCoeffSet](#) (), [setColSetBounds](#) (), [setRowSetBounds](#) (), [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)=0
- *Set an objective function coefficient.*
- virtual void [setObjCoeffSet](#) (const int *indexFirst, const int *indexLast, const double *coeffList)
- *Set a set of objective function coefficients.*
- virtual void [setObjective](#) (const double *array)
- *Set the objective coefficients for all columns.*
- virtual void [setObjSense](#) (double s)=0
- *Set the objective function sense.*
- virtual void [setColLower](#) (int elementIndex, double elementValue)=0
- *Set a single column lower bound.*
- virtual void [setColLower](#) (const double *array)
- *Set the lower bounds for all columns.*
- virtual void [setColUpper](#) (int elementIndex, double elementValue)=0
- *Set a single column upper bound.*
- virtual void [setColUpper](#) (const double *array)
- *Set the upper bounds for all columns.*

- virtual void [setColBounds](#) (int elementIndex, double lower, double upper)
Set a single column lower and upper bound.
- virtual void [setColSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the upper and lower bounds of a set of columns.
- virtual void [setRowLower](#) (int elementIndex, double elementValue)=0
Set a single row lower bound.
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)=0
Set a single row upper bound.
- virtual void [setRowBounds](#) (int elementIndex, double lower, double upper)
Set a single row lower and upper bound.
- virtual void [setRowSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a set of rows.
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)=0
Set the type of a single row.
- virtual void [setRowSetTypes](#) (const int *indexFirst, const int *indexLast, const char *senseList, const double *rhsList, const double *rangeList)
Set the type of a set of rows.
- virtual void [setColSolution](#) (const double *colsol)=0
Set the primal solution variable values.
- virtual void [setRowPrice](#) (const double *rowprice)=0
Set dual solution variable values.
- virtual int [reducedCostFix](#) (double gap, bool justInteger=true)
Fix variables at bound based on reduced cost.

Methods to set variable type

- virtual void [setContinuous](#) (int index)=0
Set the index-th variable to be a continuous variable.
- virtual void [setInteger](#) (int index)=0
Set the index-th variable to be an integer variable.
- virtual void [setContinuous](#) (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be continuous variables.
- virtual void [setInteger](#) (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be integer variables.

Methods for row and column names

Osi defines three name management disciplines: 'auto names' (0), 'lazy names' (1), and 'full names' (2).

See the description of [OsiNameDiscipline](#) for details. Changing the name discipline (via [setIntParam\(\)](#)) will not automatically add or remove name information, but setting the discipline to auto will make existing information inaccessible until the discipline is reset to lazy or full.

By definition, a row index of [getNumRows\(\)](#) (i.e., one larger than the largest valid row index) refers to the objective function.

OSI users and implementors: While the OSI base class can define an interface and provide rudimentary support, use of names really depends on support by the OsiXXX class to ensure that names are managed correctly. If an OsiXXX class does not support names, it should return false for calls to [getIntParam\(\)](#) or [setIntParam\(\)](#) that reference [OsiNameDiscipline](#).

- virtual std::string [dfltRowColName](#) (char rc, int ndx, unsigned digits=7) const
Generate a standard name of the form Rnnnnnnn or Cnnnnnnn.
- virtual std::string [getObjName](#) (unsigned maxlen=static_cast< unsigned >(std::string::npos)) const
Return the name of the objective function.
- virtual void [setObjName](#) (std::string name)
Set the name of the objective function.

- virtual std::string [getRowName](#) (int rowIndex, unsigned maxlen=static_cast< unsigned >(std::string::npos)) const
Return the name of the row.
- virtual const [OsiNameVec](#) & [getRowNames](#) ()
Return a pointer to a vector of row names.
- virtual void [setRowName](#) (int ndx, std::string name)
Set a row name.
- virtual void [setRowNames](#) ([OsiNameVec](#) &srcNames, int srcStart, int len, int tgtStart)
Set multiple row names.
- virtual void [deleteRowNames](#) (int tgtStart, int len)
Delete len row names starting at index tgtStart.
- virtual std::string [getColName](#) (int colIndex, unsigned maxlen=static_cast< unsigned >(std::string::npos)) const
Return the name of the column.
- virtual const [OsiNameVec](#) & [getColNames](#) ()
Return a pointer to a vector of column names.
- virtual void [setColName](#) (int ndx, std::string name)
Set a column name.
- virtual void [setColNames](#) ([OsiNameVec](#) &srcNames, int srcStart, int len, int tgtStart)
Set multiple column names.
- virtual void [deleteColNames](#) (int tgtStart, int len)
Delete len column names starting at index tgtStart.
- void [setRowColNames](#) (const CoinMpsIO &mps)
Set row and column names from a CoinMpsIO object.
- void [setRowColNames](#) (CoinModel &mod)
Set row and column names from a CoinModel object.
- void [setRowColNames](#) (CoinLpIO &mod)
Set row and column names from a CoinLpIO object.

Methods to modify the constraint system.

Note that new columns are added as continuous variables.

- virtual void [addCol](#) (const CoinPackedVectorBase &vec, const double collb, const double colub, const double obj)=0
Add a column (primal variable) to the problem.
- virtual void [addCol](#) (const CoinPackedVectorBase &vec, const double collb, const double colub, const double obj, std::string name)
Add a named column (primal variable) to the problem.
- virtual void [addCol](#) (int numberElements, const int *rows, const double *elements, const double collb, const double colub, const double obj)
Add a column (primal variable) to the problem.
- virtual void [addCol](#) (int numberElements, const int *rows, const double *elements, const double collb, const double colub, const double obj, std::string name)
Add a named column (primal variable) to the problem.
- virtual void [addCols](#) (const int numcols, const CoinPackedVectorBase *const *cols, const double *collb, const double *colub, const double *obj)
Add a set of columns (primal variables) to the problem.
- virtual void [addCols](#) (const int numcols, const int *columnStarts, const int *rows, const double *elements, const double *collb, const double *colub, const double *obj)
Add a set of columns (primal variables) to the problem.
- void [addCols](#) (const CoinBuild &buildObject)
Add columns using a CoinBuild object.
- int [addCols](#) (CoinModel &modelObject)
Add columns from a model object.

- virtual void [deleteCols](#) (const int num, const int *colIndices)=0
Remove a set of columns (primal variables) from the problem.
- virtual void [addRow](#) (const CoinPackedVectorBase &vec, const double rowlb, const double rowub)=0
Add a row (constraint) to the problem.
- virtual void [addRow](#) (const CoinPackedVectorBase &vec, const double rowlb, const double rowub, std::string name)
Add a named row (constraint) to the problem.
- virtual void [addRow](#) (const CoinPackedVectorBase &vec, const char rowsen, const double rowrhs, const double rowrng)=0
Add a row (constraint) to the problem.
- virtual void [addRow](#) (const CoinPackedVectorBase &vec, const char rowsen, const double rowrhs, const double rowrng, std::string name)
Add a named row (constraint) to the problem.
- virtual void [addRow](#) (int numberElements, const int *columns, const double *element, const double rowlb, const double rowub)
Add a row (constraint) to the problem.
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase *const *rows, const double *rowlb, const double *rowub)
Add a set of rows (constraints) to the problem.
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase *const *rows, const char *rowsen, const double *rowrhs, const double *rowrng)
Add a set of rows (constraints) to the problem.
- virtual void [addRows](#) (const int numRows, const int *rowStarts, const int *columns, const double *element, const double *rowlb, const double *rowub)
Add a set of rows (constraints) to the problem.
- void [addRows](#) (const CoinBuild &buildObject)
Add rows using a CoinBuild object.
- int [addRows](#) (CoinModel &modelObject)
Add rows from a CoinModel object.
- virtual void [deleteRows](#) (const int num, const int *rowIndices)=0
Delete a set of rows (constraints) from the problem.
- virtual void [replaceMatrixOptional](#) (const CoinPackedMatrix &)
Replace the constraint matrix.
- virtual void [replaceMatrix](#) (const CoinPackedMatrix &)
Replace the constraint matrix.
- virtual void [saveBaseModel](#) ()
Save a copy of the base model.
- virtual void [restoreBaseModel](#) (int numberOfRows)
Reduce the constraint system to the specified number of constraints.
- virtual [ApplyCutsReturnCode](#) [applyCuts](#) (const [OsiCuts](#) &cs, double effectivenessLb=0.0)
Apply a collection of cuts.
- virtual void [applyRowCuts](#) (int numberCuts, const [OsiRowCut](#) *cuts)
Apply a collection of row cuts which are all effective.
- virtual void [applyRowCuts](#) (int numberCuts, const [OsiRowCut](#) **cuts)
Apply a collection of row cuts which are all effective.
- void [deleteBranchingInfo](#) (int numberDeleted, const int *which)
Deletes branching information before columns deleted.

Methods for problem input and output

- virtual void [loadProblem](#) (const CoinPackedMatrix &matrix, const double *collb, const double *colub, const double *obj, const double *rowlb, const double *rowub)=0
Load in a problem by copying the arguments.
- virtual void [assignProblem](#) (CoinPackedMatrix * &matrix, double * &collb, double * &colub, double * &obj, double * &rowlb, double * &rowub)=0

- Load in a problem by assuming ownership of the arguments.*
 - virtual void [loadProblem](#) (const CoinPackedMatrix &matrix, const double *collb, const double *colub, const double *obj, const char *rowsen, const double *rowrhs, const double *rowrng)=0
 - Load in a problem by copying the arguments.*
 - virtual void [assignProblem](#) (CoinPackedMatrix * &matrix, double * &collb, double * &colub, double * &obj, char * &rowsen, double * &rowrhs, double * &rowrng)=0
 - Load in a problem by assuming ownership of the arguments.*
 - 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)=0
 - Load in a problem by copying the arguments.*
 - 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)=0
 - Load in a problem by copying the arguments.*
 - virtual int [loadFromCoinModel](#) (CoinModel &modelObject, bool keepSolution=false)
 - Load a model from a CoinModel object.*
 - virtual int [readMps](#) (const char *filename, const char *extension="mps")
 - Read a problem in MPS format from the given filename.*
 - virtual int [readMps](#) (const char *filename, const char *extension, int &numberSets, CoinSet **&sets)
 - Read a problem in MPS format from the given full filename.*
 - virtual int [readGMPL](#) (const char *filename, const char *dataname=NULL)
 - Read a problem in GMPL format from the given filenames.*
 - virtual void [writeMps](#) (const char *filename, const char *extension="mps", double objSense=0.0) const =0
 - Write the problem in MPS format to the specified file.*
 - int [writeMpsNative](#) (const char *filename, const char **rowNames, const char **columnNames, int formatType=0, int numberAcross=2, double objSense=0.0, int numberSOS=0, const CoinSet *setInfo=NULL) const
 - Write the problem in MPS format to the specified file with more control over the output.*
 - virtual void [writeLp](#) (const char *filename, const char *extension="lp", double epsilon=1e-5, int numberAcross=10, int decimals=5, double objSense=0.0, bool useRowNames=true) const
 - Write the problem into an Lp file of the given filename with the specified extension.*
 - virtual void [writeLp](#) (FILE *fp, double epsilon=1e-5, int numberAcross=10, int decimals=5, double objSense=0.0, bool useRowNames=true) const
 - Write the problem into the file pointed to by the parameter fp.*
 - int [writeLpNative](#) (const char *filename, char const *const *const rowNames, char const *const *const columnNames, const double epsilon=1.0e-5, const int numberAcross=10, const int decimals=5, const double objSense=0.0, const bool useRowNames=true) const
 - Write the problem into an Lp file.*
 - int [writeLpNative](#) (FILE *fp, char const *const *const rowNames, char const *const *const columnNames, const double epsilon=1.0e-5, const int numberAcross=10, const int decimals=5, const double objSense=0.0, const bool useRowNames=true) const
 - Write the problem into the file pointed to by the parameter fp.*
 - virtual int [readLp](#) (const char *filename, const double epsilon=1e-5)
 - Read file in LP format from file with name filename.*
 - int [readLp](#) (FILE *fp, const double epsilon=1e-5)
 - Read file in LP format from the file pointed to by fp.*

Setting/Accessing application data

- void [setApplicationData](#) (void *appData)
 - Set application data.*
- void [setAuxiliaryInfo](#) (OsiAuxInfo *auxiliaryInfo)
 - Create a clone of an Auxiliary Information object.*
- void * [getApplicationData](#) () const

- *Get application data.*
- `OsiAuxInfo * getAuxiliaryInfo ()` const
Get pointer to auxiliary info object.

Message handling

See the COIN library documentation for additional information about COIN message facilities.

- virtual void `passInMessageHandler` (CoinMessageHandler *handler)
Pass in a message handler.
- void `newLanguage` (CoinMessages::Language language)
Set language.
- void `setLanguage` (CoinMessages::Language language)
- CoinMessageHandler * `messageHandler` () const
Return a pointer to the current message handler.
- CoinMessages `messages` ()
Return the current set of messages.
- CoinMessages * `messagesPointer` ()
Return a pointer to the current set of messages.
- bool `defaultHandler` () const
Return true if default handler.

Methods for dealing with discontinuities other than integers.

Osi should be able to know about SOS and other types.

This is an optional section where such information can be stored.

- void `findIntegers` (bool justCount)
Identify integer variables and create corresponding objects.
- virtual int `findIntegersAndSOS` (bool justCount)
Identify integer variables and SOS and create corresponding objects.
- int `numberObjects` () const
Get the number of objects.
- void `setNumberObjects` (int number)
Set the number of objects.
- `OsiObject ** objects` () const
Get the array of objects.
- const `OsiObject * object` (int which) const
Get the specified object.
- `OsiObject * modifiableObject` (int which) const
Get the specified object.
- void `deleteObjects` ()
Delete all object information.
- void `addObjects` (int numberObjects, `OsiObject **objects`)
Add in object information.
- double `forceFeasible` ()
Use current solution to set bounds so current integer feasible solution will stay feasible.

Methods related to testing generated cuts

See the documentation for `OsiRowCutDebugger` for additional details.

- virtual void `activateRowCutDebugger` (const char *modelName)
Activate the row cut debugger.
- virtual void `activateRowCutDebugger` (const double *solution, bool enforceOptimality=true)
Activate the row cut debugger using a full solution array.
- const `OsiRowCutDebugger * getRowCutDebugger` () const

- *Get the row cut debugger provided the solution known to the debugger is within the feasible region held in the solver.*
- `OsiRowCutDebugger * getRowCutDebuggerAlways () const`
Get the row cut debugger object.

OsiSimplexInterface

Simplex Interface

Methods for an advanced interface to a simplex solver. The interface comprises two groups of methods. Group 1 contains methods for tableau access. Group 2 contains methods for dictating individual simplex pivots.

- virtual int `canDoSimplexInterface () const`
Return the simplex implementation level.

OsiSimplex Group 1

Tableau access methods.

This group of methods provides access to rows and columns of the basis inverse and to rows and columns of the tableau.

- virtual void `enableFactorization () const`
Prepare the solver for the use of tableau access methods.
- virtual void `disableFactorization () const`
Undo the effects of `enableFactorization`.
- virtual bool `basisIsAvailable () const`
Check if an optimal basis is available.
- bool `optimalBasisIsAvailable () const`
Synonym for `basisIsAvailable`.
- virtual void `getBasisStatus (int *cstat, int *rstat) const`
Retrieve status information for column and row variables.
- virtual int `setBasisStatus (const int *cstat, const int *rstat)`
Set the status of column and row variables and update the basis factorization and solution.
- virtual void `getReducedGradient (double *columnReducedCosts, double *duals, const double *c) const`
Calculate duals and reduced costs for the given objective coefficients.
- virtual void `getBlvARow (int row, double *z, double *slack=NULL) const`
Get a row of the tableau.
- virtual void `getBlvRow (int row, double *z) const`
Get a row of the basis inverse.
- virtual void `getBlvACol (int col, double *vec) const`
Get a column of the tableau.
- virtual void `getBlvCol (int col, double *vec) const`
Get a column of the basis inverse.
- virtual void `getBasics (int *index) const`
Get indices of basic variables.

OsiSimplex Group 2

Pivoting methods

This group of methods provides for control of individual pivots by a simplex solver.

- virtual void `enableSimplexInterface (bool doingPrimal)`
Enables normal operation of subsequent functions.
- virtual void `disableSimplexInterface ()`
Undo whatever setting changes the above method had to make.
- virtual int `pivot (int colIn, int colOut, int outStatus)`
Perform a pivot by substituting a colIn for colOut in the basis.
- virtual int `primalPivotResult (int colIn, int sign, int &colOut, int &outStatus, double &t, CoinPackedVector *dx)`

Obtain a result of the primal pivot Outputs: colOut – leaving column, outStatus – its status, t – step size, and, if dx!=NULL, *dx – primal ray direction.

- virtual int [dualPivotResult](#) (int &colIn, int &sign, int colOut, int outStatus, double &t, CoinPackedVector *dx)
Obtain a result of the dual pivot (similar to the previous method) Differences: entering variable and a sign of its change are now the outputs, the leaving variable and its status – the inputs If dx!=NULL, then *dx contains dual ray Return code: same.

Constructors and destructors

- [OsiSolverInterface](#) ()
Default Constructor.
- virtual [OsiSolverInterface](#) * [clone](#) (bool copyData=true) const =0
Clone.
- [OsiSolverInterface](#) (const [OsiSolverInterface](#) &)
Copy constructor.
- [OsiSolverInterface](#) & [operator=](#) (const [OsiSolverInterface](#) &rhs)
Assignment operator.
- virtual [~OsiSolverInterface](#) ()
Destructor.
- virtual void [reset](#) ()
Reset the solver interface.

Protected Member Functions

Protected methods

- virtual void [applyRowCut](#) (const [OsiRowCut](#) &rc)=0
Apply a row cut (append to the constraint matrix).
- virtual void [applyColCut](#) (const [OsiColCut](#) &cc)=0
Apply a column cut (adjust the bounds of one or more variables).
- 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 [convertSenseToBound](#) (const char sense, const double right, const double range, double &lower, double &upper) const
A quick inlined function to convert from the sense/rhs/range style of constraint definition to the lb/ub style.
- 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.
- void [setInitialData](#) ()
Set [OsiSolverInterface](#) object state for default constructor.

Protected Attributes

Protected member data

- [OsiRowCutDebugger](#) * [rowCutDebugger_](#)
Pointer to row cut debugger object.
- [CoinMessageHandler](#) * [handler_](#)
Message handler.
- bool [defaultHandler_](#)
Flag to say if the current handler is the default handler.
- [CoinMessages](#) [messages_](#)
Messages.

- int [numberIntegers_](#)
Number of integers.
- int [numberObjects_](#)
Total number of objects.
- [OsiObject](#) ** [object_](#)
Integer and ... information (integer info normally at beginning)
- char * [columnType_](#)
Column type 0 - continuous 1 - binary (may get fixed later) 2 - general integer (may get fixed later)

Private Attributes

Private member data

- [OsiAuxInfo](#) * [appDataEtc_](#)
Pointer to user-defined data structure - and more if user wants.
- int [iParam_](#) [[OsiLastiParam](#)]
Array of integer parameters.
- double [dblParam_](#) [[OsiLastDbiParam](#)]
Array of double parameters.
- std::string [strParam_](#) [[OsiLastStrParam](#)]
Array of string parameters.
- bool [hintParam_](#) [[OsiLastHintParam](#)]
Array of hint parameters.
- [OsiHintStrength](#) [hintStrength_](#) [[OsiLastHintParam](#)]
Array of hint strengths.
- CoinWarmStart * [ws_](#)
Warm start information used for hot starts when the default hot start implementation is used.
- std::vector< double > [strictColSolution_](#)
Column solution satisfying lower and upper column bounds.
- [OsiNameVec](#) [rowNames_](#)
Row names.
- [OsiNameVec](#) [colNames_](#)
Column names.
- std::string [objName_](#)
Objective name.

Friends

- void [OsiSolverInterfaceCommonUnitTest](#) (const [OsiSolverInterface](#) *emptySi, const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiSolverInterface](#) class.
- void [OsiSolverInterfaceMpsUnitTest](#) (const std::vector< [OsiSolverInterface](#) * > &vecSiP, const std::string &mpsDir)
A function that tests that a lot of problems given in MPS files (mostly the NETLIB problems) solve properly with all the specified solvers.

8.32.1 Detailed Description

Abstract Base Class for describing an interface to a solver.

Many [OsiSolverInterface](#) query methods return a const pointer to the requested read-only data. If the model data is changed or the solver is called, these pointers may no longer be valid and should be refreshed by invoking the member function to obtain an updated copy of the pointer. For example:

```
OsiSolverInterface solverInterfacePtr ;
const double * ruBnds = solverInterfacePtr->getRowUpper();
solverInterfacePtr->applyCuts(someSetOfCuts);
// ruBnds is no longer a valid pointer and must be refreshed
ruBnds = solverInterfacePtr->getRowUpper();
```

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.

Definition at line 62 of file OsiSolverInterface.hpp.

8.32.2 Member Typedef Documentation

8.32.2.1 typedef std::vector<std::string> OsiSolverInterface::OsiNameVec

Data type for name vectors.

Definition at line 888 of file OsiSolverInterface.hpp.

8.32.3 Constructor & Destructor Documentation

8.32.3.1 OsiSolverInterface::OsiSolverInterface ()

Default Constructor.

8.32.3.2 OsiSolverInterface::OsiSolverInterface (const OsiSolverInterface &)

Copy constructor.

8.32.3.3 virtual OsiSolverInterface::~~OsiSolverInterface () [virtual]

Destructor.

8.32.4 Member Function Documentation

8.32.4.1 virtual void OsiSolverInterface::initialSolve () [pure virtual]

Solve initial LP relaxation.

Implemented in [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiGrbSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.2 virtual void OsiSolverInterface::resolve () [pure virtual]

Resolve an LP relaxation after problem modification.

Note the 're-' in 'resolve'. [initialSolve\(\)](#) should be used to solve the problem for the first time.

Implemented in [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiGrbSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.3 virtual void OsiSolverInterface::branchAndBound () [pure virtual]

Invoke solver's built-in enumeration algorithm.

Implemented in [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiGrbSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.4 `virtual bool OsiSolverInterface::setIntParam (OsiIntParam key, int value) [inline],[virtual]`

Set an integer parameter.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiGrbSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

Definition at line 266 of file `OsiSolverInterface.hpp`.

8.32.4.5 `virtual bool OsiSolverInterface::setDbParam (OsiDbParam key, double value) [inline],[virtual]`

Set a double parameter.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiGrbSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

Definition at line 272 of file `OsiSolverInterface.hpp`.

8.32.4.6 `virtual bool OsiSolverInterface::setStrParam (OsiStrParam key, const std::string & value) [inline],[virtual]`

Set a string parameter.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiGrbSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

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

8.32.4.7 `virtual bool OsiSolverInterface::setHintParam (OsiHintParam key, bool yesNo = true, OsiHintStrength strength = OsiHintTry, void * =NULL) [inline],[virtual]`

Set a hint parameter.

The `otherInformation` parameter can be used to pass in an arbitrary block of information which is interpreted by the OSI and the underlying solver. Users are cautioned that this hook is solver-specific.

Implementors: The default implementation completely ignores `otherInformation` and always throws an exception for `OsiForceDo`. This is almost certainly not the behaviour you want; you really should override this method.

Reimplemented in [OsiGlpkSolverInterface](#), and [OsiGrbSolverInterface](#).

Definition at line 294 of file `OsiSolverInterface.hpp`.

8.32.4.8 `virtual bool OsiSolverInterface::getIntParam (OsiIntParam key, int & value) const [inline],[virtual]`

Get an integer parameter.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiGrbSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

Definition at line 307 of file `OsiSolverInterface.hpp`.

8.32.4.9 `virtual bool OsiSolverInterface::getDbParam (OsiDbParam key, double & value) const [inline],[virtual]`

Get a double parameter.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiGrbSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

Definition at line 313 of file `OsiSolverInterface.hpp`.

8.32.4.10 `virtual bool OsiSolverInterface::getStrParam (OsiStrParam key, std::string & value) const [inline], [virtual]`

Get a string parameter.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiGrbSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

Definition at line 319 of file `OsiSolverInterface.hpp`.

8.32.4.11 `virtual bool OsiSolverInterface::getHintParam (OsiHintParam key, bool & yesNo, OsiHintStrength & strength, void *& otherInformation) const [inline], [virtual]`

Get a hint parameter (all information)

Return all available information for the hint: sense, strength, and any extra information associated with the hint.

Implementors: The default implementation will always set `otherInformation` to `NULL`. This is almost certainly not the behaviour you want; you really should override this method.

Reimplemented in [OsiGrbSolverInterface](#).

Definition at line 333 of file `OsiSolverInterface.hpp`.

8.32.4.12 `virtual bool OsiSolverInterface::getHintParam (OsiHintParam key, bool & yesNo, OsiHintStrength & strength) const [inline], [virtual]`

Get a hint parameter (sense and strength only)

Return only the sense and strength of the hint.

Reimplemented in [OsiGrbSolverInterface](#).

Definition at line 347 of file `OsiSolverInterface.hpp`.

8.32.4.13 `virtual bool OsiSolverInterface::getHintParam (OsiHintParam key, bool & yesNo) const [inline], [virtual]`

Get a hint parameter (sense only)

Return only the sense (true/false) of the hint.

Reimplemented in [OsiGrbSolverInterface](#).

Definition at line 359 of file `OsiSolverInterface.hpp`.

8.32.4.14 `void OsiSolverInterface::copyParameters (OsiSolverInterface & rhs)`

Copy all parameters in this section from one solver to another.

Note that the current implementation also copies the `appData` block, message handler, and `rowCutDebugger`. Arguably these should have independent copy methods.

8.32.4.15 `double OsiSolverInterface::getIntegerTolerance () const [inline]`

Return the integrality tolerance of the underlying solver.

We should be able to get an integrality tolerance, but until that time just use the primal tolerance

Todo This method should be replaced; it's architecturally wrong. This should be an honest `dblParam` with a keyword. Underlying solvers that do not support integer variables should return false for set and get on this parameter. Underlying solvers that support integrality should add this to the parameters they support, using whatever tolerance is appropriate. -lh, 091021-

Definition at line 386 of file OsiSolverInterface.hpp.

8.32.4.16 `virtual bool OsiSolverInterface::isAbandoned () const [pure virtual]`

Are there numerical difficulties?

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.17 `virtual bool OsiSolverInterface::isProvenOptimal () const [pure virtual]`

Is optimality proven?

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.18 `virtual bool OsiSolverInterface::isProvenPrimalInfeasible () const [pure virtual]`

Is primal infeasibility proven?

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.19 `virtual bool OsiSolverInterface::isProvenDualInfeasible () const [pure virtual]`

Is dual infeasibility proven?

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiSpxSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.20 `virtual bool OsiSolverInterface::isPrimalObjectiveLimitReached () const [virtual]`

Is the given primal objective limit reached?

Reimplemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.21 `virtual bool OsiSolverInterface::isDualObjectiveLimitReached () const [virtual]`

Is the given dual objective limit reached?

Reimplemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.22 `virtual bool OsiSolverInterface::isIterationLimitReached () const [pure virtual]`

Iteration limit reached?

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.23 `virtual CoinWarmStart* OsiSolverInterface::getEmptyWarmStart () const [pure 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.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.24 `virtual CoinWarmStart* OsiSolverInterface::getWarmStart () const` [pure 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.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.25 `virtual CoinWarmStart* OsiSolverInterface::getPointerToWarmStart (bool & mustDelete)` [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. This does not necessarily create an object - may just point to one. must Delete set true if user should delete returned object.

8.32.4.26 `virtual bool OsiSolverInterface::setWarmStart (const CoinWarmStart * warmstart)` [pure 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 a null parameter should cause the solver interface to refresh its warm start information from the underlying solver.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.27 `virtual void OsiSolverInterface::markHotStart ()` [virtual]

Create a hot start snapshot of the optimization process.

Reimplemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.28 `virtual void OsiSolverInterface::solveFromHotStart ()` [virtual]

Optimize starting from the hot start snapshot.

Reimplemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.29 `virtual void OsiSolverInterface::unmarkHotStart ()` [virtual]

Delete the hot start snapshot.

Reimplemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.30 `virtual int OsiSolverInterface::getNumCols () const` [pure virtual]

Get the number of columns.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.31 `virtual int OsiSolverInterface::getNumRows () const` [pure virtual]

Get the number of rows.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.32 `virtual int OsiSolverInterface::getNumElements () const [pure virtual]`

Get the number of nonzero elements.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.33 `virtual int OsiSolverInterface::getNumIntegers () const [virtual]`

Get the number of integer variables.

8.32.4.34 `virtual const double* OsiSolverInterface::getColLower () const [pure virtual]`

Get a pointer to an array[[getNumCols\(\)](#)] of column lower bounds.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.35 `virtual const double* OsiSolverInterface::getColUpper () const [pure virtual]`

Get a pointer to an array[[getNumCols\(\)](#)] of column upper bounds.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.36 `virtual const char* OsiSolverInterface::getRowSense () const [pure virtual]`

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

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

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.37 `virtual const double* OsiSolverInterface::getRightHandSide () const [pure virtual]`

Get a pointer to an 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

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.38 `virtual const double* OsiSolverInterface::getRowRange () const [pure virtual]`

Get a pointer to an 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

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.39 `virtual const double* OsiSolverInterface::getRowLower () const [pure virtual]`

Get a pointer to an array[getNumRows()] of row lower bounds.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.40 `virtual const double* OsiSolverInterface::getRowUpper () const [pure virtual]`

Get a pointer to an array[getNumRows()] of row upper bounds.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.41 `virtual const double* OsiSolverInterface::getObjCoefficients () const [pure virtual]`

Get a pointer to an array[getNumCols()] of objective function coefficients.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.42 `virtual double OsiSolverInterface::getObjSense () const [pure virtual]`

Get the objective function sense.

- 1 for minimisation (default)
- -1 for maximisation

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.43 `virtual bool OsiSolverInterface::isContinuous (int colIndex) const [pure virtual]`

Return true if the variable is continuous.

Implemented in [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.44 `virtual bool OsiSolverInterface::isBinary (int colIndex) const [virtual]`

Return true if the variable is binary.

8.32.4.45 `virtual bool OsiSolverInterface::isInteger (int colIndex) const [virtual]`

Return true if the variable is integer.

This method returns true if the variable is binary or general integer.

8.32.4.46 `virtual bool OsiSolverInterface::isIntegerNonBinary (int colIndex) const` `[virtual]`

Return true if the variable is general integer.

8.32.4.47 `virtual bool OsiSolverInterface::isFreeBinary (int colIndex) const` `[virtual]`

Return true if the variable is binary and not fixed.

8.32.4.48 `const char* OsiSolverInterface::columnType (bool refresh = false) const` `[inline]`

Return an array[[getNumCols\(\)](#)] of column types.

Deprecated See [getColType](#)

Definition at line 593 of file `OsiSolverInterface.hpp`.

8.32.4.49 `virtual const char* OsiSolverInterface::getColType (bool refresh = false) const` `[virtual]`

Return an array[[getNumCols\(\)](#)] of column types.

- 0 - continuous
- 1 - binary
- 2 - general integer

If `refresh` is true, the classification of integer variables as binary or general integer will be reevaluated. If the current bounds are [0,1], or if the variable is fixed at 0 or 1, it will be classified as binary, otherwise it will be classified as general integer.

8.32.4.50 `virtual const CoinPackedMatrix* OsiSolverInterface::getMatrixByRow () const` `[pure virtual]`

Get a pointer to a row-wise copy of the matrix.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.51 `virtual const CoinPackedMatrix* OsiSolverInterface::getMatrixByCol () const` `[pure virtual]`

Get a pointer to a column-wise copy of the matrix.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.52 `virtual CoinPackedMatrix* OsiSolverInterface::getMutableMatrixByRow () const` `[inline], [virtual]`

Get a pointer to a mutable row-wise copy of the matrix.

Returns NULL if the request is not meaningful (i.e., the OSI will not recognise any modifications to the matrix).

Definition at line 620 of file `OsiSolverInterface.hpp`.

8.32.4.53 `virtual CoinPackedMatrix* OsiSolverInterface::getMutableMatrixByCol () const` `[inline], [virtual]`

Get a pointer to a mutable column-wise copy of the matrix.

Returns NULL if the request is not meaningful (i.e., the OSI will not recognise any modifications to the matrix).

Definition at line 627 of file `OsiSolverInterface.hpp`.

8.32.4.54 `virtual double OsiSolverInterface::getInfinity () const [pure virtual]`

Get the solver's value for infinity.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.55 `virtual const double* OsiSolverInterface::getColSolution () const [pure virtual]`

Get a pointer to an array[[getNumCols\(\)](#)] of primal variable values.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.56 `virtual const double* OsiSolverInterface::getStrictColSolution () [virtual]`

Get a pointer to an array[[getNumCols\(\)](#)] of primal variable values guaranteed to be between the column lower and upper bounds.

8.32.4.57 `virtual const double* OsiSolverInterface::getRowPrice () const [pure virtual]`

Get pointer to array[[getNumRows\(\)](#)] of dual variable values.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.58 `virtual const double* OsiSolverInterface::getReducedCost () const [pure virtual]`

Get a pointer to an array[[getNumCols\(\)](#)] of reduced costs.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.59 `virtual const double* OsiSolverInterface::getRowActivity () const [pure virtual]`

Get a pointer to array[[getNumRows\(\)](#)] of row activity levels.

The row activity for a row is the left-hand side evaluated at the current solution.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.60 `virtual double OsiSolverInterface::getObjValue () const [pure virtual]`

Get the objective function value.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.61 `virtual int OsiSolverInterface::getIterationCount () const [pure virtual]`

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

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.62 `virtual std::vector<double*> OsiSolverInterface::getDualRays (int maxNumRays, bool fullRay = false) const [pure virtual]`

Get as many dual rays as the solver can provide.

In case of proven primal infeasibility there should (with high probability) be at least one.

The first `getNumRows()` ray components will always be associated with the row duals (as returned by `getRowPrice()`). If `fullRay` is true, the final `getNumCols()` entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

Note

Implementors of solver interfaces note that the double pointers in the vector should point to arrays of length `getNumRows()` (`fullRay = false`) or `(getNumRows()+getNumCols())` (`fullRay = true`) and they should be allocated with `new[]`.

Clients of solver interfaces note that it is the client's responsibility to free the double pointers in the vector using `delete[]`. Clients are reminded that a problem can be dual and primal infeasible.

Implemented in `OsiGlpkSolverInterface`, `OsiCpxSolverInterface`, `OsiSpxSolverInterface`, `OsiMskSolverInterface`, `OsiGrbSolverInterface`, and `OsiXprSolverInterface`.

8.32.4.63 `virtual std::vector<double*> OsiSolverInterface::getPrimalRays (int maxNumRays) const` [pure virtual]

Get as many primal rays as the solver can provide.

In case of proven dual infeasibility there should (with high probability) be at least one.

Note

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

Clients of solver interfaces note that it is the client's responsibility to free the double pointers in the vector using `delete[]`. Clients are reminded that a problem can be dual and primal infeasible.

Implemented in `OsiGlpkSolverInterface`, `OsiCpxSolverInterface`, `OsiSpxSolverInterface`, `OsiMskSolverInterface`, `OsiGrbSolverInterface`, and `OsiXprSolverInterface`.

8.32.4.64 `virtual OsiVectorInt OsiSolverInterface::getFractionalIndices (const double etol = 1.e-05) const` [virtual]

Get vector of indices of primal variables which are integer variables but have fractional values in the current solution.

8.32.4.65 `virtual void OsiSolverInterface::setObjCoeff (int elementIndex, double elementValue)` [pure virtual]

Set an objective function coefficient.

Implemented in `OsiGlpkSolverInterface`, `OsiCpxSolverInterface`, `OsiSpxSolverInterface`, `OsiMskSolverInterface`, `OsiGrbSolverInterface`, and `OsiXprSolverInterface`.

8.32.4.66 `virtual void OsiSolverInterface::setObjCoeffSet (const int * indexFirst, const int * indexLast, const double * coeffList)` [virtual]

Set a set of objective function coefficients.

Reimplemented in `OsiCpxSolverInterface`, `OsiMskSolverInterface`, and `OsiGrbSolverInterface`.

8.32.4.67 `virtual void OsiSolverInterface::setObjective (const double * array)` [virtual]

Set the objective coefficients for all columns.

array `[getNumCols()]` is an array of values for the objective. This defaults to a series of set operations and is here for speed.

8.32.4.68 `virtual void OsiSolverInterface::setObjSense (double s) [pure virtual]`

Set the objective function sense.

Use 1 for minimisation (default), -1 for maximisation.

Note

Implementors note that objective function sense is a parameter of the OSI, not a property of the problem. Objective sense can be set prior to problem load and should not be affected by loading a new problem.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.69 `virtual void OsiSolverInterface::setColLower (int elementIndex, double elementValue) [pure virtual]`

Set a single column lower bound.

Use `-getInfinity()` for -infinity.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiSpxSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.70 `virtual void OsiSolverInterface::setColLower (const double * array) [virtual]`

Set the lower bounds for all columns.

array [\[getNumCols\(\)\]](#) is an array of values for the lower bounds. This defaults to a series of set operations and is here for speed.

8.32.4.71 `virtual void OsiSolverInterface::setColUpper (int elementIndex, double elementValue) [pure virtual]`

Set a single column upper bound.

Use `getInfinity()` for infinity.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiSpxSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.72 `virtual void OsiSolverInterface::setColUpper (const double * array) [virtual]`

Set the upper bounds for all columns.

array [\[getNumCols\(\)\]](#) is an array of values for the upper bounds. This defaults to a series of set operations and is here for speed.

8.32.4.73 `virtual void OsiSolverInterface::setColBounds (int elementIndex, double lower, double upper) [inline],
[virtual]`

Set a single column lower and upper bound.

The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#)

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiSpxSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

Definition at line 778 of file `OsiSolverInterface.hpp`.

8.32.4.74 `virtual void OsiSolverInterface::setColSetBounds (const int * indexFirst, const int * indexLast, const double * boundList) [virtual]`

Set the upper and lower bounds of a set of columns.

The default implementation just invokes [setColBounds\(\)](#) over and over again. For each column, boundList must contain both a lower and upper bound, in that order.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.75 `virtual void OsiSolverInterface::setRowLower (int elementIndex, double elementValue) [pure virtual]`

Set a single row lower bound.

Use `-getInfinity()` for -infinity.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiSpxSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.76 `virtual void OsiSolverInterface::setRowUpper (int elementIndex, double elementValue) [pure virtual]`

Set a single row upper bound.

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

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiSpxSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.77 `virtual void OsiSolverInterface::setRowBounds (int elementIndex, double lower, double upper) [inline],
[virtual]`

Set a single row lower and upper bound.

The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#)

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiSpxSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

Definition at line 805 of file `OsiSolverInterface.hpp`.

8.32.4.78 `virtual void OsiSolverInterface::setRowSetBounds (const int * indexFirst, const int * indexLast, const double *
boundList) [virtual]`

Set the bounds on a set of rows.

The default implementation just invokes [setRowBounds\(\)](#) over and over again. For each row, boundList must contain both a lower and upper bound, in that order.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.79 `virtual void OsiSolverInterface::setRowType (int index, char sense, double rightHandSide, double range) [pure
virtual]`

Set the type of a single row.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiSpxSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.80 `virtual void OsiSolverInterface::setRowSetTypes (const int * indexFirst, const int * indexLast, const char * senseList,
const double * rhsList, const double * rangeList) [virtual]`

Set the type of a set of rows.

The default implementation just invokes [setRowType\(\)](#) over and over again.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.81 `virtual void OsiSolverInterface::setColSolution (const double * colsol) [pure virtual]`

Set the primal solution variable values.

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

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.82 `virtual void OsiSolverInterface::setRowPrice (const double * rowprice) [pure 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.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.83 `virtual int OsiSolverInterface::reducedCostFix (double gap, bool justInteger = true) [virtual]`

Fix variables at bound based on reduced cost.

For variables currently at bound, fix the variable at bound if the reduced cost exceeds the gap. Return the number of variables fixed.

If `justInteger` is set to false, the routine will also fix continuous variables, but the test still assumes a delta of 1.0.

8.32.4.84 `virtual void OsiSolverInterface::setContinuous (int index) [pure virtual]`

Set the index-th variable to be a continuous variable.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.85 `virtual void OsiSolverInterface::setInteger (int index) [pure virtual]`

Set the index-th variable to be an integer variable.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.86 `virtual void OsiSolverInterface::setContinuous (const int * indices, int len) [virtual]`

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

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.87 `virtual void OsiSolverInterface::setInteger (const int * indices, int len) [virtual]`

Set the variables listed in `indices` (which is of length `len`) to be integer variables.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.88 `virtual std::string OsiSolverInterface::dfltRowColName (char rc, int ndx, unsigned digits = 7) const [virtual]`

Generate a standard name of the form Rnnnnnnn or Cnnnnnnn.

Set `rc` to 'r' for a row name, 'c' for a column name. The 'nnnnnnn' part is generated from `ndx` and will contain 7 digits by default, padded with zeros if necessary. As a special case, `ndx = getNumRows()` is interpreted as a request for the name of the objective function. OBJECTIVE is returned, truncated to `digits+1` characters to match the row and column names.

8.32.4.89 `virtual std::string OsiSolverInterface::getObjName (unsigned maxlen = static_cast< unsigned >(std::string::npos)) const [virtual]`

Return the name of the objective function.

8.32.4.90 `virtual void OsiSolverInterface::setObjName (std::string name) [inline],[virtual]`

Set the name of the objective function.

Reimplemented in [OsiGlpkSolverInterface](#).

Definition at line 929 of file `OsiSolverInterface.hpp`.

8.32.4.91 `virtual std::string OsiSolverInterface::getRowName (int rowIndex, unsigned maxlen = static_cast< unsigned >(std::string::npos)) const [virtual]`

Return the name of the row.

The routine will *always* return some name, regardless of the name discipline or the level of support by an OsiXXX derived class. Use `maxLen` to limit the length.

8.32.4.92 `virtual const OsiNameVec& OsiSolverInterface::getRowNames () [virtual]`

Return a pointer to a vector of row names.

If the name discipline ([OsiNameDiscipline](#)) is auto, the return value will be a vector of length zero. If the name discipline is lazy, the vector will contain only names supplied by the client and will be no larger than needed to hold those names; entries not supplied will be null strings. In particular, the objective name is *not* included in the vector for lazy names. If the name discipline is full, the vector will have `getNumRows()` names, either supplied or generated, plus one additional entry for the objective name.

8.32.4.93 `virtual void OsiSolverInterface::setRowName (int ndx, std::string name) [virtual]`

Set a row name.

Quietly does nothing if the name discipline ([OsiNameDiscipline](#)) is auto. Quietly fails if the row index is invalid.

Reimplemented in [OsiGlpkSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.94 `virtual void OsiSolverInterface::setRowNames (OsiNameVec & srcNames, int srcStart, int len, int tgtStart) [virtual]`

Set multiple row names.

The run of `len` entries starting at `srcNames[srcStart]` are installed as row names starting at row index `tgtStart`. The base class implementation makes repeated calls to `setRowName`.

8.32.4.95 `virtual void OsiSolverInterface::deleteRowNames (int tgtStart, int len)` [virtual]

Delete len row names starting at index tgtStart.

The specified row names are removed and the remaining row names are copied down to close the gap.

8.32.4.96 `virtual std::string OsiSolverInterface::getColName (int colIndex, unsigned maxLen = static_cast< unsigned >(std::string::npos)) const` [virtual]

Return the name of the column.

The routine will *always* return some name, regardless of the name discipline or the level of support by an OsiXXX derived class. Use maxLen to limit the length.

8.32.4.97 `virtual const OsiNameVec& OsiSolverInterface::getColNames ()` [virtual]

Return a pointer to a vector of column names.

If the name discipline ([OsiNameDiscipline](#)) is auto, the return value will be a vector of length zero. If the name discipline is lazy, the vector will contain only names supplied by the client and will be no larger than needed to hold those names; entries not supplied will be null strings. If the name discipline is full, the vector will have [getNumCols\(\)](#) names, either supplied or generated.

8.32.4.98 `virtual void OsiSolverInterface::setColName (int ndx, std::string name)` [virtual]

Set a column name.

Quietly does nothing if the name discipline ([OsiNameDiscipline](#)) is auto. Quietly fails if the column index is invalid.

Reimplemented in [OsiGlpkSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.99 `virtual void OsiSolverInterface::setColNames (OsiNameVec & srcNames, int srcStart, int len, int tgtStart)` [virtual]

Set multiple column names.

The run of len entries starting at srcNames[srcStart] are installed as column names starting at column index tgtStart. The base class implementation makes repeated calls to setColName.

8.32.4.100 `virtual void OsiSolverInterface::deleteColNames (int tgtStart, int len)` [virtual]

Delete len column names starting at index tgtStart.

The specified column names are removed and the remaining column names are copied down to close the gap.

8.32.4.101 `void OsiSolverInterface::setRowColNames (const CoinMpsIO & mps)`

Set row and column names from a CoinMpsIO object.

Also sets the name of the objective function. If the name discipline is auto, you get what you asked for. This routine does not use setRowName or setColName.

8.32.4.102 `void OsiSolverInterface::setRowColNames (CoinModel & mod)`

Set row and column names from a CoinModel object.

If the name discipline is auto, you get what you asked for. This routine does not use setRowName or setColName.

8.32.4.103 `void OsiSolverInterface::setRowColNames (CoinLpIO & mod)`

Set row and column names from a CoinLpIO object.

Also sets the name of the objective function. If the name discipline is auto, you get what you asked for. This routine does not use `setRowName` or `setColName`.

8.32.4.104 `virtual void OsiSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [pure virtual]`

Add a column (primal variable) to the problem.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.105 `virtual void OsiSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj, std::string name) [virtual]`

Add a named column (primal variable) to the problem.

The default implementation adds the column, then changes the name. This can surely be made more efficient within an `OsiXXX` class.

8.32.4.106 `virtual void OsiSolverInterface::addCol (int numberElements, const int * rows, const double * elements, const double collb, const double colub, const double obj) [virtual]`

Add a column (primal variable) to the problem.

8.32.4.107 `virtual void OsiSolverInterface::addCol (int numberElements, const int * rows, const double * elements, const double collb, const double colub, const double obj, std::string name) [virtual]`

Add a named column (primal variable) to the problem.

The default implementation adds the column, then changes the name. This can surely be made more efficient within an `OsiXXX` class.

8.32.4.108 `virtual void OsiSolverInterface::addCols (const int numcols, const CoinPackedVectorBase *const * cols, const double * collb, const double * colub, const double * obj) [virtual]`

Add a set of columns (primal variables) to the problem.

The default implementation simply makes repeated calls to [addCol\(\)](#).

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.109 `virtual void OsiSolverInterface::addCols (const int numcols, const int * columnStarts, const int * rows, const double * elements, const double * collb, const double * colub, const double * obj) [virtual]`

Add a set of columns (primal variables) to the problem.

The default implementation simply makes repeated calls to [addCol\(\)](#).

8.32.4.110 `void OsiSolverInterface::addCols (const CoinBuild & buildObject)`

Add columns using a `CoinBuild` object.

8.32.4.111 `int OsiSolverInterface::addCols (CoinModel & modelObject)`

Add columns from a model object.

returns -1 if object in bad state (i.e. has row information) otherwise number of errors `modelObject` non const as can be regularized as part of build

8.32.4.112 `virtual void OsiSolverInterface::deleteCols (const int num, const int * colIndices) [pure 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.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.113 `virtual void OsiSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [pure virtual]`

Add a row (constraint) to the problem.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.114 `virtual void OsiSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub, std::string name) [virtual]`

Add a named row (constraint) to the problem.

The default implementation adds the row, then changes the name. This can surely be made more efficient within an OsiXXX class.

8.32.4.115 `virtual void OsiSolverInterface::addRow (const CoinPackedVectorBase & vec, const char rowSEN, const double rowRHS, const double rowrng) [pure virtual]`

Add a row (constraint) to the problem.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.116 `virtual void OsiSolverInterface::addRow (const CoinPackedVectorBase & vec, const char rowSEN, const double rowRHS, const double rowrng, std::string name) [virtual]`

Add a named row (constraint) to the problem.

The default implementation adds the row, then changes the name. This can surely be made more efficient within an OsiXXX class.

8.32.4.117 `virtual void OsiSolverInterface::addRow (int numberElements, const int * columns, const double * element, const double rowlb, const double rowub) [virtual]`

Add a row (constraint) to the problem.

Converts to `addRow(CoinPackedVectorBase&,const double,const double)`.

8.32.4.118 `virtual void OsiSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const double * rowlb, const double * rowub) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.119 `virtual void OsiSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const char * rowSEN, const double * rowRHS, const double * rowRNG) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.120 `virtual void OsiSolverInterface::addRows (const int numrows, const int * rowStarts, const int * columns, const double * element, const double * rowLB, const double * rowUB) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

8.32.4.121 `void OsiSolverInterface::addRows (const CoinBuild & buildObject)`

Add rows using a CoinBuild object.

8.32.4.122 `int OsiSolverInterface::addRows (CoinModel & modelObject)`

Add rows from a CoinModel object.

Returns -1 if the object is in the wrong state (*i.e.*, has column-major information), otherwise the number of errors.

The modelObject is not const as it can be regularized as part of the build.

8.32.4.123 `virtual void OsiSolverInterface::deleteRows (const int num, const int * rowIndices) [pure 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.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.124 `virtual void OsiSolverInterface::replaceMatrixOptional (const CoinPackedMatrix &) [inline],[virtual]`

Replace the constraint matrix.

I (JJF) am getting annoyed because I can't just replace a matrix. The default behavior of this is do nothing so only use where that would not matter, e.g. strengthening a matrix for MIP.

Definition at line 1228 of file OsiSolverInterface.hpp.

8.32.4.125 `virtual void OsiSolverInterface::replaceMatrix (const CoinPackedMatrix &) [inline],[virtual]`

Replace the constraint matrix.

And if it does matter (not used at present)

Definition at line 1234 of file OsiSolverInterface.hpp.

8.32.4.126 `virtual void OsiSolverInterface::saveBaseModel () [inline],[virtual]`

Save a copy of the base model.

If solver wants it can save a copy of "base" (continuous) model here.

Definition at line 1240 of file OsiSolverInterface.hpp.

8.32.4.127 `virtual void OsiSolverInterface::restoreBaseModel (int numberOfRows) [virtual]`

Reduce the constraint system to the specified number of constraints.

If solver wants it can restore a copy of "base" (continuous) model here.

Note

The name is somewhat misleading. Implementors should consider the opportunity to optimise behaviour in the common case where `numberOfRows` is exactly the number of original constraints. Do not, however, neglect the possibility that `numberOfRows` does not equal the number of original constraints.

8.32.4.128 `virtual ApplyCutsReturnCode OsiSolverInterface::applyCuts (const OsiCuts & cs, double effectivenessLb = 0.0) [virtual]`

Apply a collection of cuts.

Only cuts which have an `effectiveness >= effectivenessLb` are applied.

- `ReturnCode.getNumineffective()` – number of cuts which were not applied because they had an `effectiveness < effectivenessLb`
- `ReturnCode.getNuminconsistent()` – number of invalid cuts
- `ReturnCode.getNuminconsistentWrtIntegerModel()` – number of cuts that are invalid with respect to this integer model
- `ReturnCode.getNuminfeasible()` – number of cuts that would make this integer model infeasible
- `ReturnCode.getNumApplied()` – number of integer cuts which were applied to the integer model
- `cs.size() == getNumineffective() + getNuminconsistent() + getNuminconsistentWrtIntegerModel() + getNuminfeasible() + getNumApplied()`

Reimplemented in [OsiGrbSolverInterface](#).

8.32.4.129 `virtual void OsiSolverInterface::applyRowCuts (int numberOfCuts, const OsiRowCut * cuts) [virtual]`

Apply a collection of row cuts which are all effective.

`applyCuts` seems to do one at a time which seems inefficient. Would be even more efficient to pass an array of pointers.

8.32.4.130 `virtual void OsiSolverInterface::applyRowCuts (int numberOfCuts, const OsiRowCut ** cuts) [virtual]`

Apply a collection of row cuts which are all effective.

This is passed in as an array of pointers.

8.32.4.131 `void OsiSolverInterface::deleteBranchingInfo (int numberOfDeleted, const int * which)`

Deletes branching information before columns deleted.

8.32.4.132 `virtual void OsiSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub) [pure virtual]`

Load in a problem by copying the arguments.

The constraints on the rows are given by lower and upper bounds.

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

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

Note that the default values for `rowub` and `rowlb` produce the constraint $-inf \leq ax \leq inf$. This is probably not what you want.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiXprSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.133 `virtual void OsiSolverInterface::assignProblem (CoinPackedMatrix * & matrix, double * & collb, double * & colub, double * & obj, double * & rowlb, double * & rowub) [pure 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 argument values see the matching `loadProblem` method.

Warning

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

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiXprSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.134 `virtual void OsiSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * collb, const double * colub, const double * obj, const char * rowsen, const double * rowrhs, const double * rowrng) [pure 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

Note that the default values for `rowsen`, `rowrhs`, and `rowrng` produce the constraint $ax \geq 0$.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiXprSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.135 `virtual void OsiSolverInterface::assignProblem (CoinPackedMatrix * & matrix, double * & collb, double * & colub, double * & obj, char * & rowsen, double * & rowrhs, double * & rowrng) [pure 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 argument values see the matching loadProblem method.

Warning

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

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpXSolverInterface](#), [OsiMskSolverInterface](#), [OsiXprSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.136 `virtual void OsiSolverInterface::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) [pure virtual]`

Load in a problem by copying the arguments.

The constraint matrix is is specified with standard column-major column starts / row indices / coefficients vectors. The constraints on the rows are given by lower and upper bounds.

The matrix vectors must be gap-free. Note that `start` must have `numcols+1` entries so that the length of the last column can be calculated as `start[numcols]-start[numcols-1]`.

See the previous loadProblem method using rowlb and rowub for default argument values.

8.32.4.137 `virtual void OsiSolverInterface::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) [pure virtual]`

Load in a problem by copying the arguments.

The constraint matrix is is specified with standard column-major column starts / row indices / coefficients vectors. The constraints on the rows are given by sense/rhs/range triplets.

The matrix vectors must be gap-free. Note that `start` must have `numcols+1` entries so that the length of the last column can be calculated as `start[numcols]-start[numcols-1]`.

See the previous loadProblem method using sense/rhs/range for default argument values.

8.32.4.138 `virtual int OsiSolverInterface::loadFromCoinModel (CoinModel & modelObject, bool keepSolution = false) [virtual]`

Load a model from a CoinModel object.

Return the number of errors encountered.

The modelObject parameter cannot be const as it may be changed as part of process. If keepSolution is true will try and keep warmStart.

8.32.4.139 `virtual int OsiSolverInterface::readMps (const char * filename, const char * extension = "mps") [virtual]`

Read a problem in MPS format from the given filename.

The default implementation uses `CoinMpsIO::readMps()` to read the MPS file and returns the number of errors encountered.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiXprSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.140 `virtual int OsiSolverInterface::readMps (const char * filename, const char * extension, int & numberSets, CoinSet **& sets) [virtual]`

Read a problem in MPS format from the given full filename.

This uses `CoinMpsIO::readMps()` to read the MPS file and returns the number of errors encountered. It also may return an array of set information

8.32.4.141 `virtual int OsiSolverInterface::readGMPL (const char * filename, const char * dataname=NULL) [virtual]`

Read a problem in GMPL format from the given filenames.

The default implementation uses `CoinMpsIO::readGMPL()`. This capability is available only if the third-party package Glpk is installed.

8.32.4.142 `virtual void OsiSolverInterface::writeMps (const char * filename, const char * extension = "mps", double objSense = 0.0) const [pure 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 the implementation.

Implemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), [OsiSpxSolverInterface](#), [OsiMskSolverInterface](#), [OsiXprSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.143 `int OsiSolverInterface::writeMpsNative (const char * filename, const char ** rowNames, const char ** columnNames, int formatType = 0, int numberAcross = 2, double objSense = 0.0, int numberSOS = 0, const CoinSet * setInfo = NULL) const`

Write the problem in MPS format to the specified file with more control over the output.

Row and column names may be null. `formatType` is

- 0 - normal
- 1 - extra accuracy
- 2 - IEEE hex

Returns non-zero on I/O error

8.32.4.144 `virtual void OsiSolverInterface::writeLp (const char * filename, const char * extension = "lp", double epsilon = 1e-5, int numberAcross = 10, int decimals = 5, double objSense = 0.0, bool useRowNames = true) const [virtual]`

Write the problem into an Lp file of the given filename with the specified extension.

Coefficients with value less than epsilon away from an integer value are written as integers. Write at most `numberAcross` monomials on a line. Write non integer numbers with decimals digits after the decimal point.

The written problem is always a minimization problem. If the current problem is a maximization problem, the intended objective function for the written problem is the current objective function multiplied by -1. If the current problem is a minimization problem, the intended objective function for the written problem is the current objective function. If `objSense` < 0, the intended objective function is multiplied by -1 before writing the problem. It is left unchanged otherwise.

Write objective function name and constraint names if `useRowNames` is true. This version calls [writeLpNative\(\)](#).

```
8.32.4.145 virtual void OsiSolverInterface::writeLp ( FILE * fp, double epsilon = 1e-5, int numberAcross = 10, int decimals = 5,
double objSense = 0.0, bool useRowNames = true ) const [virtual]
```

Write the problem into the file pointed to by the parameter *fp*.

Other parameters are similar to those of [writeLp\(\)](#) with first parameter filename.

```
8.32.4.146 int OsiSolverInterface::writeLpNative ( const char * filename, char const *const *const rowNames, char const *const
*const columnNames, const double epsilon = 1.0e-5, const int numberAcross = 10, const int decimals = 5, const
double objSense = 0.0, const bool useRowNames = true ) const
```

Write the problem into an Lp file.

Parameters are similar to those of [writeLp\(\)](#), but in addition row names and column names may be given.

Parameter *rowNames* may be NULL, in which case default row names are used. If *rowNames* is not NULL, it must have exactly one entry per row in the problem and one additional entry (*rowNames*[[getNumRows\(\)](#)]) with the objective function name. These [getNumRows\(\)](#)+1 entries must be distinct. If this is not the case, default row names are used. In addition, format restrictions are imposed on names (see [CoinLpIO::is_invalid_name\(\)](#) for details).

Similar remarks can be made for the parameter *columnNames* which must either be NULL or have exactly [getNumCols\(\)](#) distinct entries.

Write objective function name and constraint names if *useRowNames* is true.

```
8.32.4.147 int OsiSolverInterface::writeLpNative ( FILE * fp, char const *const *const rowNames, char const *const *const
columnNames, const double epsilon = 1.0e-5, const int numberAcross = 10, const int decimals = 5, const double
objSense = 0.0, const bool useRowNames = true ) const
```

Write the problem into the file pointed to by the parameter *fp*.

Other parameters are similar to those of [writeLpNative\(\)](#) with first parameter filename.

```
8.32.4.148 virtual int OsiSolverInterface::readLp ( const char * filename, const double epsilon = 1e-5 ) [virtual]
```

Read file in LP format from file with name *filename*.

See class [CoinLpIO](#) for description of this format.

```
8.32.4.149 int OsiSolverInterface::readLp ( FILE * fp, const double epsilon = 1e-5 )
```

Read file in LP format from the file pointed to by *fp*.

See class [CoinLpIO](#) for description of this format.

```
8.32.4.150 void OsiSolverInterface::setApplicationData ( void * appData )
```

Set application data.

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

```
8.32.4.151 void OsiSolverInterface::setAuxiliaryInfo ( OsiAuxInfo * auxiliaryInfo )
```

Create a clone of an Auxiliary Information object.

The base class just stores an application data pointer but can be more general. Application data pointer is designed for one user while this can be extended to cope with more general extensions.

8.32.4.152 `void* OsiSolverInterface::getApplicationData () const`

Get application data.

8.32.4.153 `OsiAuxInfo* OsiSolverInterface::getAuxiliaryInfo () const`

Get pointer to auxiliary info object.

8.32.4.154 `virtual void OsiSolverInterface::passInMessageHandler (CoinMessageHandler * handler) [virtual]`

Pass in a message handler.

It is the client's responsibility to destroy a message handler installed by this routine; it will not be destroyed when the solver interface is destroyed.

Reimplemented in [OsiCpxSolverInterface](#), [OsiMskSolverInterface](#), and [OsiXprSolverInterface](#).

8.32.4.155 `void OsiSolverInterface::newLanguage (CoinMessages::Language language)`

Set language.

8.32.4.156 `void OsiSolverInterface::setLanguage (CoinMessages::Language language) [inline]`

Definition at line 1622 of file `OsiSolverInterface.hpp`.

8.32.4.157 `CoinMessageHandler* OsiSolverInterface::messageHandler () const [inline]`

Return a pointer to the current message handler.

Definition at line 1625 of file `OsiSolverInterface.hpp`.

8.32.4.158 `CoinMessages OsiSolverInterface::messages () [inline]`

Return the current set of messages.

Definition at line 1628 of file `OsiSolverInterface.hpp`.

8.32.4.159 `CoinMessages* OsiSolverInterface::messagesPointer () [inline]`

Return a pointer to the current set of messages.

Definition at line 1631 of file `OsiSolverInterface.hpp`.

8.32.4.160 `bool OsiSolverInterface::defaultHandler () const [inline]`

Return true if default handler.

Definition at line 1634 of file `OsiSolverInterface.hpp`.

8.32.4.161 `void OsiSolverInterface::findIntegers (bool justCount)`

Identify integer variables and create corresponding objects.

Record integer variables and create an [OsiSimpleInteger](#) object for each one. All existing [OsiSimpleInteger](#) objects will be destroyed. If `justCount` then no objects created and we just store `numberIntegers_`

8.32.4.162 `virtual int OsiSolverInterface::findIntegersAndSOS (bool justCount) [virtual]`

Identify integer variables and SOS and create corresponding objects.

Record integer variables and create an [OsiSimpleInteger](#) object for each one. All existing [OsiSimpleInteger](#) objects will

be destroyed. If the solver supports SOS then do the same for SOS.

If justCount then no objects created and we just store numberIntegers_ Returns number of SOS

8.32.4.163 `int OsiSolverInterface::numberObjects () const [inline]`

Get the number of objects.

Definition at line 1665 of file OsiSolverInterface.hpp.

8.32.4.164 `void OsiSolverInterface::setNumberObjects (int number) [inline]`

Set the number of objects.

Definition at line 1667 of file OsiSolverInterface.hpp.

8.32.4.165 `OsiObject** OsiSolverInterface::objects () const [inline]`

Get the array of objects.

Definition at line 1671 of file OsiSolverInterface.hpp.

8.32.4.166 `const OsiObject* OsiSolverInterface::object (int which) const [inline]`

Get the specified object.

Definition at line 1674 of file OsiSolverInterface.hpp.

8.32.4.167 `OsiObject* OsiSolverInterface::modifiableObject (int which) const [inline]`

Get the specified object.

Definition at line 1676 of file OsiSolverInterface.hpp.

8.32.4.168 `void OsiSolverInterface::deleteObjects ()`

Delete all object information.

8.32.4.169 `void OsiSolverInterface::addObjects (int numberOfObjects, OsiObject** objects)`

Add in object information.

Objects are cloned; the owner can delete the originals.

8.32.4.170 `double OsiSolverInterface::forceFeasible ()`

Use current solution to set bounds so current integer feasible solution will stay feasible.

Only feasible bounds will be used, even if current solution outside bounds. The amount of such violation will be returned (and if small can be ignored)

8.32.4.171 `virtual void OsiSolverInterface::activateRowCutDebugger (const char * modelName) [virtual]`

Activate the row cut debugger.

If *modelName* is in the set of known models then all cuts are checked to see that they do NOT cut off the optimal solution known to the debugger.

8.32.4.172 `virtual void OsiSolverInterface::activateRowCutDebugger (const double * solution, bool enforceOptimality = true)`
`[virtual]`

Activate the row cut debugger using a full solution array.

Activate the debugger for a model not included in the debugger's internal database. Cuts will be checked to see that they do NOT cut off the given solution.

`solution` must be a full solution vector, but only the integer variables need to be correct. The debugger will fill in the continuous variables by solving an lp relaxation with the integer variables fixed as specified. If the given values for the continuous variables should be preserved, set `keepContinuous` to true.

8.32.4.173 `const OsiRowCutDebugger* OsiSolverInterface::getRowCutDebugger () const`

Get the row cut debugger provided the solution known to the debugger is within the feasible region held in the solver.

If there is a row cut debugger object associated with model AND if the solution known to the debugger is within the solver's current feasible region (i.e., the column bounds held in the solver are compatible with the known solution) then a pointer to the debugger is returned which may be used to test validity of cuts.

Otherwise NULL is returned

8.32.4.174 `OsiRowCutDebugger* OsiSolverInterface::getRowCutDebuggerAlways () const`

Get the row cut debugger object.

Return the row cut debugger object if it exists. One common usage of this method is to obtain a debugger object in order to execute [OsiRowCutDebugger::redoSolution](#) (so that the stored solution is again compatible with the problem held in the solver).

8.32.4.175 `virtual int OsiSolverInterface::canDoSimplexInterface () const` `[virtual]`

Return the simplex implementation level.

The return codes are:

- 0: the simplex interface is not implemented.
- 1: the Group 1 (tableau access) methods are implemented.
- 2: the Group 2 (pivoting) methods are implemented

The codes are cumulative - a solver which implements Group 2 also implements Group 1.

Reimplemented in [OsiCpxSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.176 `virtual void OsiSolverInterface::enableFactorization () const` `[virtual]`

Prepare the solver for the use of tableau access methods.

Prepares the solver for the use of the tableau access methods, if any such preparation is required.

The `const` attribute is required due to the places this method may be called (e.g., within `CglCutGenerator::generateCuts()`).

Reimplemented in [OsiCpxSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.177 `virtual void OsiSolverInterface::disableFactorization () const` `[virtual]`

Undo the effects of [enableFactorization](#).

Reimplemented in [OsiCpxSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.178 `virtual bool OsiSolverInterface::basisIsAvailable () const [virtual]`

Check if an optimal basis is available.

Returns true if the problem has been solved to optimality and a basis is available. This should be used to see if the tableau access operations are possible and meaningful.

Note

Implementors please note that this method may be called before [enableFactorization](#).

Reimplemented in [OsiCpxSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.179 `bool OsiSolverInterface::optimalBasisIsAvailable () const [inline]`

Synonym for [basisIsAvailable](#).

Definition at line 1802 of file `OsiSolverInterface.hpp`.

8.32.4.180 `virtual void OsiSolverInterface::getBasisStatus (int * cstat, int * rstat) const [virtual]`

Retrieve status information for column and row variables.

This method returns status as integer codes:

- 0: free
- 1: basic
- 2: nonbasic at upper bound
- 3: nonbasic at lower bound

The [getWarmStart](#) method provides essentially the same functionality for a simplex-oriented solver, but the implementation details are very different.

Note

Logical variables associated with rows are all assumed to have +1 coefficients, so for a \leq constraint the logical will be at lower bound if the constraint is tight.

Implementors may choose to implement this method as a wrapper which converts a `CoinWarmStartBasis` to the requested representation.

Reimplemented in [OsiCpxSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.181 `virtual int OsiSolverInterface::setBasisStatus (const int * cstat, const int * rstat) [virtual]`

Set the status of column and row variables and update the basis factorization and solution.

Status information should be coded as documented for [getBasisStatus](#). Returns 0 if all goes well, 1 if something goes wrong.

This method differs from [setWarmStart](#) in the format of the input and in its immediate effect. Think of it as [setWarmStart](#) immediately followed by [resolve](#), but no pivots are allowed.

Note

Implementors may choose to implement this method as a wrapper that calls [setWarmStart](#) and [resolve](#) if the no pivot requirement can be satisfied.

8.32.4.182 `virtual void OsiSolverInterface::getReducedGradient (double * columnReducedCosts, double * duals, const double * c) const` [virtual]

Calculate duals and reduced costs for the given objective coefficients.

The solver's objective coefficient vector is not changed.

8.32.4.183 `virtual void OsiSolverInterface::getBlvARow (int row, double * z, double * slack = NULL) const` [virtual]

Get a row of the tableau.

If *slack* is not null, it will be loaded with the coefficients for the artificial (logical) variables (i.e., the row of the basis inverse).

Reimplemented in [OsiCpxSolverInterface](#).

8.32.4.184 `virtual void OsiSolverInterface::getBlvRow (int row, double * z) const` [virtual]

Get a row of the basis inverse.

Reimplemented in [OsiCpxSolverInterface](#).

8.32.4.185 `virtual void OsiSolverInterface::getBlvACol (int col, double * vec) const` [virtual]

Get a column of the tableau.

Reimplemented in [OsiCpxSolverInterface](#).

8.32.4.186 `virtual void OsiSolverInterface::getBlvCol (int col, double * vec) const` [virtual]

Get a column of the basis inverse.

Reimplemented in [OsiCpxSolverInterface](#).

8.32.4.187 `virtual void OsiSolverInterface::getBasics (int * index) const` [virtual]

Get indices of basic variables.

If the logical (artificial) for row *i* is basic, the index should be coded as ([getNumCols](#) + *i*). The order of indices must match the order of elements in the vectors returned by [getBlvACol](#) and [getBlvCol](#).

Reimplemented in [OsiCpxSolverInterface](#).

8.32.4.188 `virtual void OsiSolverInterface::enableSimplexInterface (bool doingPrimal)` [virtual]

Enables normal operation of subsequent functions.

This method is supposed to ensure that all typical things (like reduced costs, etc.) are updated when individual pivots are executed and can be queried by other methods. says whether will be doing primal or dual

8.32.4.189 `virtual void OsiSolverInterface::disableSimplexInterface ()` [virtual]

Undo whatever setting changes the above method had to make.

Reimplemented in [OsiCpxSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.190 `virtual int OsiSolverInterface::pivot (int colIn, int colOut, int outStatus)` [virtual]

Perform a pivot by substituting a *colIn* for *colOut* in the basis.

The status of the leaving variable is given in *outStatus*. Where 1 is to upper bound, -1 to lower bound Return code was undefined - now for OsiClp is 0 for okay, 1 if inaccuracy forced re-factorization (should be okay) and -1 for singular

factorization

8.32.4.191 `virtual int OsiSolverInterface::primalPivotResult (int colIn, int sign, int & colOut, int & outStatus, double & t, CoinPackedVector * dx) [virtual]`

Obtain a result of the primal pivot Outputs: *colOut* – leaving column, *outStatus* – its status, *t* – step size, and, if *dx*!=NULL, **dx* – primal ray direction.

Inputs: *colIn* – entering column, *sign* – direction of its change (+/-1). Both for *colIn* and *colOut*, artificial variables are index by the negative of the row index minus 1. Return code (for now): 0 – leaving variable found, -1 – everything else? Clearly, more informative set of return values is required Primal and dual solutions are updated

8.32.4.192 `virtual int OsiSolverInterface::dualPivotResult (int & colIn, int & sign, int colOut, int outStatus, double & t, CoinPackedVector * dx) [virtual]`

Obtain a result of the dual pivot (similar to the previous method) Differences: entering variable and a sign of its change are now the outputs, the leaving variable and its status – the inputs If *dx*!=NULL, then **dx* contains dual ray Return code: same.

8.32.4.193 `virtual OsiSolverInterface* OsiSolverInterface::clone (bool copyData =true) const [pure virtual]`

Clone.

The result of calling `clone(false)` is defined to be equivalent to calling the default constructor `OsiSolverInterface()`.

Implemented in [OsiGlpkSolverInterface](#), [OsiMskSolverInterface](#), [OsiCpxSolverInterface](#), [OsiXprSolverInterface](#), [OsiGrbSolverInterface](#), and [OsiSpxSolverInterface](#).

8.32.4.194 `OsiSolverInterface& OsiSolverInterface::operator= (const OsiSolverInterface & rhs)`

Assignment operator.

8.32.4.195 `virtual void OsiSolverInterface::reset () [virtual]`

Reset the solver interface.

A call to `reset()` returns the solver interface to the same state as it would have if it had just been constructed by calling the default constructor `OsiSolverInterface()`.

Reimplemented in [OsiGlpkSolverInterface](#), [OsiCpxSolverInterface](#), and [OsiGrbSolverInterface](#).

8.32.4.196 `virtual void OsiSolverInterface::applyRowCut (const OsiRowCut & rc) [protected],[pure virtual]`

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

Implemented in [OsiCpxSolverInterface](#), [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiMskSolverInterface](#), [OsiXprSolverInterface](#), and [OsiSpxSolverInterface](#).

8.32.4.197 `virtual void OsiSolverInterface::applyColCut (const OsiColCut & cc) [protected],[pure virtual]`

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

Implemented in [OsiCpxSolverInterface](#), [OsiGrbSolverInterface](#), [OsiGlpkSolverInterface](#), [OsiMskSolverInterface](#), [OsiXprSolverInterface](#), and [OsiSpxSolverInterface](#).

8.32.4.198 `void OsiSolverInterface::convertBoundToSense (const double lower, const double upper, char & sense, double & right, double & range) const [inline],[protected]`

A quick inlined function to convert from the lb/ub style of constraint definition to the sense/rhs/range style.

Definition at line 2074 of file OsiSolverInterface.hpp.

```
8.32.4.199 void OsiSolverInterface::convertSenseToBound ( const char sense, const double right, const double range, double &
            lower, double & upper ) const [inline], [protected]
```

A quick inlined function to convert from the sense/rhs/range style of constraint definition to the lb/ub style.

Definition at line 2108 of file OsiSolverInterface.hpp.

```
8.32.4.200 template<class T > T OsiSolverInterface::forceIntoRange ( const T value, const T lower, const T upper ) const
            [inline], [protected]
```

A quick inlined function to force a value to be between a minimum and a maximum value.

Definition at line 1990 of file OsiSolverInterface.hpp.

```
8.32.4.201 void OsiSolverInterface::setInitialData ( ) [protected]
```

Set [OsiSolverInterface](#) object state for default constructor.

This routine establishes the initial values of data fields in the [OsiSolverInterface](#) object when the object is created using the default constructor.

8.32.5 Friends And Related Function Documentation

```
8.32.5.1 void OsiSolverInterfaceCommonUnitTest ( const OsiSolverInterface * emptySi, const std::string & mpsDir, const
            std::string & netlibDir ) [friend]
```

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

Some time ago, if this method is compiled with optimization, the compilation took 10-15 minutes and the machine pages (has 256M core memory!)...

```
8.32.5.2 void OsiSolverInterfaceMpsUnitTest ( const std::vector< OsiSolverInterface * > & vecSiP, const std::string & mpsDir
            ) [friend]
```

A function that tests that a lot of problems given in MPS files (mostly the NETLIB problems) solve properly with all the specified solvers.

The routine creates a vector of NetLib problems (problem name, objective, various other characteristics), and a vector of solvers to be tested.

Each solver is run on each problem. The run is deemed successful if the solver reports the correct problem size after loading and returns the correct objective value after optimization.

If multiple solvers are available, the results are compared pairwise against the results reported by adjacent solvers in the solver vector. Due to limitations of the volume solver, it must be the last solver in vecEmptySiP.

8.32.6 Member Data Documentation

```
8.32.6.1 OsiRowCutDebugger* OsiSolverInterface::rowCutDebugger_ [mutable], [protected]
```

Pointer to row cut debugger object.

Mutable so that we can update the solution held in the debugger while maintaining const'ness for the Osi object.

Definition at line 2009 of file OsiSolverInterface.hpp.

8.32.6.2 `CoinMessageHandler* OsiSolverInterface::handler_` `[protected]`

Message handler.

Definition at line 2012 of file OsiSolverInterface.hpp.

8.32.6.3 `bool OsiSolverInterface::defaultHandler_` `[protected]`

Flag to say if the current handler is the default handler.

Indicates if the solver interface object is responsible for destruction of the handler (true) or if the client is responsible (false).

Definition at line 2018 of file OsiSolverInterface.hpp.

8.32.6.4 `CoinMessages OsiSolverInterface::messages_` `[protected]`

Messages.

Definition at line 2020 of file OsiSolverInterface.hpp.

8.32.6.5 `int OsiSolverInterface::numberIntegers_` `[protected]`

Number of integers.

Definition at line 2022 of file OsiSolverInterface.hpp.

8.32.6.6 `int OsiSolverInterface::numberObjects_` `[protected]`

Total number of objects.

Definition at line 2024 of file OsiSolverInterface.hpp.

8.32.6.7 `OsiObject** OsiSolverInterface::object_` `[protected]`

Integer and ... information (integer info normally at beginning)

Definition at line 2027 of file OsiSolverInterface.hpp.

8.32.6.8 `char* OsiSolverInterface::columnType_` `[mutable], [protected]`

Column type 0 - continuous 1 - binary (may get fixed later) 2 - general integer (may get fixed later)

Definition at line 2033 of file OsiSolverInterface.hpp.

8.32.6.9 `OsiAuxInfo* OsiSolverInterface::appDataEtc_` `[private]`

Pointer to user-defined data structure - and more if user wants.

Definition at line 2043 of file OsiSolverInterface.hpp.

8.32.6.10 `int OsiSolverInterface::intParam_[OsiLastIntParam]` `[private]`

Array of integer parameters.

Definition at line 2045 of file OsiSolverInterface.hpp.

8.32.6.11 `double OsiSolverInterface::dblParam_[OsiLastDbiParam]` `[private]`

Array of double parameters.

Definition at line 2047 of file OsiSolverInterface.hpp.

8.32.6.12 `std::string OsiSolverInterface::strParam_[OsiLastStrParam]` [private]

Array of string parameters.

Definition at line 2049 of file `OsiSolverInterface.hpp`.

8.32.6.13 `bool OsiSolverInterface::hintParam_[OsiLastHintParam]` [private]

Array of hint parameters.

Definition at line 2051 of file `OsiSolverInterface.hpp`.

8.32.6.14 `OsiHintStrength OsiSolverInterface::hintStrength_[OsiLastHintParam]` [private]

Array of hint strengths.

Definition at line 2053 of file `OsiSolverInterface.hpp`.

8.32.6.15 `CoinWarmStart* OsiSolverInterface::ws_` [private]

Warm start information used for hot starts when the default hot start implementation is used.

Definition at line 2056 of file `OsiSolverInterface.hpp`.

8.32.6.16 `std::vector<double> OsiSolverInterface::strictColSolution_` [private]

Column solution satisfying lower and upper column bounds.

Definition at line 2058 of file `OsiSolverInterface.hpp`.

8.32.6.17 `OsiNameVec OsiSolverInterface::rowNames_` [private]

Row names.

Definition at line 2061 of file `OsiSolverInterface.hpp`.

8.32.6.18 `OsiNameVec OsiSolverInterface::colNames_` [private]

Column names.

Definition at line 2063 of file `OsiSolverInterface.hpp`.

8.32.6.19 `std::string OsiSolverInterface::objName_` [private]

Objective name.

Definition at line 2065 of file `OsiSolverInterface.hpp`.

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

- `/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverInterface.hpp`

8.33 OsiSolverResult Class Reference

Solver Result Class.

```
#include <OsiSolverBranch.hpp>
```

Public Member Functions

Add and Get methods

- void `createResult` (const `OsiSolverInterface` &solver, const double *lowerBefore, const double *upperBefore)
Create result.
- void `restoreResult` (`OsiSolverInterface` &solver) const
Restore result.
- const `CoinWarmStartBasis` & `basis` () const
Get basis.
- double `objectiveValue` () const
Objective value (as minimization)
- const double * `primalSolution` () const
Primal solution.
- const double * `dualSolution` () const
Dual solution.
- const `OsiSolverBranch` & `fixed` () const
Extra fixed.

Constructors and destructors

- `OsiSolverResult` ()
Default Constructor.
- `OsiSolverResult` (const `OsiSolverInterface` &solver, const double *lowerBefore, const double *upperBefore)
Constructor from solver.
- `OsiSolverResult` (const `OsiSolverResult` &rhs)
Copy constructor.
- `OsiSolverResult` & `operator=` (const `OsiSolverResult` &rhs)
Assignment operator.
- `~OsiSolverResult` ()
Destructor.

Private Attributes

Private member data

- double `objectiveValue_`
Value of objective (if \geq `OsiSolverInterface::getInfinity()` then infeasible)
- `CoinWarmStartBasis` `basis_`
Warm start information.
- double * `primalSolution_`
Primal solution (numberColumns)
- double * `dualSolution_`
Dual solution (numberRows)
- `OsiSolverBranch` `fixed_`
Which extra variables have been fixed (only way== -1 counts)

8.33.1 Detailed Description

Solver Result Class.

This provides information on a result as a set of tighter bounds on both ways

Definition at line 83 of file `OsiSolverBranch.hpp`.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 `OsiSolverResult::OsiSolverResult ()`

Default Constructor.

8.33.2.2 `OsiSolverResult::OsiSolverResult (const OsiSolverInterface & solver, const double * lowerBefore, const double * upperBefore)`

Constructor from solver.

8.33.2.3 `OsiSolverResult::OsiSolverResult (const OsiSolverResult & rhs)`

Copy constructor.

8.33.2.4 `OsiSolverResult::~~OsiSolverResult ()`

Destructor.

8.33.3 Member Function Documentation

8.33.3.1 `void OsiSolverResult::createResult (const OsiSolverInterface & solver, const double * lowerBefore, const double * upperBefore)`

Create result.

8.33.3.2 `void OsiSolverResult::restoreResult (OsiSolverInterface & solver) const`

Restore result.

8.33.3.3 `const CoinWarmStartBasis& OsiSolverResult::basis () const [inline]`

Get basis.

Definition at line 96 of file `OsiSolverBranch.hpp`.

8.33.3.4 `double OsiSolverResult::objectiveValue () const [inline]`

Objective value (as minimization)

Definition at line 100 of file `OsiSolverBranch.hpp`.

8.33.3.5 `const double* OsiSolverResult::primalSolution () const [inline]`

Primal solution.

Definition at line 104 of file `OsiSolverBranch.hpp`.

8.33.3.6 `const double* OsiSolverResult::dualSolution () const [inline]`

Dual solution.

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

8.33.3.7 `const OsiSolverBranch& OsiSolverResult::fixed () const [inline]`

Extra fixed.

Definition at line 112 of file `OsiSolverBranch.hpp`.

8.33.3.8 OsiSolverResult& OsiSolverResult::operator= (const OsiSolverResult & rhs)

Assignment operator.

8.33.4 Member Data Documentation

8.33.4.1 double OsiSolverResult::objectiveValue_ [private]

Value of objective (if \geq [OsiSolverInterface::getInfinity\(\)](#) then infeasible)

Definition at line 141 of file OsiSolverBranch.hpp.

8.33.4.2 CoinWarmStartBasis OsiSolverResult::basis_ [private]

Warm start information.

Definition at line 143 of file OsiSolverBranch.hpp.

8.33.4.3 double* OsiSolverResult::primalSolution_ [private]

Primal solution (numberColumns)

Definition at line 145 of file OsiSolverBranch.hpp.

8.33.4.4 double* OsiSolverResult::dualSolution_ [private]

Dual solution (numberRows)

Definition at line 147 of file OsiSolverBranch.hpp.

8.33.4.5 OsiSolverBranch OsiSolverResult::fixed_ [private]

Which extra variables have been fixed (only way== -1 counts)

Definition at line 149 of file OsiSolverBranch.hpp.

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

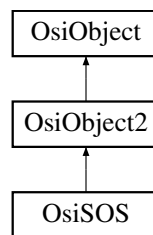
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiSolverBranch.hpp](#)

8.34 OsiSOS Class Reference

Define Special Ordered Sets of type 1 and 2.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for OsiSOS:



Public Member Functions

- `OsiSOS ()`
- `OsiSOS (const OsiSolverInterface *solver, int numberMembers, const int *which, const double *weights, int type=1)`
Useful constructor - which are indices and weights are also given.
- `OsiSOS (const OsiSOS &)`
- `virtual OsiObject * clone () const`
Clone.
- `OsiSOS & operator= (const OsiSOS &rhs)`
- `virtual ~OsiSOS ()`
- `virtual double infeasibility (const OsiBranchingInformation *info, int &whichWay) const`
Infeasibility - large is 0.5.
- `virtual double feasibleRegion (OsiSolverInterface *solver, const OsiBranchingInformation *info) const`
Set bounds to fix the variable at the current (integer) value.
- `virtual OsiBranchingObject * createBranch (OsiSolverInterface *solver, const OsiBranchingInformation *info, int way) const`
Creates a branching object.
- `virtual double upEstimate () const`
Return "up" estimate (default 1.0e-5)
- `virtual double downEstimate () const`
Return "down" estimate (default 1.0e-5)
- `virtual void resetSequenceEtc (int numberColumns, const int *originalColumns)`
Redoes data when sequence numbers change.
- `int numberMembers () const`
Number of members.
- `const int * members () const`
Members (indices in range 0 ... numberColumns-1)
- `int sosType () const`
SOS type.
- `int setType () const`
SOS type.
- `const double * weights () const`
Array of weights.
- `virtual bool canDoHeuristics () const`
Return true if object can take part in normal heuristics.
- `void setIntegerValued (bool yesNo)`
Set whether set is integer valued or not.
- `virtual bool canHandleShadowPrices () const`
Return true if knows how to deal with Pseudo Shadow Prices.
- `void setNumberMembers (int value)`
Set number of members.
- `int * mutableMembers () const`
Members (indices in range 0 ... numberColumns-1)
- `void setSosType (int value)`
Set SOS type.
- `double * mutableWeights () const`
Array of weights.

Protected Attributes

- int * [members_](#)
data
- double * [weights_](#)
Weights.
- int [numberMembers_](#)
Number of members.
- int [sosType_](#)
SOS type.
- bool [integerValued_](#)
Whether integer valued.

8.34.1 Detailed Description

Define Special Ordered Sets of type 1 and 2.

These do not have to be integer - so do not appear in lists of integers.

`which_` points columns of matrix

Definition at line 674 of file `OsiBranchingObject.hpp`.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 `OsiSOS::OsiSOS ()`

8.34.2.2 `OsiSOS::OsiSOS (const OsiSolverInterface * solver, int numberMembers, const int * which, const double * weights, int type = 1)`

Useful constructor - which are indices and weights are also given.

If null then 0,1,2.. type is SOS type

8.34.2.3 `OsiSOS::OsiSOS (const OsiSOS &)`8.34.2.4 `virtual OsiSOS::~OsiSOS () [virtual]`

8.34.3 Member Function Documentation

8.34.3.1 `virtual OsiObject* OsiSOS::clone () const [virtual]`

Clone.

Implements [OsiObject](#).

8.34.3.2 `OsiSOS& OsiSOS::operator= (const OsiSOS & rhs)`8.34.3.3 `virtual double OsiSOS::infeasibility (const OsiBranchingInformation * info, int & whichWay) const [virtual]`

Infeasibility - large is 0.5.

Implements [OsiObject](#).

8.34.3.4 `virtual double OsiSOS::feasibleRegion (OsiSolverInterface * solver, const OsiBranchingInformation * info) const [virtual]`

Set bounds to fix the variable at the current (integer) value.

Given an integer value, set the lower and upper bounds to fix the variable. Returns amount it had to move variable.

Implements [OsiObject](#).

8.34.3.5 `virtual OsiBranchingObject* OsiSOS::createBranch (OsiSolverInterface * solver, const OsiBranchingInformation * info, int way) const [virtual]`

Creates a branching object.

The preferred direction is set by `way`, 0 for down, 1 for up.

Reimplemented from [OsiObject](#).

8.34.3.6 `virtual double OsiSOS::upEstimate () const [virtual]`

Return "up" estimate (default 1.0e-5)

Reimplemented from [OsiObject](#).

8.34.3.7 `virtual double OsiSOS::downEstimate () const [virtual]`

Return "down" estimate (default 1.0e-5)

Reimplemented from [OsiObject](#).

8.34.3.8 `virtual void OsiSOS::resetSequenceEtc (int numberColumns, const int * originalColumns) [virtual]`

Redoes data when sequence numbers change.

Reimplemented from [OsiObject](#).

8.34.3.9 `int OsiSOS::numberMembers () const [inline]`

Number of members.

Definition at line 726 of file `OsiBranchingObject.hpp`.

8.34.3.10 `const int* OsiSOS::members () const [inline]`

Members (indices in range 0 ... `numberColumns-1`)

Definition at line 730 of file `OsiBranchingObject.hpp`.

8.34.3.11 `int OsiSOS::sosType () const [inline]`

SOS type.

Definition at line 734 of file `OsiBranchingObject.hpp`.

8.34.3.12 `int OsiSOS::setType () const [inline]`

SOS type.

Definition at line 738 of file `OsiBranchingObject.hpp`.

8.34.3.13 `const double* OsiSOS::weights () const [inline]`

Array of weights.

Definition at line 742 of file OsiBranchingObject.hpp.

8.34.3.14 `virtual bool OsiSOS::canDoHeuristics () const [inline],[virtual]`

Return true if object can take part in normal heuristics.

Reimplemented from [OsiObject](#).

Definition at line 747 of file OsiBranchingObject.hpp.

8.34.3.15 `void OsiSOS::setIntegerValued (bool yesNo) [inline]`

Set whether set is integer valued or not.

Definition at line 750 of file OsiBranchingObject.hpp.

8.34.3.16 `virtual bool OsiSOS::canHandleShadowPrices () const [inline],[virtual]`

Return true if knows how to deal with Pseudo Shadow Prices.

Reimplemented from [OsiObject](#).

Definition at line 753 of file OsiBranchingObject.hpp.

8.34.3.17 `void OsiSOS::setNumberMembers (int value) [inline]`

Set number of members.

Definition at line 756 of file OsiBranchingObject.hpp.

8.34.3.18 `int* OsiSOS::mutableMembers () const [inline]`

Members (indices in range 0 ... numberColumns-1)

Definition at line 760 of file OsiBranchingObject.hpp.

8.34.3.19 `void OsiSOS::setSosType (int value) [inline]`

Set SOS type.

Definition at line 764 of file OsiBranchingObject.hpp.

8.34.3.20 `double* OsiSOS::mutableWeights () const [inline]`

Array of weights.

Definition at line 768 of file OsiBranchingObject.hpp.

8.34.4 Member Data Documentation

8.34.4.1 `int* OsiSOS::members_ [protected]`

data

Members (indices in range 0 ... numberColumns-1)

Definition at line 774 of file OsiBranchingObject.hpp.

8.34.4.2 `double* OsiSOS::weights_` [protected]

Weights.

Definition at line 776 of file `OsiBranchingObject.hpp`.

8.34.4.3 `int OsiSOS::numberMembers_` [protected]

Number of members.

Definition at line 779 of file `OsiBranchingObject.hpp`.

8.34.4.4 `int OsiSOS::sosType_` [protected]

SOS type.

Definition at line 781 of file `OsiBranchingObject.hpp`.

8.34.4.5 `bool OsiSOS::integerValued_` [protected]

Whether integer valued.

Definition at line 783 of file `OsiBranchingObject.hpp`.

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

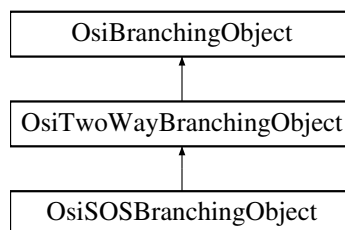
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp](#)

8.35 OsiSOSBranchingObject Class Reference

Branching object for Special ordered sets.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for `OsiSOSBranchingObject`:



Public Member Functions

- `OsiSOSBranchingObject` ()
- `OsiSOSBranchingObject` (`OsiSolverInterface` *solver, const `OsiSOS` *originalObject, int way, double separator)
- `OsiSOSBranchingObject` (const `OsiSOSBranchingObject` &)
- `OsiSOSBranchingObject` & operator= (const `OsiSOSBranchingObject` &rhs)
- virtual `OsiBranchingObject` * clone () const
Clone.
- virtual `~OsiSOSBranchingObject` ()
- virtual double branch (`OsiSolverInterface` *solver)
Does next branch and updates state.

- virtual void `print` (const [OsiSolverInterface](#) *solver=NULL)

Print something about branch - only if log level high.

Additional Inherited Members

8.35.1 Detailed Description

Branching object for Special ordered sets.

Definition at line 789 of file `OsiBranchingObject.hpp`.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 `OsiSOSBranchingObject::OsiSOSBranchingObject ()`

8.35.2.2 `OsiSOSBranchingObject::OsiSOSBranchingObject (OsiSolverInterface * solver, const OsiSOS * originalObject, int way, double separator)`

8.35.2.3 `OsiSOSBranchingObject::OsiSOSBranchingObject (const OsiSOSBranchingObject &)`

8.35.2.4 `virtual OsiSOSBranchingObject::~OsiSOSBranchingObject ()` [virtual]

8.35.3 Member Function Documentation

8.35.3.1 `OsiSOSBranchingObject& OsiSOSBranchingObject::operator= (const OsiSOSBranchingObject & rhs)`

8.35.3.2 `virtual OsiBranchingObject* OsiSOSBranchingObject::clone () const` [virtual]

Clone.

Implements [OsiBranchingObject](#).

8.35.3.3 `virtual double OsiSOSBranchingObject::branch (OsiSolverInterface * solver)` [virtual]

Does next branch and updates state.

Implements [OsiTwoWayBranchingObject](#).

8.35.3.4 `virtual void OsiSOSBranchingObject::print (const OsiSolverInterface * solver = NULL)` [virtual]

Print something about branch - only if log level high.

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

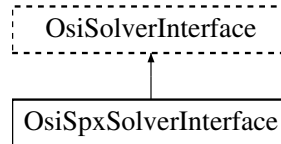
- `/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp`

8.36 OsiSpxSolverInterface Class Reference

SoPlex Solver Interface Instantiation of [OsiSpxSolverInterface](#) for SoPlex.

```
#include <OsiSpxSolverInterface.hpp>
```

Inheritance diagram for `OsiSpxSolverInterface`:



Public Member Functions

- virtual void [setObjSense](#) (double s)
Set objective function sense (1 for min (default), -1 for max.)
- virtual void [setColSolution](#) (const double *colsol)
Set the primal solution column values.
- virtual void [setRowPrice](#) (const double *rowprice)
Set dual solution vector.

Solve methods

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

Parameter set/get methods

The set methods return true if the parameter was set to the given value, false otherwise.

There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the volume algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.

The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.

- bool [setIntParam](#) ([OsiIntParam](#) key, int value)
Set an integer parameter.
- bool [setDbiParam](#) ([OsiDbiParam](#) key, double value)
Set a double parameter.
- bool [getIntParam](#) ([OsiIntParam](#) key, int &value) const
Get an integer parameter.
- bool [getDbiParam](#) ([OsiDbiParam](#) key, double &value) const
Get a double parameter.
- bool [getStrParam](#) ([OsiStrParam](#) key, std::string &value) const
Get a string parameter.
- void [setTimeLimit](#) (double value)
- double [getTimeLimit](#) () const

Methods returning info on how the solution process terminated

- virtual bool [isAbandoned](#) () const
Are there a numerical difficulties?
- virtual bool [isProvenOptimal](#) () const

- Is optimality proven?*
- virtual bool [isProvenPrimalInfeasible](#) () const
- Is primal infeasibility proven?*
- virtual bool [isProvenDualInfeasible](#) () const
- Is dual infeasibility proven?*
- virtual bool [isDualObjectiveLimitReached](#) () const
- Is the given dual objective limit reached?*
- virtual bool [isIterationLimitReached](#) () const
- Iteration limit reached?*
- virtual bool [isTimeLimitReached](#) () const
- Time limit reached?*

WarmStart related methods

- CoinWarmStart * [getEmptyWarmStart](#) () const
- Get empty warm start object.*
- virtual CoinWarmStart * [getWarmStart](#) () const
- Get warmstarting information.*
- virtual bool [setWarmStart](#) (const CoinWarmStart *warmstart)
- Set warmstarting information.*

Hotstart related methods (primarily used in strong branching).

The user can create a hotstart (a snapshot) of the optimization process then reoptimize over and over again always starting from there.

NOTE: between hotstarted optimizations only bound changes are allowed.

- virtual void [markHotStart](#) ()
- Create a hotstart point of the optimization process.*
- virtual void [solveFromHotStart](#) ()
- Optimize starting from the hotstart.*
- virtual void [unmarkHotStart](#) ()
- Delete the snapshot.*

Methods related to querying the input data

- virtual int [getNumCols](#) () const
- Get number of columns.*
- virtual int [getNumRows](#) () const
- Get number of rows.*
- virtual int [getNumElements](#) () const
- Get number of nonzero elements.*
- virtual const double * [getColLower](#) () const
- Get pointer to array[getNumCols()] of column lower bounds.*
- virtual const double * [getColUpper](#) () const
- Get pointer to array[getNumCols()] of column upper bounds.*
- virtual const char * [getRowSense](#) () const
- Get pointer to array[getNumRows()] of row constraint senses.*
- virtual const double * [getRightHandSide](#) () const
- Get pointer to array[getNumRows()] of rows right-hand sides.*
- virtual const double * [getRowRange](#) () const
- Get pointer to array[getNumRows()] of row ranges.*
- virtual const double * [getRowLower](#) () const
- Get pointer to array[getNumRows()] of row lower bounds.*
- virtual const double * [getRowUpper](#) () const

- *Get pointer to array[getNumRows()] of row upper bounds.*
- virtual const double * [getObjCoefficients](#) () const
- *Get pointer to array[getNumCols()] of objective function coefficients.*
- virtual double [getObjSense](#) () const
- *Get objective function sense (1 for min (default), -1 for max)*
- virtual bool [isContinuous](#) (int colNumber) const
- *Return true if column is continuous.*
- virtual const CoinPackedMatrix * [getMatrixByRow](#) () const
- *Get pointer to row-wise copy of matrix.*
- virtual const CoinPackedMatrix * [getMatrixByCol](#) () const
- *Get pointer to column-wise copy of matrix.*
- virtual double [getInfinity](#) () const
- *Get solver's value for infinity.*

Methods related to querying the solution

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

Changing bounds on variables and constraints

- virtual void [setObjCoeff](#) (int elementIndex, double elementValue)
- *Set an objective function coefficient.*
- virtual void [setColLower](#) (int elementIndex, double elementValue)
- *Set a single column lower bound*
- *Use -COIN_DBL_MAX for -infinity.*
- virtual void [setColUpper](#) (int elementIndex, double elementValue)
- *Set a single column upper bound*
- *Use COIN_DBL_MAX for infinity.*
- virtual void [setColBounds](#) (int elementIndex, double lower, double upper)
- *Set a single column lower and upper bound*
- *The default implementation just invokes setColLower and setColUpper*
- virtual void [setRowLower](#) (int elementIndex, double elementValue)
- *Set a single row lower bound*
- *Use -COIN_DBL_MAX for -infinity.*
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)
- *Set a single row upper bound*
- *Use COIN_DBL_MAX for infinity.*
- virtual void [setRowBounds](#) (int elementIndex, double lower, double upper)
- *Set a single row lower and upper bound*
- *The default implementation just invokes setRowLower and setRowUpper*

- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)
Set the type of a single row

Integrality related changing methods

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

Methods to expand a problem.

Note that if a column is added then by default it will correspond to a continuous variable.

- virtual void [addCol](#) (const CoinPackedVectorBase &vec, const double collb, const double colub, const double obj)
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)
Add a row (constraint) to the problem.
- virtual void [deleteRows](#) (const int num, const int *rowIndices)
Delete a set of rows (constraints) from the problem.

Methods to input a problem

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

Write the problem into an mps file of the given filename.

Constructors and destructor

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

Protected Member Functions

Protected methods

- virtual void [applyRowCut](#) (const [OsiRowCut](#) &rc)
Apply a row cut. Return true if cut was applied.
- virtual void [applyColCut](#) (const [OsiColCut](#) &cc)
Apply a column cut (bound adjustment).

Protected Attributes

Protected member data

- [soplex::SoPlex](#) * [soplex_](#)
SoPlex solver object.

Private Attributes

Private member data

- [soplex::DIdxSet](#) * [spxintvars_](#)
indices of integer variables
- void * [hotStartCStat_](#)
Hotstart information.
- int [hotStartCStatSize_](#)
- void * [hotStartRStat_](#)
- int [hotStartRStatSize_](#)
- int [hotStartMaxIteration_](#)

Cached information derived from the SoPlex model

- [soplex::DVector](#) * [obj_](#)
Pointer to objective Vector.
- char * [rowsense_](#)
Pointer to dense vector of row sense indicators.
- double * [rhs_](#)
Pointer to dense vector of row right-hand side values.
- double * [rowrange_](#)

- *Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)*
- `soplex::DVector * colsol_`
Pointer to primal solution vector.
- `soplex::DVector * rowsol_`
Pointer to dual solution vector.
- `soplex::DVector * redcost_`
Pointer to reduced cost vector.
- `soplex::DVector * rowact_`
Pointer to row activity (slack) vector.
- `CoinPackedMatrix * matrixByRow_`
Pointer to row-wise copy of problem matrix coefficients.
- `CoinPackedMatrix * matrixByCol_`
Pointer to row-wise copy of problem matrix coefficients.

Friends

- `void OsiSpxSolverInterfaceUnitTest (const std::string &mpsDir, const std::string &netlibDir)`
A function that tests the methods in the [OsiSpxSolverInterface](#) class.

Private methods

- `enum keepCachedFlag {
KEEPCACHED_NONE = 0, KEEPCACHED_COLUMN = 1, KEEPCACHED_ROW = 2, KEEPCACHED_MATRIX = 4,
KEEPCACHED_RESULTS = 8, KEEPCACHED_PROBLEM = KEEPCACHED_COLUMN | KEEPCACHED_ROW | KEEPCACHED_MATRIX, KEEPCACHED_ALL = KEEPCACHED_PROBLEM | KEEPCACHED_RESULTS,
FREECACHED_COLUMN = KEEPCACHED_PROBLEM & ~KEEPCACHED_COLUMN,
FREECACHED_ROW = KEEPCACHED_PROBLEM & ~KEEPCACHED_ROW, FREECACHED_MATRIX = KEEPCACHED_PROBLEM & ~KEEPCACHED_MATRIX, FREECACHED_RESULTS = KEEPCACHED_ALL & ~KEEPCACHED_RESULTS }`
- `void freeCachedColRim ()`
free cached column rim vectors
- `void freeCachedRowRim ()`
free cached row rim vectors
- `void freeCachedResults ()`
free cached result vectors
- `void freeCachedMatrix ()`
free cached matrices
- `void freeCachedData (int keepCached=KEEPCACHED_NONE)`
free all cached data (except specified entries, see [getLpPtr\(\)](#))
- `void freeAllMemory ()`
free all allocated memory

Additional Inherited Members

8.36.1 Detailed Description

SoPlex Solver Interface Instantiation of [OsiSpxSolverInterface](#) for SoPlex.

Definition at line 32 of file [OsiSpxSolverInterface.hpp](#).

8.36.2 Member Enumeration Documentation

8.36.2.1 enum `OsiSpxSolverInterface::keepCachedFlag` [private]

Enumerator

KEEPCACHED_NONE discard all cached data (default)
KEEPCACHED_COLUMN column information: objective values, lower and upper bounds, variable types
KEEPCACHED_ROW row information: right hand sides, ranges and senses, lower and upper bounds for row
KEEPCACHED_MATRIX problem matrix: matrix ordered by column and by row
KEEPCACHED_RESULTS LP solution: primal and dual solution, reduced costs, row activities.
KEEPCACHED_PROBLEM only discard cached LP solution
KEEPCACHED_ALL keep all cached data (similar to `getMutableLpPtr()`)
FREECACHED_COLUMN free only cached column and LP solution information
FREECACHED_ROW free only cached row and LP solution information
FREECACHED_MATRIX free only cached matrix and LP solution information
FREECACHED_RESULTS free only cached LP solution information

Definition at line 648 of file `OsiSpxSolverInterface.hpp`.

8.36.3 Constructor & Destructor Documentation

8.36.3.1 `OsiSpxSolverInterface::OsiSpxSolverInterface ()`

Default Constructor.

8.36.3.2 `OsiSpxSolverInterface::OsiSpxSolverInterface (const OsiSpxSolverInterface &)`

Copy constructor.

8.36.3.3 `virtual OsiSpxSolverInterface::~~OsiSpxSolverInterface ()` [virtual]

Destructor.

8.36.4 Member Function Documentation

8.36.4.1 `virtual void OsiSpxSolverInterface::initialSolve ()` [virtual]

Solve initial LP relaxation.

Implements [OsiSolverInterface](#).

8.36.4.2 `virtual void OsiSpxSolverInterface::resolve ()` [virtual]

Resolve an LP relaxation after problem modification.

Implements [OsiSolverInterface](#).

8.36.4.3 `virtual void OsiSpxSolverInterface::branchAndBound ()` [virtual]

Invoke solver's built-in enumeration algorithm.

Implements [OsiSolverInterface](#).

8.36.4.4 `bool OsiSpxSolverInterface::setIntParam (OsiIntParam key, int value) [virtual]`

Set an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.36.4.5 `bool OsiSpxSolverInterface::setDbIParam (OsiDbIParam key, double value) [virtual]`

Set a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.36.4.6 `bool OsiSpxSolverInterface::getIntParam (OsiIntParam key, int & value) const [virtual]`

Get an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.36.4.7 `bool OsiSpxSolverInterface::getDbIParam (OsiDbIParam key, double & value) const [virtual]`

Get a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.36.4.8 `bool OsiSpxSolverInterface::getStrParam (OsiStrParam key, std::string & value) const [virtual]`

Get a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.36.4.9 `void OsiSpxSolverInterface::setTimeLimit (double value)`

8.36.4.10 `double OsiSpxSolverInterface::getTimeLimit () const`

8.36.4.11 `virtual bool OsiSpxSolverInterface::isAbandoned () const [virtual]`

Are there a numerical difficulties?

Implements [OsiSolverInterface](#).

8.36.4.12 `virtual bool OsiSpxSolverInterface::isProvenOptimal () const [virtual]`

Is optimality proven?

Implements [OsiSolverInterface](#).

8.36.4.13 `virtual bool OsiSpxSolverInterface::isProvenPrimalInfeasible () const [virtual]`

Is primal infeasibility proven?

Implements [OsiSolverInterface](#).

8.36.4.14 `virtual bool OsiSpxSolverInterface::isProvenDualInfeasible () const [virtual]`

Is dual infeasibility proven?

Implements [OsiSolverInterface](#).

8.36.4.15 `virtual bool OsiSpxSolverInterface::isDualObjectiveLimitReached () const [virtual]`

Is the given dual objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.36.4.16 `virtual bool OsiSpxSolverInterface::isIterationLimitReached () const [virtual]`

Iteration limit reached?

Implements [OsiSolverInterface](#).

8.36.4.17 `virtual bool OsiSpxSolverInterface::isTimeLimitReached () const [virtual]`

Time limit reached?

8.36.4.18 `CoinWarmStart* OsiSpxSolverInterface::getEmptyWarmStart () const [inline],[virtual]`

Get empty warm start object.

Implements [OsiSolverInterface](#).

Definition at line 106 of file `OsiSpxSolverInterface.hpp`.

8.36.4.19 `virtual CoinWarmStart* OsiSpxSolverInterface::getWarmStart () const [virtual]`

Get warmstarting information.

Implements [OsiSolverInterface](#).

8.36.4.20 `virtual bool OsiSpxSolverInterface::setWarmStart (const CoinWarmStart * warmstart) [virtual]`

Set warmstarting information.

Return true/false depending on whether the warmstart information was accepted or not.

Implements [OsiSolverInterface](#).

8.36.4.21 `virtual void OsiSpxSolverInterface::markHotStart () [virtual]`

Create a hotstart point of the optimization process.

Reimplemented from [OsiSolverInterface](#).

8.36.4.22 `virtual void OsiSpxSolverInterface::solveFromHotStart () [virtual]`

Optimize starting from the hotstart.

Reimplemented from [OsiSolverInterface](#).

8.36.4.23 `virtual void OsiSpxSolverInterface::unmarkHotStart () [virtual]`

Delete the snapshot.

Reimplemented from [OsiSolverInterface](#).

8.36.4.24 `virtual int OsiSpxSolverInterface::getNumCols () const [virtual]`

Get number of columns.

Implements [OsiSolverInterface](#).

8.36.4.25 `virtual int OsiSpxSolverInterface::getNumRows () const [virtual]`

Get number of rows.

Implements [OsiSolverInterface](#).

8.36.4.26 `virtual int OsiSpxSolverInterface::getNumElements () const [virtual]`

Get number of nonzero elements.

Implements [OsiSolverInterface](#).

8.36.4.27 `virtual const double* OsiSpxSolverInterface::getColLower () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column lower bounds.

Implements [OsiSolverInterface](#).

8.36.4.28 `virtual const double* OsiSpxSolverInterface::getColUpper () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column upper bounds.

Implements [OsiSolverInterface](#).

8.36.4.29 `virtual const char* OsiSpxSolverInterface::getRowSense () const [virtual]`

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

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

Implements [OsiSolverInterface](#).

8.36.4.30 `virtual const double* OsiSpxSolverInterface::getRightHandSide () const [virtual]`

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

- if `rowsense()[i] == 'L'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'G'` then `rhs()[i] == rowlower()[i]`
- if `rowsense()[i] == 'R'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'N'` then `rhs()[i] == 0.0`

Implements [OsiSolverInterface](#).

8.36.4.31 `virtual const double* OsiSpxSolverInterface::getRowRange () const [virtual]`

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

- if `rowsense()[i] == 'R'` then `rowrange()[i] == rowupper()[i] - rowlower()[i]`
- if `rowsense()[i] != 'R'` then `rowrange()[i]` is 0.0

Implements [OsiSolverInterface](#).

8.36.4.32 `virtual const double* OsiSpxSolverInterface::getRowLower () const` [virtual]

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

Implements [OsiSolverInterface](#).

8.36.4.33 `virtual const double* OsiSpxSolverInterface::getRowUpper () const` [virtual]

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

Implements [OsiSolverInterface](#).

8.36.4.34 `virtual const double* OsiSpxSolverInterface::getObjCoefficients () const` [virtual]

Get pointer to array[[getNumCols\(\)](#)] of objective function coefficients.

Implements [OsiSolverInterface](#).

8.36.4.35 `virtual double OsiSpxSolverInterface::getObjSense () const` [virtual]

Get objective function sense (1 for min (default), -1 for max)

Implements [OsiSolverInterface](#).

8.36.4.36 `virtual bool OsiSpxSolverInterface::isContinuous (int colNumber) const` [virtual]

Return true if column is continuous.

Implements [OsiSolverInterface](#).

8.36.4.37 `virtual const CoinPackedMatrix* OsiSpxSolverInterface::getMatrixByRow () const` [virtual]

Get pointer to row-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.36.4.38 `virtual const CoinPackedMatrix* OsiSpxSolverInterface::getMatrixByCol () const` [virtual]

Get pointer to column-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.36.4.39 `virtual double OsiSpxSolverInterface::getInfinity () const` [virtual]

Get solver's value for infinity.

Implements [OsiSolverInterface](#).

8.36.4.40 `virtual const double* OsiSpxSolverInterface::getColSolution () const` [virtual]

Get pointer to array[[getNumCols\(\)](#)] of primal solution vector.

Implements [OsiSolverInterface](#).

8.36.4.41 `virtual const double* OsiSpxSolverInterface::getRowPrice () const` [virtual]

Get pointer to array[[getNumRows\(\)](#)] of dual prices.

Implements [OsiSolverInterface](#).

8.36.4.42 `virtual const double* OsiSpxSolverInterface::getReducedCost () const [virtual]`

Get a pointer to array[[getNumCols\(\)](#)] of reduced costs.

Implements [OsiSolverInterface](#).

8.36.4.43 `virtual const double* OsiSpxSolverInterface::getRowActivity () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.36.4.44 `virtual double OsiSpxSolverInterface::getObjValue () const [virtual]`

Get objective function value.

Implements [OsiSolverInterface](#).

8.36.4.45 `virtual int OsiSpxSolverInterface::getIterationCount () const [virtual]`

Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver).

Implements [OsiSolverInterface](#).

8.36.4.46 `virtual std::vector<double*> OsiSpxSolverInterface::getDualRays (int maxNumRays, bool fullRay = false) const [virtual]`

Get as many dual rays as the solver can provide.

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

The first [getNumRows\(\)](#) ray components will always be associated with the row duals (as returned by [getRowPrice\(\)](#)). If `fullRay` is true, the final [getNumCols\(\)](#) entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.36.4.47 `virtual std::vector<double*> OsiSpxSolverInterface::getPrimalRays (int maxNumRays) const [virtual]`

Get as many primal rays as the solver can provide.

(In case of proven dual infeasibility there should be at least one.)

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.36.4.48 `virtual void OsiSpxSolverInterface::setObjCoeff (int elementIndex, double elementValue) [virtual]`

Set an objective function coefficient.

Implements [OsiSolverInterface](#).

8.36.4.49 `virtual void OsiSpxSolverInterface::setColLower (int elementIndex, double elementValue) [virtual]`

Set a single column lower bound

Use -COIN_DBL_MAX for -infinity.

Implements [OsiSolverInterface](#).

8.36.4.50 `virtual void OsiSpxSolverInterface::setColUpper (int elementIndex, double elementValue) [virtual]`

Set a single column upper bound

Use COIN_DBL_MAX for infinity.

Implements [OsiSolverInterface](#).

8.36.4.51 `virtual void OsiSpxSolverInterface::setColBounds (int elementIndex, double lower, double upper) [virtual]`

Set a single column lower and upper bound

The default implementation just invokes `setColLower` and `setColUpper`

Reimplemented from [OsiSolverInterface](#).

8.36.4.52 `virtual void OsiSpxSolverInterface::setRowLower (int elementIndex, double elementValue) [virtual]`

Set a single row lower bound

Use -COIN_DBL_MAX for -infinity.

Implements [OsiSolverInterface](#).

8.36.4.53 `virtual void OsiSpxSolverInterface::setRowUpper (int elementIndex, double elementValue) [virtual]`

Set a single row upper bound

Use COIN_DBL_MAX for infinity.

Implements [OsiSolverInterface](#).

8.36.4.54 `virtual void OsiSpxSolverInterface::setRowBounds (int elementIndex, double lower, double upper) [virtual]`

Set a single row lower and upper bound

The default implementation just invokes `setRowLower` and `setRowUpper`

Reimplemented from [OsiSolverInterface](#).

8.36.4.55 `virtual void OsiSpxSolverInterface::setRowType (int index, char sense, double rightHandSide, double range) [virtual]`

Set the type of a single row

Implements [OsiSolverInterface](#).

8.36.4.56 `virtual void OsiSpxSolverInterface::setContinuous (int index) [virtual]`

Set the index-th variable to be a continuous variable.

Implements [OsiSolverInterface](#).

8.36.4.57 `virtual void OsiSpxSolverInterface::setInteger (int index) [virtual]`

Set the index-th variable to be an integer variable.

Implements [OsiSolverInterface](#).

8.36.4.58 `virtual void OsiSpxSolverInterface::setObjSense (double s) [virtual]`

Set objective function sense (1 for min (default), -1 for max,)

Implements [OsiSolverInterface](#).

8.36.4.59 `virtual void OsiSpxSolverInterface::setColSolution (const double * colsol) [virtual]`

Set the primal solution column values.

`colsol[numcols()]` is an array of values of the problem column variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of `colsol()` until changed by another call to `setColsol()` or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

Implements [OsiSolverInterface](#).

8.36.4.60 `virtual void OsiSpxSolverInterface::setRowPrice (const double * rowprice) [virtual]`

Set dual solution vector.

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

Implements [OsiSolverInterface](#).

8.36.4.61 `virtual void OsiSpxSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [virtual]`

Add a column (primal variable) to the problem.

Implements [OsiSolverInterface](#).

8.36.4.62 `virtual void OsiSpxSolverInterface::deleteCols (const int num, const int * colIndices) [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](#).

8.36.4.63 `virtual void OsiSpxSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.36.4.64 `virtual void OsiSpxSolverInterface::addRow (const CoinPackedVectorBase & vec, const char rowsen, const double rowrhs, const double rowrng) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.36.4.65 `virtual void OsiSpxSolverInterface::deleteRows (const int num, const int * rowIndices) [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](#).

8.36.4.66 `virtual void OsiSpxSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * colub, const double * colub, const double * obj, const double * rowlb, const double * rowub) [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
- `colub`: 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](#).

8.36.4.67 `virtual void OsiSpxSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& colub, double *& colub, double *& obj, double *& rowlb, double *& rowub) [virtual]`

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

For default values see the previous method.

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

Implements [OsiSolverInterface](#).

8.36.4.68 `virtual void OsiSpxSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * colub, const double * colub, const double * obj, const char * rowsen, const double * rowrhs, const double * rowrng) [virtual]`

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

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

- `colub`: all columns have upper bound infinity
- `colub`: 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](#).

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

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

For default values see the previous method.

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

Implements [OsiSolverInterface](#).

8.36.4.70 `virtual void OsiSpxSolverInterface::loadProblem (const int numcols, const int numRows, const int * start, const int * index, const double * value, const double * collb, const double * colub, const double * obj, const double * rowlb, const double * rowub) [virtual]`

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

8.36.4.71 `virtual void OsiSpxSolverInterface::loadProblem (const int numcols, const int numRows, const int * start, const int * index, const double * value, const double * collb, const double * colub, const double * obj, const char * rowsen, const double * rowrhs, const double * rowrng) [virtual]`

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

8.36.4.72 `virtual int OsiSpxSolverInterface::readMps (const char * filename, const char * extension = "mps") [virtual]`

Read an mps file from the given filename.

Reimplemented from [OsiSolverInterface](#).

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

Write the problem into an mps file of the given filename.

If objSense is non zero then -1.0 forces the code to write a maximization objective and +1.0 to write a minimization one.

If 0.0 then solver can do what it wants

Implements [OsiSolverInterface](#).

8.36.4.74 `virtual OsiSolverInterface* OsiSpxSolverInterface::clone (bool copyData = true) const [virtual]`

Clone.

Implements [OsiSolverInterface](#).

8.36.4.75 `OsiSpxSolverInterface& OsiSpxSolverInterface::operator= (const OsiSpxSolverInterface & rhs)`

Assignment operator.

8.36.4.76 `virtual void OsiSpxSolverInterface::applyRowCut (const OsiRowCut & rc) [protected], [virtual]`

Apply a row cut. Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.36.4.77 `virtual void OsiSpxSolverInterface::applyColCut (const OsiColCut & cc) [protected], [virtual]`

Apply a column cut (bound adjustment).

Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.36.4.78 `void OsiSpxSolverInterface::freeCachedColRim () [private]`

free cached column rim vectors

8.36.4.79 `void OsiSpxSolverInterface::freeCachedRowRim () [private]`

free cached row rim vectors

8.36.4.80 `void OsiSpxSolverInterface::freeCachedResults () [private]`

free cached result vectors

8.36.4.81 `void OsiSpxSolverInterface::freeCachedMatrix () [private]`

free cached matrices

8.36.4.82 `void OsiSpxSolverInterface::freeCachedData (int keepCached = KEEPCACHED_NONE) [private]`

free all cached data (except specified entries, see `getLpPtr()`)

8.36.4.83 `void OsiSpxSolverInterface::freeAllMemory () [private]`

free all allocated memory

8.36.5 Friends And Related Function Documentation

8.36.5.1 `void OsiSpxSolverInterfaceUnitTest (const std::string & mpsDir, const std::string & netlibDir) [friend]`

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

8.36.6 Member Data Documentation

8.36.6.1 `soplex::SoPlex* OsiSpxSolverInterface::soplex_ [protected]`

SoPlex solver object.

Definition at line 628 of file `OsiSpxSolverInterface.hpp`.

8.36.6.2 `soplex::DIdxSet* OsiSpxSolverInterface::spxintvars_ [private]`

indices of integer variables

Definition at line 685 of file `OsiSpxSolverInterface.hpp`.

8.36.6.3 `void* OsiSpxSolverInterface::hotStartCStat_ [private]`

Hotstart information.

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

8.36.6.4 `int OsiSpxSolverInterface::hotStartCStatSize_ [private]`

Definition at line 689 of file `OsiSpxSolverInterface.hpp`.

8.36.6.5 `void* OsiSpxSolverInterface::hotStartRStat_` [private]

Definition at line 690 of file OsiSpxSolverInterface.hpp.

8.36.6.6 `int OsiSpxSolverInterface::hotStartRStatSize_` [private]

Definition at line 691 of file OsiSpxSolverInterface.hpp.

8.36.6.7 `int OsiSpxSolverInterface::hotStartMaxIteration_` [private]

Definition at line 692 of file OsiSpxSolverInterface.hpp.

8.36.6.8 `soplex::DVector* OsiSpxSolverInterface::obj_` [mutable], [private]

Pointer to objective Vector.

Definition at line 697 of file OsiSpxSolverInterface.hpp.

8.36.6.9 `char* OsiSpxSolverInterface::rowsense_` [mutable], [private]

Pointer to dense vector of row sense indicators.

Definition at line 700 of file OsiSpxSolverInterface.hpp.

8.36.6.10 `double* OsiSpxSolverInterface::rhs_` [mutable], [private]

Pointer to dense vector of row right-hand side values.

Definition at line 703 of file OsiSpxSolverInterface.hpp.

8.36.6.11 `double* OsiSpxSolverInterface::rowrange_` [mutable], [private]

Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)

Definition at line 706 of file OsiSpxSolverInterface.hpp.

8.36.6.12 `soplex::DVector* OsiSpxSolverInterface::colsol_` [mutable], [private]

Pointer to primal solution vector.

Definition at line 709 of file OsiSpxSolverInterface.hpp.

8.36.6.13 `soplex::DVector* OsiSpxSolverInterface::rowsol_` [mutable], [private]

Pointer to dual solution vector.

Definition at line 712 of file OsiSpxSolverInterface.hpp.

8.36.6.14 `soplex::DVector* OsiSpxSolverInterface::redcost_` [mutable], [private]

Pointer to reduced cost vector.

Definition at line 715 of file OsiSpxSolverInterface.hpp.

8.36.6.15 `soplex::DVector* OsiSpxSolverInterface::rowact_` [mutable], [private]

Pointer to row activity (slack) vector.

Definition at line 718 of file OsiSpxSolverInterface.hpp.

8.36.6.16 `CoinPackedMatrix* OsiSpxSolverInterface::matrixByRow_` [mutable], [private]

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 721 of file `OsiSpxSolverInterface.hpp`.

8.36.6.17 `CoinPackedMatrix* OsiSpxSolverInterface::matrixByCol_` [mutable], [private]

Pointer to row-wise copy of problem matrix coefficients.

Definition at line 724 of file `OsiSpxSolverInterface.hpp`.

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

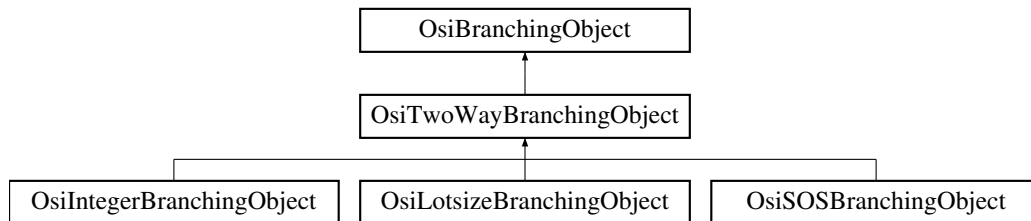
- </home/ted/COIN/trunk/Osi/src/OsiSpx/OsiSpxSolverInterface.hpp>

8.37 OsiTwoWayBranchingObject Class Reference

This just adds two-wayness to a branching object.

```
#include <OsiBranchingObject.hpp>
```

Inheritance diagram for `OsiTwoWayBranchingObject`:



Public Member Functions

- [OsiTwoWayBranchingObject \(\)](#)
Default constructor.
- [OsiTwoWayBranchingObject \(OsiSolverInterface *solver, const OsiObject *originalObject, int way, double value\)](#)
Create a standard two-way branch object.
- [OsiTwoWayBranchingObject \(const OsiTwoWayBranchingObject &\)](#)
Copy constructor.
- [OsiTwoWayBranchingObject & operator= \(const OsiTwoWayBranchingObject &rhs\)](#)
Assignment operator.
- [virtual ~OsiTwoWayBranchingObject \(\)](#)
Destructor.
- [virtual double branch \(OsiSolverInterface *solver\)=0](#)
Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.
- [int firstBranch \(\) const](#)
- [int way \(\) const](#)
Way returns -1 on down +1 on up.

Protected Attributes

- `int firstBranch_`

Which way was first branch -1 = down, +1 = up.

8.37.1 Detailed Description

This just adds two-wayness to a branching object.

Definition at line 467 of file OsiBranchingObject.hpp.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 OsiTwoWayBranchingObject::OsiTwoWayBranchingObject ()

Default constructor.

8.37.2.2 OsiTwoWayBranchingObject::OsiTwoWayBranchingObject (OsiSolverInterface * *solver*, const OsiObject * *originalObject*, int *way*, double *value*)

Create a standard two-way branch object.

Specifies a simple two-way branch. Specify *way* = -1 to set the object state to perform the down arm first, *way* = 1 for the up arm.

8.37.2.3 OsiTwoWayBranchingObject::OsiTwoWayBranchingObject (const OsiTwoWayBranchingObject &)

Copy constructor.

8.37.2.4 virtual OsiTwoWayBranchingObject::~~OsiTwoWayBranchingObject () [virtual]

Destructor.

8.37.3 Member Function Documentation

8.37.3.1 OsiTwoWayBranchingObject& OsiTwoWayBranchingObject::operator= (const OsiTwoWayBranchingObject & *rhs*)

Assignment operator.

8.37.3.2 virtual double OsiTwoWayBranchingObject::branch (OsiSolverInterface * *solver*) [pure virtual]

Sets the bounds for the variable according to the current arm of the branch and advances the object state to the next arm.

state. Returns change in guessed objective on next branch

Implements [OsiBranchingObject](#).

Implemented in [OsiLotsizeBranchingObject](#), [OsiSOSBranchingObject](#), and [OsiIntegerBranchingObject](#).

8.37.3.3 int OsiTwoWayBranchingObject::firstBranch () const [inline]

Definition at line 500 of file OsiBranchingObject.hpp.

8.37.3.4 `int OsiTwoWayBranchingObject::way () const [inline]`

Way returns -1 on down +1 on up.

Definition at line 502 of file `OsiBranchingObject.hpp`.

8.37.4 Member Data Documentation

8.37.4.1 `int OsiTwoWayBranchingObject::firstBranch_ [protected]`

Which way was first branch -1 = down, +1 = up.

Definition at line 506 of file `OsiBranchingObject.hpp`.

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

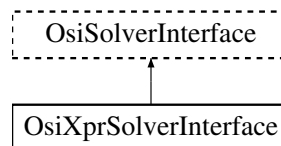
- [/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp](#)

8.38 OsiXprSolverInterface Class Reference

XPRESS-MP Solver Interface.

```
#include <OsiXprSolverInterface.hpp>
```

Inheritance diagram for `OsiXprSolverInterface`:



Public Member Functions

- virtual void `setObjSense` (double s)
Set objective function sense (1 for min (default), -1 for max,)
- virtual void `setColSolution` (const double *colsol)
Set the primal solution column values.
- virtual void `setRowPrice` (const double *rowprice)
Set dual solution vector.

Solve methods

- virtual void `initialSolve` ()
Solve initial LP relaxation.
- virtual void `resolve` ()
Resolve an LP relaxation after problem modification.
- virtual void `branchAndBound` ()
Invoke solver's built-in enumeration algorithm.

Parameter set/get methods

The set methods return true if the parameter was set to the given value, false otherwise.

There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the volume algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.

The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.

- bool [setIntParam](#) ([OsiIntParam](#) key, int value)
Set an integer parameter.
- bool [setDbfParam](#) ([OsiDbfParam](#) key, double value)
Set a double parameter.
- bool [setStrParam](#) ([OsiStrParam](#) key, const std::string &value)
Set a string parameter.
- bool [getIntParam](#) ([OsiIntParam](#) key, int &value) const
Get an integer parameter.
- bool [getDbfParam](#) ([OsiDbfParam](#) key, double &value) const
Get a double parameter.
- bool [getStrParam](#) ([OsiStrParam](#) key, std::string &value) const
Get a string parameter.
- void [setMipStart](#) (bool value)
- bool [getMipStart](#) () const

Methods returning info on how the solution process terminated

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

WarmStart related methods

- CoinWarmStart * [getEmptyWarmStart](#) () const
Get empty warm start object.
- virtual CoinWarmStart * [getWarmStart](#) () const
Get warmstarting information.
- virtual bool [setWarmStart](#) (const CoinWarmStart *warmstart)
Set warmstarting information.

Hotstart related methods (primarily used in strong branching).

The user can create a hotstart (a snapshot) of the optimization process then reoptimize over and over again always starting from there.

NOTE: between hotstarted optimizations only bound changes are allowed.

- virtual void [markHotStart](#) ()

- *Create a hotstart point of the optimization process.*
- virtual void `solveFromHotStart` ()
- *Optimize starting from the hotstart.*
- virtual void `unmarkHotStart` ()
- *Delete the snapshot.*

Methods related to querying the input data

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

Methods related to querying the solution

- virtual const double * `getColSolution` () const
- *Get pointer to array[getNumCols()] of primal solution vector.*
- virtual const double * `getRowPrice` () const
- *Get pointer to array[getNumRows()] of dual prices.*
- virtual const double * `getReducedCost` () const
- *Get a pointer to array[getNumCols()] of reduced costs.*
- virtual const double * `getRowActivity` () const
- *Get pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).*
- virtual double `getObjValue` () const
- *Get objective function value.*
- virtual int `getIterationCount` () const

- *Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver.*
virtual std::vector< double * > [getDualRays](#) (int maxNumRays, bool fullRay=false) const
- *Get as many dual rays as the solver can provide.*
virtual std::vector< double * > [getPrimalRays](#) (int maxNumRays) const
- *Get as many primal rays as the solver can provide.*

Changing bounds on variables and constraints

- virtual void [setObjCoeff](#) (int elementIndex, double elementValue)
Set an objective function coefficient.
- virtual void [setColLower](#) (int elementIndex, double elementValue)
Set a single column lower bound
Use -COIN_DBL_MAX for -infinity.
- virtual void [setColUpper](#) (int elementIndex, double elementValue)
Set a single column upper bound
Use COIN_DBL_MAX for infinity.
- virtual void [setColBounds](#) (int elementIndex, double lower, double upper)
Set a single column lower and upper bound
The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#)
- virtual void [setColSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a number of columns simultaneously
The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#) over and over again.
- virtual void [setRowLower](#) (int elementIndex, double elementValue)
Set a single row lower bound
Use -COIN_DBL_MAX for -infinity.
- virtual void [setRowUpper](#) (int elementIndex, double elementValue)
Set a single row upper bound
Use COIN_DBL_MAX for infinity.
- virtual void [setRowBounds](#) (int elementIndex, double lower, double upper)
Set a single row lower and upper bound
The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#)
- virtual void [setRowType](#) (int index, char sense, double rightHandSide, double range)
Set the type of a single row
- virtual void [setRowSetBounds](#) (const int *indexFirst, const int *indexLast, const double *boundList)
Set the bounds on a number of rows simultaneously
The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#) over and over again.
- virtual void [setRowSetTypes](#) (const int *indexFirst, const int *indexLast, const char *senseList, const double *rhsList, const double *rangeList)
Set the type of a number of rows simultaneously
The default implementation just invokes [setRowType\(\)](#) over and over again.

Integrality related changing methods

- virtual void [setContinuous](#) (int index)
Set the index-th variable to be a continuous variable.
- virtual void [setInteger](#) (int index)
Set the index-th variable to be an integer variable.
- virtual void [setContinuous](#) (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be continuous variables.
- virtual void [setInteger](#) (const int *indices, int len)
Set the variables listed in indices (which is of length len) to be integer variables.

Methods to expand a problem.

Note that if a column is added then by default it will correspond to a continuous variable.

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

Methods to input a problem

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

Message handling

- void **passInMessageHandler** (CoinMessageHandler *handler)

Pass in a message handler It is the client's responsibility to destroy a message handler installed by this routine; it will not be destroyed when the solver interface is destroyed.

Constructors and destructors

- [OsiXprSolverInterface](#) (int newrows=50, int newnz=100)
Default Constructor.
- virtual [OsiSolverInterface](#) * [clone](#) (bool copyData=true) const
Clone.
- [OsiXprSolverInterface](#) (const [OsiXprSolverInterface](#) &)
Copy constructor.
- [OsiXprSolverInterface](#) & [operator=](#) (const [OsiXprSolverInterface](#) &rhs)
Assignment operator.
- virtual [~OsiXprSolverInterface](#) ()
Destructor.

Static Public Member Functions

- static int [version](#) ()
Return XPRESS-MP Version number.

Protected Member Functions

Protected methods

- virtual void [applyRowCut](#) (const [OsiRowCut](#) &rc)
Apply a row cut. Return true if cut was applied.
- virtual void [applyColCut](#) (const [OsiColCut](#) &cc)
Apply a column cut (bound adjustment).

Private Member Functions

Private methods

- void [gutsOfCopy](#) (const [OsiXprSolverInterface](#) &source)
The real work of a copy constructor (used by copy and assignment)
- void [gutsOfConstructor](#) ()
The real work of a constructor (used by construct and assignment)
- void [gutsOfDestructor](#) ()
The real work of a destructor (used by copy and assignment)
- void [freeSolution](#) ()
Destroy cached copy of solution data (whenever it changes)
- void [freeCachedResults](#) ()
Destroy cached copies of problem and solution data (whenever they change)
- int [getNumIntVars](#) () const
Number of integer variables in the problem.

Methods to support for XPRESS-MP multiple matrix facility

- void [getVarTypes](#) () const
Build cached copy of variable types.
- void [activateMe](#) () const
Save the current problem in XPRESS (if necessary) and make this problem current (restore if necessary).
- bool [isDataLoaded](#) () const
Save and restore are necessary if there is data associated with this problem.

Private Attributes

- bool [domipstart](#)

Whether to pass a column solution to XPRESS before starting MIP solve (loadmipsol)

Data to support for XPRESS-MP multiple matrix facility

- [XPRESSprob prob_](#)
- [std::string xprProbname_](#)

XPRESS problem name (should be unique for each saved problem)

Cached copies of XPRESS-MP problem data

- [CoinPackedMatrix * matrixByRow_](#)
Pointer to row-wise copy of problem matrix coefficients.
- [CoinPackedMatrix * matrixByCol_](#)
- [double * colupper_](#)
Pointer to dense vector of structural variable upper bounds.
- [double * collower_](#)
Pointer to dense vector of structural variable lower bounds.
- [double * rowupper_](#)
Pointer to dense vector of slack variable upper bounds.
- [double * rowlower_](#)
Pointer to dense vector of slack variable lower bounds.
- [char * rowsense_](#)
Pointer to dense vector of row sense indicators.
- [double * rhs_](#)
Pointer to dense vector of row right-hand side values.
- [double * rowrange_](#)
Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)
- [double * objcoeffs_](#)
Pointer to dense vector of objective coefficients.
- [double objsense_](#)
Sense of objective (1 for min; -1 for max)
- [double * colsol_](#)
Pointer to dense vector of primal structural variable values.
- [double * rowsol_](#)
Pointer to dense vector of primal slack variable values.
- [double * rowact_](#)
Pointer to dense vector of primal slack variable values.
- [double * rowprice_](#)
Pointer to dense vector of dual row variable values.
- [double * colprice_](#)
Pointer to dense vector of dual column variable values.
- [int * ivarind_](#)
Pointer to list of indices of XPRESS "global" variables.
- [char * ivartype_](#)
Pointer to list of global variable types:
- [char * vartype_](#)
Pointer to dense vector of variable types (as above, or 'C' for continuous)
- bool [lastsolveasmip](#)
Indicates whether the last solve was for a MIP or an LP.

Static Private Attributes

Private static class data

- static const char * [logFileName_](#)
Name of the logfile.
- static FILE * [logFilePtr_](#)
The FILE to the logfile.*
- static unsigned int [numInstances_](#)
Number of live problem instances.
- static unsigned int [osiSerial_](#)
Counts calls to [incrementInstanceCounter\(\)](#)

Friends

- void [OsiXprSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiXprSolverInterface](#) class.

Static instance counter methods

- [XPRSProb](#) [getLpPtr](#) ()
Return a pointer to the XPRESS problem.
- static void [incrementInstanceCounter](#) ()
XPRESS has a context that must be created prior to all other XPRESS calls.
- static void [decrementInstanceCounter](#) ()
XPRESS has a context that should be deleted after XPRESS calls.
- static unsigned int [getNumInstances](#) ()
Return the number of instances of instantiated objects using XPRESS services.

Log File

- static int [iXprCallCount_](#)
- static FILE * [getLogFilePtr](#) ()
Get logfile FILE.*
- static void [setLogFileName](#) (const char *filename)
Set logfile name.

Additional Inherited Members

8.38.1 Detailed Description

XPRESS-MP Solver Interface.

Instantiation of [OsiSolverInterface](#) for XPRESS-MP

Definition at line 21 of file [OsiXprSolverInterface.hpp](#).

8.38.2 Constructor & Destructor Documentation

8.38.2.1 OsiXprSolverInterface::OsiXprSolverInterface (int newrows = 50, int newnz = 100)

Default Constructor.

8.38.2.2 `OsiXprSolverInterface::OsiXprSolverInterface (const OsiXprSolverInterface &)`

Copy constructor.

8.38.2.3 `virtual OsiXprSolverInterface::~~OsiXprSolverInterface () [virtual]`

Destructor.

8.38.3 Member Function Documentation

8.38.3.1 `virtual void OsiXprSolverInterface::initialSolve () [virtual]`

Solve initial LP relaxation.

Implements [OsiSolverInterface](#).

8.38.3.2 `virtual void OsiXprSolverInterface::resolve () [virtual]`

Resolve an LP relaxation after problem modification.

Implements [OsiSolverInterface](#).

8.38.3.3 `virtual void OsiXprSolverInterface::branchAndBound () [virtual]`

Invoke solver's built-in enumeration algorithm.

Implements [OsiSolverInterface](#).

8.38.3.4 `bool OsiXprSolverInterface::setIntParam (OsiIntParam key, int value) [virtual]`

Set an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.38.3.5 `bool OsiXprSolverInterface::setDbiParam (OsiDbiParam key, double value) [virtual]`

Set a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.38.3.6 `bool OsiXprSolverInterface::setStrParam (OsiStrParam key, const std::string & value) [virtual]`

Set a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.38.3.7 `bool OsiXprSolverInterface::getIntParam (OsiIntParam key, int & value) const [virtual]`

Get an integer parameter.

Reimplemented from [OsiSolverInterface](#).

8.38.3.8 `bool OsiXprSolverInterface::getDbiParam (OsiDbiParam key, double & value) const [virtual]`

Get a double parameter.

Reimplemented from [OsiSolverInterface](#).

8.38.3.9 `bool OsiXprSolverInterface::getStrParam (OsiStrParam key, std::string & value) const` [virtual]

Get a string parameter.

Reimplemented from [OsiSolverInterface](#).

8.38.3.10 `void OsiXprSolverInterface::setMipStart (bool value)` [inline]

Definition at line 64 of file `OsiXprSolverInterface.hpp`.

8.38.3.11 `bool OsiXprSolverInterface::getMipStart () const` [inline]

Definition at line 66 of file `OsiXprSolverInterface.hpp`.

8.38.3.12 `virtual bool OsiXprSolverInterface::isAbandoned () const` [virtual]

Are there a numerical difficulties?

Implements [OsiSolverInterface](#).

8.38.3.13 `virtual bool OsiXprSolverInterface::isProvenOptimal () const` [virtual]

Is optimality proven?

Implements [OsiSolverInterface](#).

8.38.3.14 `virtual bool OsiXprSolverInterface::isProvenPrimalInfeasible () const` [virtual]

Is primal infeasibility proven?

Implements [OsiSolverInterface](#).

8.38.3.15 `virtual bool OsiXprSolverInterface::isProvenDualInfeasible () const` [virtual]

Is dual infeasibility proven?

Implements [OsiSolverInterface](#).

8.38.3.16 `virtual bool OsiXprSolverInterface::isPrimalObjectiveLimitReached () const` [virtual]

Is the given primal objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.38.3.17 `virtual bool OsiXprSolverInterface::isDualObjectiveLimitReached () const` [virtual]

Is the given dual objective limit reached?

Reimplemented from [OsiSolverInterface](#).

8.38.3.18 `virtual bool OsiXprSolverInterface::isIterationLimitReached () const` [virtual]

Iteration limit reached?

Implements [OsiSolverInterface](#).

8.38.3.19 `CoinWarmStart* OsiXprSolverInterface::getEmptyWarmStart () const` [virtual]

Get empty warm start object.

Implements [OsiSolverInterface](#).

8.38.3.20 `virtual CoinWarmStart* OsiXprSolverInterface::getWarmStart () const [virtual]`

Get warmstarting information.

Implements [OsiSolverInterface](#).

8.38.3.21 `virtual bool OsiXprSolverInterface::setWarmStart (const CoinWarmStart * warmstart) [virtual]`

Set warmstarting information.

Return true/false depending on whether the warmstart information was accepted or not.

Implements [OsiSolverInterface](#).

8.38.3.22 `virtual void OsiXprSolverInterface::markHotStart () [virtual]`

Create a hotstart point of the optimization process.

Reimplemented from [OsiSolverInterface](#).

8.38.3.23 `virtual void OsiXprSolverInterface::solveFromHotStart () [virtual]`

Optimize starting from the hotstart.

Reimplemented from [OsiSolverInterface](#).

8.38.3.24 `virtual void OsiXprSolverInterface::unmarkHotStart () [virtual]`

Delete the snapshot.

Reimplemented from [OsiSolverInterface](#).

8.38.3.25 `virtual int OsiXprSolverInterface::getNumCols () const [virtual]`

Get number of columns.

Implements [OsiSolverInterface](#).

8.38.3.26 `virtual int OsiXprSolverInterface::getNumRows () const [virtual]`

Get number of rows.

Implements [OsiSolverInterface](#).

8.38.3.27 `virtual int OsiXprSolverInterface::getNumElements () const [virtual]`

Get number of nonzero elements.

Implements [OsiSolverInterface](#).

8.38.3.28 `virtual const double* OsiXprSolverInterface::getColLower () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column lower bounds.

Implements [OsiSolverInterface](#).

8.38.3.29 `virtual const double* OsiXprSolverInterface::getColUpper () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of column upper bounds.

Implements [OsiSolverInterface](#).

8.38.3.30 `virtual const char* OsiXprSolverInterface::getRowSense () const [virtual]`

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

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

Implements [OsiSolverInterface](#).

8.38.3.31 `virtual const double* OsiXprSolverInterface::getRightHandSide () const [virtual]`

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

- if `rowsense()[i] == 'L'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'G'` then `rhs()[i] == rowlower()[i]`
- if `rowsense()[i] == 'R'` then `rhs()[i] == rowupper()[i]`
- if `rowsense()[i] == 'N'` then `rhs()[i] == 0.0`

Implements [OsiSolverInterface](#).

8.38.3.32 `virtual const double* OsiXprSolverInterface::getRowRange () const [virtual]`

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

- if `rowsense()[i] == 'R'` then `rowrange()[i] == rowupper()[i] - rowlower()[i]`
- if `rowsense()[i] != 'R'` then `rowrange()[i]` is 0.0

Implements [OsiSolverInterface](#).

8.38.3.33 `virtual const double* OsiXprSolverInterface::getRowLower () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.38.3.34 `virtual const double* OsiXprSolverInterface::getRowUpper () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.38.3.35 `virtual const double* OsiXprSolverInterface::getObjCoefficients () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of objective function coefficients.

Implements [OsiSolverInterface](#).

8.38.3.36 `virtual double OsiXprSolverInterface::getObjSense () const [virtual]`

Get objective function sense (1 for min (default), -1 for max)

Implements [OsiSolverInterface](#).

8.38.3.37 `virtual bool OsiXprSolverInterface::isContinuous (int colIndex) const [virtual]`

Return true if variable is continuous.

Implements [OsiSolverInterface](#).

8.38.3.38 `virtual const CoinPackedMatrix* OsiXprSolverInterface::getMatrixByRow () const [virtual]`

Get pointer to row-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.38.3.39 `virtual const CoinPackedMatrix* OsiXprSolverInterface::getMatrixByCol () const [virtual]`

Get pointer to column-wise copy of matrix.

Implements [OsiSolverInterface](#).

8.38.3.40 `virtual double OsiXprSolverInterface::getInfinity () const [virtual]`

Get solver's value for infinity.

Implements [OsiSolverInterface](#).

8.38.3.41 `virtual const double* OsiXprSolverInterface::getColSolution () const [virtual]`

Get pointer to array[[getNumCols\(\)](#)] of primal solution vector.

Implements [OsiSolverInterface](#).

8.38.3.42 `virtual const double* OsiXprSolverInterface::getRowPrice () const [virtual]`

Get pointer to array[[getNumRows\(\)](#)] of dual prices.

Implements [OsiSolverInterface](#).

8.38.3.43 `virtual const double* OsiXprSolverInterface::getReducedCost () const [virtual]`

Get a pointer to array[[getNumCols\(\)](#)] of reduced costs.

Implements [OsiSolverInterface](#).

8.38.3.44 `virtual const double* OsiXprSolverInterface::getRowActivity () const [virtual]`

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

Implements [OsiSolverInterface](#).

8.38.3.45 `virtual double OsiXprSolverInterface::getObjValue () const [virtual]`

Get objective function value.

Implements [OsiSolverInterface](#).

8.38.3.46 `virtual int OsiXprSolverInterface::getIterationCount () const [virtual]`

Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver.

Implements [OsiSolverInterface](#).

8.38.3.47 `virtual std::vector<double*> OsiXprSolverInterface::getDualRays (int maxNumRays, bool fullRay = false) const [virtual]`

Get as many dual rays as the solver can provide.

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

The first [getNumRows\(\)](#) ray components will always be associated with the row duals (as returned by [getRowPrice\(\)](#)). If `fullRay` is true, the final [getNumCols\(\)](#) entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.38.3.48 `virtual std::vector<double*> OsiXprSolverInterface::getPrimalRays (int maxNumRays) const [virtual]`

Get as many primal rays as the solver can provide.

(In case of proven dual infeasibility there should be at least one.)

NOTE for implementers of solver interfaces:

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

NOTE for users of solver interfaces:

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

Implements [OsiSolverInterface](#).

8.38.3.49 `virtual void OsiXprSolverInterface::setObjCoeff (int elementIndex, double elementValue) [virtual]`

Set an objective function coefficient.

Implements [OsiSolverInterface](#).

8.38.3.50 `virtual void OsiXprSolverInterface::setColLower (int elementIndex, double elementValue) [virtual]`

Set a single column lower bound

Use `-COIN_DBL_MAX` for -infinity.

Implements [OsiSolverInterface](#).

8.38.3.51 `virtual void OsiXprSolverInterface::setColUpper (int elementIndex, double elementValue) [virtual]`

Set a single column upper bound

Use `COIN_DBL_MAX` for infinity.

Implements [OsiSolverInterface](#).

8.38.3.52 `virtual void OsiXprSolverInterface::setColBounds (int elementIndex, double lower, double upper) [virtual]`

Set a single column lower and upper bound

The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#)

Reimplemented from [OsiSolverInterface](#).

8.38.3.53 `virtual void OsiXprSolverInterface::setColSetBounds (const int * indexFirst, const int * indexLast, const double * boundList) [virtual]`

Set the bounds on a number of columns simultaneously

The default implementation just invokes [setColLower\(\)](#) and [setColUpper\(\)](#) over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the variables whose <i>either</i> bound changes
<i>boundList</i>	the new lower/upper bound pairs for the variables

Reimplemented from [OsiSolverInterface](#).

8.38.3.54 `virtual void OsiXprSolverInterface::setRowLower (int elementIndex, double elementValue) [virtual]`

Set a single row lower bound

Use -COIN_DBL_MAX for -infinity.

Implements [OsiSolverInterface](#).

8.38.3.55 `virtual void OsiXprSolverInterface::setRowUpper (int elementIndex, double elementValue) [virtual]`

Set a single row upper bound

Use COIN_DBL_MAX for infinity.

Implements [OsiSolverInterface](#).

8.38.3.56 `virtual void OsiXprSolverInterface::setRowBounds (int elementIndex, double lower, double upper) [virtual]`

Set a single row lower and upper bound

The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#)

Reimplemented from [OsiSolverInterface](#).

8.38.3.57 `virtual void OsiXprSolverInterface::setRowType (int index, char sense, double rightHandSide, double range) [virtual]`

Set the type of a single row

Implements [OsiSolverInterface](#).

8.38.3.58 `virtual void OsiXprSolverInterface::setRowSetBounds (const int * indexFirst, const int * indexLast, const double * boundList) [virtual]`

Set the bounds on a number of rows simultaneously

The default implementation just invokes [setRowLower\(\)](#) and [setRowUpper\(\)](#) over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the constraints whose <i>either</i> bound changes
<i>boundList</i>	the new lower/upper bound pairs for the constraints

Reimplemented from [OsiSolverInterface](#).

8.38.3.59 `virtual void OsiXprSolverInterface::setRowSetTypes (const int * indexFirst, const int * indexLast, const char * senseList, const double * rhsList, const double * rangeList) [virtual]`

Set the type of a number of rows simultaneously

The default implementation just invokes [setRowType\(\)](#) over and over again.

Parameters

<i>indexFirst, index-Last</i>	pointers to the beginning and after the end of the array of the indices of the constraints whose <i>any</i> characteristics changes
<i>senseList</i>	the new senses
<i>rhsList</i>	the new right hand sides
<i>rangeList</i>	the new ranges

Reimplemented from [OsiSolverInterface](#).

8.38.3.60 `virtual void OsiXprSolverInterface::setContinuous (int index) [virtual]`

Set the index-th variable to be a continuous variable.

Implements [OsiSolverInterface](#).

8.38.3.61 `virtual void OsiXprSolverInterface::setInteger (int index) [virtual]`

Set the index-th variable to be an integer variable.

Implements [OsiSolverInterface](#).

8.38.3.62 `virtual void OsiXprSolverInterface::setContinuous (const int * indices, int len) [virtual]`

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

Reimplemented from [OsiSolverInterface](#).

8.38.3.63 `virtual void OsiXprSolverInterface::setInteger (const int * indices, int len) [virtual]`

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

Reimplemented from [OsiSolverInterface](#).

8.38.3.64 `virtual void OsiXprSolverInterface::setObjSense (double s) [virtual]`

Set objective function sense (1 for min (default), -1 for max,)

Implements [OsiSolverInterface](#).

8.38.3.65 `virtual void OsiXprSolverInterface::setColSolution (const double * colsol) [virtual]`

Set the primal solution column values.

`colsol[numcols()]` is an array of values of the problem column variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of `colsol()` until changed by another call to `setColsol()`

or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

Implements [OsiSolverInterface](#).

8.38.3.66 `virtual void OsiXprSolverInterface::setRowPrice (const double * rowprice) [virtual]`

Set dual solution vector.

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

Implements [OsiSolverInterface](#).

8.38.3.67 `virtual void OsiXprSolverInterface::addCol (const CoinPackedVectorBase & vec, const double collb, const double colub, const double obj) [virtual]`

Add a column (primal variable) to the problem.

Implements [OsiSolverInterface](#).

8.38.3.68 `virtual void OsiXprSolverInterface::addCols (const int numcols, const CoinPackedVectorBase *const * cols, const double * collb, const double * colub, const double * obj) [virtual]`

Add a set of columns (primal variables) to the problem.

The default implementation simply makes repeated calls to [addCol\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.38.3.69 `virtual void OsiXprSolverInterface::deleteCols (const int num, const int * colIndices) [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](#).

8.38.3.70 `virtual void OsiXprSolverInterface::addRow (const CoinPackedVectorBase & vec, const double rowlb, const double rowub) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.38.3.71 `virtual void OsiXprSolverInterface::addRow (const CoinPackedVectorBase & vec, const char rowsen, const double rowrhs, const double rowrng) [virtual]`

Add a row (constraint) to the problem.

Implements [OsiSolverInterface](#).

8.38.3.72 `virtual void OsiXprSolverInterface::addRows (const int numRows, const CoinPackedVectorBase *const * rows, const double * rowlb, const double * rowub) [virtual]`

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.38.3.73 `virtual void OsiXprSolverInterface::addRows (const int numrows, const CoinPackedVectorBase *const * rows, const char * rowSEN, const double * rowRHS, const double * rowrng)` [virtual]

Add a set of rows (constraints) to the problem.

The default implementation simply makes repeated calls to [addRow\(\)](#).

Reimplemented from [OsiSolverInterface](#).

8.38.3.74 `virtual void OsiXprSolverInterface::deleteRows (const int num, const int * rowIndices)` [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](#).

8.38.3.75 `virtual void OsiXprSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * colLB, const double * colUB, const double * obj, const double * rowLB, const double * rowUB)` [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](#).

8.38.3.76 `virtual void OsiXprSolverInterface::assignProblem (CoinPackedMatrix *& matrix, double *& colLB, double *& colUB, double *& obj, double *& rowLB, double *& rowUB)` [virtual]

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

For default values see the previous method.

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

Implements [OsiSolverInterface](#).

8.38.3.77 `virtual void OsiXprSolverInterface::loadProblem (const CoinPackedMatrix & matrix, const double * colLB, const double * colUB, const double * obj, const char * rowSEN, const double * rowRHS, const double * rowrng)` [virtual]

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

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

- *colUB*: all columns have upper bound infinity
- *colLB*: all columns have lower bound 0
- *obj*: all variables have 0 objective coefficient
- *rowSEN*: all rows are \geq

- `rowrhs`: all right hand sides are 0
- `rowrng`: 0 for the ranged rows

Implements [OsiSolverInterface](#).

8.38.3.78 `virtual void OsiXprSolverInterface::assignProblem (CoinPackedMatrix * & matrix, double * & collb, double * & colub, double * & obj, char * & rowSEN, double * & rowrhs, double * & rowrng) [virtual]`

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

For default values see the previous method.

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

Implements [OsiSolverInterface](#).

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

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

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

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

8.38.3.81 `virtual int OsiXprSolverInterface::readMps (const char * filename, const char * extension = "mps") [virtual]`

Read an mps file from the given filename.

Reimplemented from [OsiSolverInterface](#).

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

Write the problem into an mps file of the given filename.

If `objSense` is non zero then -1.0 forces the code to write a maximization objective and +1.0 to write a minimization one.

If 0.0 then solver can do what it wants

Implements [OsiSolverInterface](#).

8.38.3.83 `void OsiXprSolverInterface::passInMessageHandler (CoinMessageHandler * handler) [virtual]`

Pass in a message handler It is the client's responsibility to destroy a message handler installed by this routine; it will not be destroyed when the solver interface is destroyed.

Reimplemented from [OsiSolverInterface](#).

8.38.3.84 `static void OsiXprSolverInterface::incrementInstanceCounter () [static]`

XPRESS has a context that must be created prior to all other XPRESS calls.

This method:

- Increments by 1 the number of uses of the XPRESS environment.
- Creates the XPRESS context when the number of uses is changed to 1 from 0.

8.38.3.85 `static void OsiXprSolverInterface::decrementInstanceCounter () [static]`

XPRESS has a context that should be deleted after XPRESS calls.

This method:

- Decrements by 1 the number of uses of the XPRESS environment.
- Deletes the XPRESS context when the number of uses is change to 0 from 1.

8.38.3.86 `static unsigned int OsiXprSolverInterface::getNumInstances () [static]`

Return the number of instances of instantiated objects using XPRESS services.

8.38.3.87 `XPRESSprob OsiXprSolverInterface::getLpPtr () [inline]`

Return a pointer to the XPRESS problem.

Definition at line 606 of file OsiXprSolverInterface.hpp.

8.38.3.88 `static int OsiXprSolverInterface::version () [static]`

Return XPRESS-MP Version number.

8.38.3.89 `static FILE* OsiXprSolverInterface::getLogFilePtr () [static]`

Get logfile FILE *.

8.38.3.90 `static void OsiXprSolverInterface::setLogFileName (const char * filename) [static]`

Set logfile name.

The logfile is an attempt to capture the calls to Xpress functions for debugging.

8.38.3.91 `virtual OsiSolverInterface* OsiXprSolverInterface::clone (bool copyData = true) const [virtual]`

Clone.

Implements [OsiSolverInterface](#).

8.38.3.92 `OsiXprSolverInterface& OsiXprSolverInterface::operator= (const OsiXprSolverInterface & rhs)`

Assignment operator.

8.38.3.93 `virtual void OsiXprSolverInterface::applyRowCut (const OsiRowCut & rc) [protected], [virtual]`

Apply a row cut. Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.38.3.94 `virtual void OsiXprSolverInterface::applyColCut (const OsiColCut & cc) [protected], [virtual]`

Apply a column cut (bound adjustment).

Return true if cut was applied.

Implements [OsiSolverInterface](#).

8.38.3.95 `void OsiXprSolverInterface::gutsOfCopy (const OsiXprSolverInterface & source) [private]`

The real work of a copy constructor (used by copy and assignment)

8.38.3.96 `void OsiXprSolverInterface::gutsOfConstructor () [private]`

The real work of a constructor (used by construct and assignment)

8.38.3.97 `void OsiXprSolverInterface::gutsOfDestructor () [private]`

The real work of a destructor (used by copy and assignment)

8.38.3.98 `void OsiXprSolverInterface::freeSolution () [private]`

Destroy cached copy of solution data (whenever it changes)

8.38.3.99 `void OsiXprSolverInterface::freeCachedResults () [private]`

Destroy cached copies of problem and solution data (whenever they change)

8.38.3.100 `int OsiXprSolverInterface::getNumIntVars () const [private]`

Number of integer variables in the problem.

8.38.3.101 `void OsiXprSolverInterface::getVarTypes () const [private]`

Build cached copy of variable types.

8.38.3.102 `void OsiXprSolverInterface::activateMe () const [private]`

Save the current problem in XPRESS (if necessary) and make this problem current (restore if necessary).

8.38.3.103 `bool OsiXprSolverInterface::isDataLoaded () const [private]`

Save and restore are necessary if there is data associated with this problem.

Also, queries to a problem with no data should respond sensibly; XPRESS query results are undefined.

8.38.4 Friends And Related Function Documentation

8.38.4.1 `void OsiXprSolverInterfaceUnitTest (const std::string & mpsDir, const std::string & netlibDir) [friend]`

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

8.38.5 Member Data Documentation

8.38.5.1 `int OsiXprSolverInterface::iXprCallCount_ [static]`

Definition at line 614 of file `OsiXprSolverInterface.hpp`.

8.38.5.2 `const char* OsiXprSolverInterface::logFileName_ [static], [private]`

Name of the logfile.

Definition at line 661 of file `OsiXprSolverInterface.hpp`.

8.38.5.3 `FILE* OsiXprSolverInterface::logFilePtr_` `[static], [private]`

The FILE* to the logfile.

Definition at line 664 of file OsiXprSolverInterface.hpp.

8.38.5.4 `unsigned int OsiXprSolverInterface::numInstances_` `[static], [private]`

Number of live problem instances.

Definition at line 667 of file OsiXprSolverInterface.hpp.

8.38.5.5 `unsigned int OsiXprSolverInterface::osiSerial_` `[static], [private]`

Counts calls to [incrementInstanceCounter\(\)](#)

Definition at line 670 of file OsiXprSolverInterface.hpp.

8.38.5.6 `XPRSProb OsiXprSolverInterface::prob_` `[mutable], [private]`

Definition at line 719 of file OsiXprSolverInterface.hpp.

8.38.5.7 `std::string OsiXprSolverInterface::xprProbname_` `[mutable], [private]`

XPRESS problem name (should be unique for each saved problem)

Definition at line 722 of file OsiXprSolverInterface.hpp.

8.38.5.8 `CoinPackedMatrix* OsiXprSolverInterface::matrixByRow_` `[mutable], [private]`

Pointer to row-wise copy of problem matrix coefficients.

Note that XPRESS keeps the objective row in the problem matrix, so row indices and counts are adjusted accordingly.

Definition at line 732 of file OsiXprSolverInterface.hpp.

8.38.5.9 `CoinPackedMatrix* OsiXprSolverInterface::matrixByCol_` `[mutable], [private]`

Definition at line 733 of file OsiXprSolverInterface.hpp.

8.38.5.10 `double* OsiXprSolverInterface::colupper_` `[mutable], [private]`

Pointer to dense vector of structural variable upper bounds.

Definition at line 736 of file OsiXprSolverInterface.hpp.

8.38.5.11 `double* OsiXprSolverInterface::collower_` `[mutable], [private]`

Pointer to dense vector of structural variable lower bounds.

Definition at line 739 of file OsiXprSolverInterface.hpp.

8.38.5.12 `double* OsiXprSolverInterface::rowupper_` `[mutable], [private]`

Pointer to dense vector of slack variable upper bounds.

Definition at line 742 of file OsiXprSolverInterface.hpp.

8.38.5.13 `double* OsiXprSolverInterface::rowlower_` `[mutable], [private]`

Pointer to dense vector of slack variable lower bounds.

Definition at line 745 of file OsiXprSolverInterface.hpp.

8.38.5.14 `char* OsiXprSolverInterface::rowsense_ [mutable],[private]`

Pointer to dense vector of row sense indicators.

Definition at line 748 of file OsiXprSolverInterface.hpp.

8.38.5.15 `double* OsiXprSolverInterface::rhs_ [mutable],[private]`

Pointer to dense vector of row right-hand side values.

Definition at line 751 of file OsiXprSolverInterface.hpp.

8.38.5.16 `double* OsiXprSolverInterface::rowrange_ [mutable],[private]`

Pointer to dense vector of slack upper bounds for range constraints (undefined for non-range rows)

Definition at line 756 of file OsiXprSolverInterface.hpp.

8.38.5.17 `double* OsiXprSolverInterface::objcoeffs_ [mutable],[private]`

Pointer to dense vector of objective coefficients.

Definition at line 759 of file OsiXprSolverInterface.hpp.

8.38.5.18 `double OsiXprSolverInterface::objsense_ [mutable],[private]`

Sense of objective (1 for min; -1 for max)

Definition at line 762 of file OsiXprSolverInterface.hpp.

8.38.5.19 `double* OsiXprSolverInterface::colsol_ [mutable],[private]`

Pointer to dense vector of primal structural variable values.

Definition at line 765 of file OsiXprSolverInterface.hpp.

8.38.5.20 `double* OsiXprSolverInterface::rowsol_ [mutable],[private]`

Pointer to dense vector of primal slack variable values.

Definition at line 768 of file OsiXprSolverInterface.hpp.

8.38.5.21 `double* OsiXprSolverInterface::rowact_ [mutable],[private]`

Pointer to dense vector of primal slack variable values.

Definition at line 771 of file OsiXprSolverInterface.hpp.

8.38.5.22 `double* OsiXprSolverInterface::rowprice_ [mutable],[private]`

Pointer to dense vector of dual row variable values.

Definition at line 774 of file OsiXprSolverInterface.hpp.

8.38.5.23 `double* OsiXprSolverInterface::colprice_ [mutable],[private]`

Pointer to dense vector of dual column variable values.

Definition at line 777 of file OsiXprSolverInterface.hpp.

8.38.5.24 `int* OsiXprSolverInterface::ivarind_` [mutable], [private]

Pointer to list of indices of XPRESS "global" variables.

Definition at line 780 of file OsiXprSolverInterface.hpp.

8.38.5.25 `char* OsiXprSolverInterface::ivartype_` [mutable], [private]

Pointer to list of global variable types:

- 'B': binary variable
- 'I': general integer variable (but might have 0-1 bounds)
- 'P': partial integer variable (not currently supported)
- 'S': sem-continuous variable (not currently supported)

Definition at line 790 of file OsiXprSolverInterface.hpp.

8.38.5.26 `char* OsiXprSolverInterface::vartype_` [mutable], [private]

Pointer to dense vector of variable types (as above, or 'C' for continuous)

Definition at line 795 of file OsiXprSolverInterface.hpp.

8.38.5.27 `bool OsiXprSolverInterface::lastsolwasmip` [mutable], [private]

Indicates whether the last solve was for a MIP or an LP.

Definition at line 798 of file OsiXprSolverInterface.hpp.

8.38.5.28 `bool OsiXprSolverInterface::domipstart` [private]

Whether to pass a column solution to XPRESS before starting MIP solve (loadmipsol)

Definition at line 803 of file OsiXprSolverInterface.hpp.

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

- [/home/ted/COIN/trunk/Osi/src/OsiXpr/OsiXprSolverInterface.hpp](#)

8.39 OsiUnitTest::TestOutcome Class Reference

A single test outcome record.

```
#include <OsiUnitTests.hpp>
```

Public Types

- enum [SeverityLevel](#) {
[NOTE](#) = 0, [PASSED](#) = 1, [WARNING](#) = 2, [ERROR](#) = 3,
[LAST](#) = 4 }

Test result.

Public Member Functions

- [TestOutcome](#) (const std::string &comp, const std::string &tst, const char *cond, [SeverityLevel](#) sev, const char *file, int line, bool exp=false)
Standard constructor.
- void [print](#) () const
Print the test outcome.

Public Attributes

- std::string [component](#)
Name of component under test.
- std::string [testname](#)
Name of test.
- std::string [testcond](#)
Condition being tested.
- [SeverityLevel](#) [severity](#)
Test result.
- bool [expected](#)
Set to true if problem is expected.
- std::string [filename](#)
Name of code file where test executed.
- int [linenumber](#)
Line number in code file where test executed.

Static Public Attributes

- static std::string [SeverityLevelName](#) [LAST]
Print strings for SeverityLevel.

8.39.1 Detailed Description

A single test outcome record.

Definition at line 166 of file OsiUnitTests.hpp.

8.39.2 Member Enumeration Documentation

8.39.2.1 enum OsiUnitTest::TestOutcome::SeverityLevel

Test result.

Enumerator

NOTE
PASSED
WARNING
ERROR
LAST

Definition at line 169 of file OsiUnitTests.hpp.

8.39.3 Constructor & Destructor Documentation

8.39.3.1 `OsiUnitTest::TestOutcome::TestOutcome (const std::string & comp, const std::string & tst, const char * cond, SeverityLevel sev, const char * file, int line, bool exp = false) [inline]`

Standard constructor.

Definition at line 193 of file OsiUnitTests.hpp.

8.39.4 Member Function Documentation

8.39.4.1 `void OsiUnitTest::TestOutcome::print () const`

Print the test outcome.

8.39.5 Member Data Documentation

8.39.5.1 `std::string OsiUnitTest::TestOutcome::SeverityLevelName[LAST] [static]`

Print strings for SeverityLevel.

Definition at line 177 of file OsiUnitTests.hpp.

8.39.5.2 `std::string OsiUnitTest::TestOutcome::component`

Name of component under test.

Definition at line 179 of file OsiUnitTests.hpp.

8.39.5.3 `std::string OsiUnitTest::TestOutcome::testname`

Name of test.

Definition at line 181 of file OsiUnitTests.hpp.

8.39.5.4 `std::string OsiUnitTest::TestOutcome::testcond`

Condition being tested.

Definition at line 183 of file OsiUnitTests.hpp.

8.39.5.5 `SeverityLevel OsiUnitTest::TestOutcome::severity`

Test result.

Definition at line 185 of file OsiUnitTests.hpp.

8.39.5.6 `bool OsiUnitTest::TestOutcome::expected`

Set to true if problem is expected.

Definition at line 187 of file OsiUnitTests.hpp.

8.39.5.7 `std::string OsiUnitTest::TestOutcome::filename`

Name of code file where test executed.

Definition at line 189 of file OsiUnitTests.hpp.

8.39.5.8 int OsiUnitTest::TestOutcome::linenumber

Line number in code file where test executed.

Definition at line 191 of file OsiUnitTests.hpp.

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

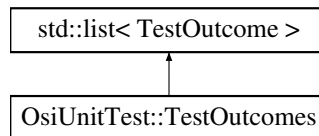
- </home/ted/COIN/trunk/Osi/src/OsiCommonTest/OsiUnitTests.hpp>

8.40 OsiUnitTest::TestOutcomes Class Reference

Utility class to maintain a list of test outcomes.

```
#include <OsiUnitTests.hpp>
```

Inheritance diagram for OsiUnitTest::TestOutcomes:



Public Member Functions

- void [add](#) (std::string comp, std::string tst, const char *cond, [TestOutcome::SeverityLevel](#) sev, const char *file, int line, bool exp=false)
Add an outcome to the list.
- void [add](#) (const [OsiSolverInterface](#) &si, std::string tst, const char *cond, [TestOutcome::SeverityLevel](#) sev, const char *file, int line, bool exp=false)
Add an outcome to the list.
- void [print](#) () const
Print the list of outcomes.
- void [getCountBySeverity](#) ([TestOutcome::SeverityLevel](#) sev, int &total, int &expected) const
Count total and expected outcomes at given severity level.

Additional Inherited Members

8.40.1 Detailed Description

Utility class to maintain a list of test outcomes.

Definition at line 204 of file OsiUnitTests.hpp.

8.40.2 Member Function Documentation

8.40.2.1 void OsiUnitTest::TestOutcomes::add (std::string *comp*, std::string *tst*, const char * *cond*, [TestOutcome::SeverityLevel](#) *sev*, const char * *file*, int *line*, bool *exp* = false) [inline]

Add an outcome to the list.

Definition at line 207 of file OsiUnitTests.hpp.

8.40.2.2 `void OsiUnitTest::TestOutcomes::add (const OsiSolverInterface & si, std::string tst, const char * cond,
TestOutcome::SeverityLevel sev, const char * file, int line, bool exp = false)`

Add an outcome to the list.

Get the component name from the solver interface.

8.40.2.3 `void OsiUnitTest::TestOutcomes::print () const`

Print the list of outcomes.

8.40.2.4 `void OsiUnitTest::TestOutcomes::getCountBySeverity (TestOutcome::SeverityLevel sev, int & total, int & expected)
const`

Count total and expected outcomes at given severity level.

Given a severity level, walk the list of outcomes and count the total number of outcomes at this severity level and the number expected.

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

- </home/ted/COIN/trunk/Osi/src/OsiCommonTest/OsiUnitTests.hpp>

9 File Documentation

9.1 /home/ted/COIN/trunk/Osi/src/Osi/config_default.h File Reference

```
#include "configall_system.h"
#include "config_osi_default.h"
```

Macros

- `#define COIN_OSI_CHECKLEVEL 0`
- `#define COIN_OSI_VERBOSITY 0`
- `#define COIN_HAS_COINUTILS 1`

9.1.1 Macro Definition Documentation

9.1.1.1 `#define COIN_OSI_CHECKLEVEL 0`

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

9.1.1.2 `#define COIN_OSI_VERBOSITY 0`

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

9.1.1.3 `#define COIN_HAS_COINUTILS 1`

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

9.2 /home/ted/COIN/trunk/Osi/src/Osi/config_osi_default.h File Reference

Macros

- `#define OSI_VERSION "trunk"`
- `#define OSI_VERSION_MAJOR 9999`
- `#define OSI_VERSION_MINOR 9999`
- `#define OSI_VERSION_RELEASE 9999`

9.2.1 Macro Definition Documentation

9.2.1.1 `#define OSI_VERSION "trunk"`

Definition at line 8 of file `config_osi_default.h`.

9.2.1.2 `#define OSI_VERSION_MAJOR 9999`

Definition at line 11 of file `config_osi_default.h`.

9.2.1.3 `#define OSI_VERSION_MINOR 9999`

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

9.2.1.4 `#define OSI_VERSION_RELEASE 9999`

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

9.3 `/home/ted/COIN/trunk/Osi/src/Osi/OsiAuxInfo.hpp` File Reference

Classes

- class [OsiAuxInfo](#)
This class allows for a more structured use of algorithmic tweaking to an [OsiSolverInterface](#).
- class [OsiBabSolver](#)
This class allows for the use of more exotic solvers e.g.

9.4 `/home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.hpp` File Reference

```
#include <cassert>
#include <string>
#include <vector>
#include "CoinError.hpp"
#include "CoinTypes.hpp"
```

Classes

- class [OsiObject](#)
Abstract base class for 'objects'.
- class [OsiObject2](#)
Define a class to add a bit of complexity to [OsiObject](#) This assumes 2 way branching.
- class [OsiBranchingObject](#)

Abstract branching object base class.

- class [OsiBranchingInformation](#)
- class [OsiTwoWayBranchingObject](#)

This just adds two-wayness to a branching object.

- class [OsiSimpleInteger](#)

Define a single integer class.

- class [OsiIntegerBranchingObject](#)

Simple branching object for an integer variable.

- class [OsiSOS](#)

Define Special Ordered Sets of type 1 and 2.

- class [OsiSOSBranchingObject](#)

Branching object for Special ordered sets.

- class [OsiLotsize](#)

Lotsize class.

- class [OsiLotsizeBranchingObject](#)

Lotsize branching object.

9.5 /home/ted/COIN/trunk/Osi/src/Osi/OsiChooseVariable.hpp File Reference

```
#include <string>
#include <vector>
#include "CoinWarmStartBasis.hpp"
#include "OsiBranchingObject.hpp"
```

Classes

- class [OsiChooseVariable](#)

This class chooses a variable to branch on.

- class [OsiPseudoCosts](#)

This class is the placeholder for the pseudocosts used by [OsiChooseStrong](#).

- class [OsiChooseStrong](#)

This class chooses a variable to branch on.

- class [OsiHotInfo](#)

This class contains the result of strong branching on a variable When created it stores enough information for strong branching.

9.6 /home/ted/COIN/trunk/Osi/src/Osi/OsiColCut.hpp File Reference

```
#include <string>
#include "CoinPackedVector.hpp"
#include "OsiCollections.hpp"
#include "OsiCut.hpp"
```

Classes

- class [OsiColCut](#)

Column Cut Class.

9.7 /home/ted/COIN/trunk/Osi/src/Osi/OsiCollections.hpp File Reference

```
#include <vector>
```

Typedefs

Typedefs for Standard Template Library collections of Osi Objects.

- typedef std::vector< int > [OsiVectorInt](#)
Vector of int.
- typedef std::vector< double > [OsiVectorDouble](#)
Vector of double.
- typedef std::vector< [OsiColCut](#) * > [OsiVectorColCutPtr](#)
Vector of [OsiColCut](#) pointers.
- typedef std::vector< [OsiRowCut](#) * > [OsiVectorRowCutPtr](#)
Vector of [OsiRowCut](#) pointers.
- typedef std::vector< [OsiCut](#) * > [OsiVectorCutPtr](#)
Vector of [OsiCut](#) pointers.

9.7.1 Typedef Documentation

9.7.1.1 typedef std::vector<int> [OsiVectorInt](#)

Vector of int.

Definition at line 22 of file OsiCollections.hpp.

9.7.1.2 typedef std::vector<double> [OsiVectorDouble](#)

Vector of double.

Definition at line 24 of file OsiCollections.hpp.

9.7.1.3 typedef std::vector<[OsiColCut](#) *> [OsiVectorColCutPtr](#)

Vector of [OsiColCut](#) pointers.

Definition at line 26 of file OsiCollections.hpp.

9.7.1.4 typedef std::vector<[OsiRowCut](#) *> [OsiVectorRowCutPtr](#)

Vector of [OsiRowCut](#) pointers.

Definition at line 28 of file OsiCollections.hpp.

9.7.1.5 typedef std::vector<[OsiCut](#) *> [OsiVectorCutPtr](#)

Vector of [OsiCut](#) pointers.

Definition at line 30 of file OsiCollections.hpp.

9.8 /home/ted/COIN/trunk/Osi/src/Osi/OsiConfig.h File Reference

```
#include "config_osi_default.h"
```


9.9 /home/ted/COIN/trunk/Osi/src/Osi/OsiCut.hpp File Reference

```
#include "OsiCollections.hpp"
#include "OsiSolverInterface.hpp"
```

Classes

- class [OsiCut](#)

Macros

- `#define COIN_DEFAULT_VALUE_FOR_DUPLICATE true`
Base Class for cut.

9.9.1 Macro Definition Documentation

9.9.1.1 `#define COIN_DEFAULT_VALUE_FOR_DUPLICATE true`

Base Class for cut.

The Base cut class contains:

- a measure of the cut's effectiveness

Definition at line 30 of file OsiCut.hpp.

9.10 /home/ted/COIN/trunk/Osi/src/Osi/OsiCuts.hpp File Reference

```
#include "CoinPragma.hpp"
#include <cmath>
#include <cfloat>
#include "OsiCollections.hpp"
#include "OsiRowCut.hpp"
#include "OsiColCut.hpp"
#include "CoinFloatEqual.hpp"
```

Classes

- class [OsiCuts](#)
Collections of row cuts and column cuts.
- class [OsiCuts::iterator](#)
Iterator.
- class [OsiCuts::const_iterator](#)
Const Iterator.
- class [OsiCuts::OsiCutCompare](#)

9.11 /home/ted/COIN/trunk/Osi/src/Osi/OsiPresolve.hpp File Reference

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

Classes

- class [OsiPresolve](#)
OSI interface to COIN problem simplification capabilities.

9.12 /home/ted/COIN/trunk/Osi/src/Osi/OsiRowCut.hpp File Reference

```
#include "CoinPackedVector.hpp"  
#include "OsiCollections.hpp"  
#include "OsiCut.hpp"
```

Classes

- class [OsiRowCut](#)
Row Cut Class.
- class [OsiRowCut2](#)
Row Cut Class which refers back to row which created it.

Macros

- `#define` [OsiRowCut_inline](#)

9.12.1 Macro Definition Documentation

9.12.1.1 `#define` [OsiRowCut_inline](#)

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

9.13 /home/ted/COIN/trunk/Osi/src/Osi/OsiRowCutDebugger.hpp File Reference

Provides a facility to validate cut constraints to ensure that they do not cut off a given solution.

```
#include <string>  
#include "OsiCuts.hpp"  
#include "OsiSolverInterface.hpp"
```

Classes

- class [OsiRowCutDebugger](#)
Validate cuts against a known solution.

9.13.1 Detailed Description

Provides a facility to validate cut constraints to ensure that they do not cut off a given solution.

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

9.14 /home/ted/COIN/trunk/Osi/src/Osi/OsiSolverBranch.hpp File Reference

```
#include "CoinWarmStartBasis.hpp"
```

Classes

- class [OsiSolverBranch](#)
Solver Branch Class.
- class [OsiSolverResult](#)
Solver Result Class.

9.15 /home/ted/COIN/trunk/Osi/src/Osi/OsiSolverInterface.hpp File Reference

```
#include <cstdlib>
#include <string>
#include <vector>
#include "CoinTypes.hpp"
#include "CoinMessageHandler.hpp"
#include "CoinPackedVectorBase.hpp"
#include "CoinPackedMatrix.hpp"
#include "CoinWarmStart.hpp"
#include "CoinFinite.hpp"
#include "CoinError.hpp"
#include "OsiCollections.hpp"
#include "OsiSolverParameters.hpp"
```

Classes

- class [OsiSolverInterface](#)
Abstract Base Class for describing an interface to a solver.
- class [OsiSolverInterface::ApplyCutsReturnCode](#)
Internal class for obtaining status from the applyCuts method.

9.16 /home/ted/COIN/trunk/Osi/src/Osi/OsiSolverParameters.hpp File Reference

Enumerations

- enum [OsiIntParam](#) { [OsiMaxNumIteration](#) = 0, [OsiMaxNumIterationHotStart](#), [OsiNameDiscipline](#), [OsiLastIntParam](#) }
- enum [OsiDbiParam](#) { [OsiDualObjectiveLimit](#) = 0, [OsiPrimalObjectiveLimit](#), [OsiDualTolerance](#), [OsiPrimalTolerance](#), [OsiObjOffset](#), [OsiLastDbiParam](#) }

- enum `OsiStrParam` { `OsiProbName` = 0, `OsiSolverName`, `OsiLastStrParam` }
- enum `OsiHintParam` {
`OsiDoPresolveInInitial` = 0, `OsiDoDualInInitial`, `OsiDoPresolveInResolve`, `OsiDoDualInResolve`,
`OsiDoScale`, `OsiDoCrash`, `OsiDoReducePrint`, `OsiDoInBranchAndCut`,
`OsiLastHintParam` }
- enum `OsiHintStrength` { `OsiHintIgnore` = 0, `OsiHintTry`, `OsiHintDo`, `OsiForceDo` }

9.16.1 Enumeration Type Documentation

9.16.1.1 enum `OsiIntParam`

Enumerator

OsiMaxNumIteration Iteration limit for initial solve and resolve. The maximum number of iterations (whatever that means for the given solver) the solver can execute in the `OsiSolverInterface::initialSolve()` and `OsiSolverInterface::resolve()` methods before terminating.

OsiMaxNumIterationHotStart Iteration limit for hot start. The maximum number of iterations (whatever that means for the given solver) the solver can execute in the `OsiSolverInterface::solveFromHotStart()` method before terminating.

OsiNameDiscipline Handling of row and column names. The name discipline specifies how the solver will handle row and column names:

- 0: Auto names: Names cannot be set by the client. Names of the form `Rnnnnnnn` or `Cnnnnnnn` are generated on demand when a name for a specific row or column is requested; `nnnnnnn` is derived from the row or column index. Requests for a vector of names return a vector with zero entries.
- 1: Lazy names: Names supplied by the client are retained. Names of the form `Rnnnnnnn` or `Cnnnnnnn` are generated on demand if no name has been supplied by the client. Requests for a vector of names return a vector sized to the largest index of a name supplied by the client; some entries in the vector may be null strings.
- 2: Full names: Names supplied by the client are retained. Names of the form `Rnnnnnnn` or `Cnnnnnnn` are generated on demand if no name has been supplied by the client. Requests for a vector of names return a vector sized to match the constraint system, and all entries will contain either the name specified by the client or a generated name.

OsiLastIntParam End marker. Used by [OsiSolverInterface](#) to allocate a fixed-sized array to store integer parameters.

Definition at line 8 of file `OsiSolverParameters.hpp`.

9.16.1.2 enum `OsiDbIParam`

Enumerator

OsiDualObjectiveLimit Dual objective limit. This is to be used as a termination criteria in algorithms where the dual objective changes monotonically (e.g., dual simplex, volume algorithm).

OsiPrimalObjectiveLimit Primal objective limit. This is to be used as a termination criteria in algorithms where the primal objective changes monotonically (e.g., primal simplex)

OsiDualTolerance Dual feasibility tolerance. The maximum amount a dual constraint can be violated and still be considered feasible.

OsiPrimalTolerance Primal feasibility tolerance. The maximum amount a primal constraint can be violated and still be considered feasible.

OsiObjOffset The value of any constant term in the objective function.

OsiLastDbIParam End marker. Used by [OsiSolverInterface](#) to allocate a fixed-sized array to store double parameters.

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

9.16.1.3 enum OsiStrParam

Enumerator

OsiProbName The name of the loaded problem. This is the string specified on the Name card of an mps file.

OsiSolverName The name of the solver. This parameter is read-only.

OsiLastStrParam End marker. Used by [OsiSolverInterface](#) to allocate a fixed-sized array to store string parameters.

Definition at line 88 of file OsiSolverParameters.hpp.

9.16.1.4 enum OsiHintParam

Enumerator

OsiDoPresolveInInitial Whether to do a presolve in initialSolve.

OsiDoDualInInitial Whether to use a dual algorithm in initialSolve. The reverse is to use a primal algorithm

OsiDoPresolveInResolve Whether to do a presolve in resolve.

OsiDoDualInResolve Whether to use a dual algorithm in resolve. The reverse is to use a primal algorithm

OsiDoScale Whether to scale problem.

OsiDoCrash Whether to create a non-slack basis (only in initialSolve)

OsiDoReducePrint Whether to reduce amount of printout, e.g., for branch and cut.

OsiDoInBranchAndCut Whether we are in branch and cut - so can modify behavior.

OsiLastHintParam Just a marker, so that [OsiSolverInterface](#) can allocate a static sized array to store parameters.

Definition at line 107 of file OsiSolverParameters.hpp.

9.16.1.5 enum OsiHintStrength

Enumerator

OsiHintIgnore Ignore hint (default)

OsiHintTry This means it is only a hint.

OsiHintDo This means do hint if at all possible.

OsiForceDo And this means throw an exception if not possible.

Definition at line 131 of file OsiSolverParameters.hpp.

9.17 /home/ted/COIN/trunk/Osi/src/OsiCommonTest/OsiUnitTests.hpp File Reference

Utility methods for OSI unit tests.

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <list>
#include <map>
```

Classes

- class [OsiUnitTest::TestOutcome](#)
A single test outcome record.
- class [OsiUnitTest::TestOutcomes](#)
Utility class to maintain a list of test outcomes.

Namespaces

- [OsiUnitTest](#)
A namespace so we can define a few 'global' variables to use during tests.

Macros

- `#define OSIUNITTEST_QUOTEME_\(x\) #x`
Convert parameter to a string (stringification)
- `#define OSIUNITTEST_QUOTEME\(x\) OSIUNITTEST_QUOTEME_\(x\)`
Convert to string with one level of expansion of the parameter.
- `#define OSIUNITTEST_ADD_OUTCOME(component, testname, testcondition, severity, expected)`
Add a test outcome to the list held in [OsiUnitTest::outcomes](#).
- `#define OSIUNITTEST_ASSERT_SEVERITY_EXPECTED(condition, failurecode, component, testname, severity, expected)`
Test for a condition and record the result.
- `#define OSIUNITTEST_ASSERT_ERROR(condition, failurecode, component, testname)`
Perform a test with severity [OsiUnitTest::TestOutcome::ERROR](#), failure not expected.
- `#define OSIUNITTEST_ASSERT_WARNING(condition, failurecode, component, testname)`
Perform a test with severity [OsiUnitTest::TestOutcome::WARNING](#), failure not expected.
- `#define OSIUNITTEST_CATCH_SEVERITY_EXPECTED(trycode, catchcode, component, testname, severity, expected)`
Perform a test surrounded by a try/catch block.
- `#define OSIUNITTEST_CATCH_ERROR(trycode, catchcode, component, testname) OSIUNITTEST_CATCH_SEVERITY_EXPECTED(trycode, catchcode, component, testname, OsiUnitTest::TestOutcome::ERROR, false)`
Perform a try/catch test with severity [OsiUnitTest::TestOutcome::ERROR](#), failure not expected.
- `#define OSIUNITTEST_CATCH_WARNING(trycode, catchcode, component, testname) OSIUNITTEST_CATCH_SEVERITY_EXPECTED(trycode, catchcode, component, testname, OsiUnitTest::TestOutcome::WARNING, false)`
Perform a try/catch test with severity [OsiUnitTest::TestOutcome::WARNING](#), failure not expected.

Functions

- void [OsiSolverInterfaceMpsUnitTest](#) (const std::vector< [OsiSolverInterface](#) * > &vecEmptySiP, const std::string &mpsDir)
A function that tests that a lot of problems given in MPS files (mostly the NETLIB problems) solve properly with all the specified solvers.
- void [OsiSolverInterfaceCommonUnitTest](#) (const [OsiSolverInterface](#) *emptySi, const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiSolverInterface](#) class.
- void [OsiColCutUnitTest](#) (const [OsiSolverInterface](#) *baseSiP, const std::string &mpsDir)

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

- void [OsiRowCutUnitTest](#) (const [OsiSolverInterface](#) *baseSiP, const std::string &mpsDir)

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

- void [OsiRowCutDebuggerUnitTest](#) (const [OsiSolverInterface](#) *siP, const std::string &mpsDir)

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

- void [OsiCutsUnitTest](#) ()

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

- void [OsiUnitTest::failureMessage](#) (const std::string &solverName, const std::string &message)

Print an error message.

- void [OsiUnitTest::failureMessage](#) (const [OsiSolverInterface](#) &si, const std::string &message)

- void [OsiUnitTest::failureMessage](#) (const std::string &solverName, const std::string &testname, const std::string &testcond)

Print an error message, specifying the test name and condition.

- void [OsiUnitTest::failureMessage](#) (const [OsiSolverInterface](#) &si, const std::string &testname, const std::string &testcond)

- void [OsiUnitTest::testingMessage](#) (const char *const msg)

Print a message.

- bool [OsiUnitTest::equivalentVectors](#) (const [OsiSolverInterface](#) *si1, const [OsiSolverInterface](#) *si2, double tol, const double *v1, const double *v2, int size)

Utility method to check equality.

- bool [OsiUnitTest::compareProblems](#) ([OsiSolverInterface](#) *osi1, [OsiSolverInterface](#) *osi2)

Compare two problems for equality.

- bool [OsiUnitTest::isEquivalent](#) (const CoinPackedVectorBase &pv, int n, const double *fv)

Compare a packed vector with an expanded vector.

- bool [OsiUnitTest::processParameters](#) (int argc, const char **argv, std::map< std::string, std::string > &parms, const std::map< std::string, int > &ignorekeywords=std::map< std::string, int >())

Process command line parameters.

- template<typename Component >

bool [OsiUnitTest::OsiUnitTestAssertSeverityExpected](#) (bool condition, const char *condition_str, const char *filename, int line, const Component &component, const std::string &testname, TestOutcome::SeverityLevel severity, bool expected)

Variables

- unsigned int [OsiUnitTest::verbosity](#)

Verbosity level of unit tests.

- unsigned int [OsiUnitTest::haltonerror](#)

Behaviour on failing a test.

- TestOutcomes [OsiUnitTest::outcomes](#)

Test outcomes.

9.17.1 Detailed Description

Utility methods for OSI unit tests.

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

9.17.2 Macro Definition Documentation

9.17.2.1 `#define OSIUNITTEST_QUOTEME_(x) #x`

Convert parameter to a string (stringification)

Definition at line 231 of file OsiUnitTests.hpp.

9.17.2.2 `#define OSIUNITTEST_QUOTEME(x) OSIUNITTEST_QUOTEME_(x)`

Convert to string with one level of expansion of the parameter.

Definition at line 233 of file OsiUnitTests.hpp.

9.17.2.3 `#define OSIUNITTEST_ADD_OUTCOME(component, testname, testcondition, severity, expected)`

Value:

```
OsiUnitTest::outcomes.add(component, testname, testcondition, severity, \
    __FILE__, __LINE__, expected)
```

Add a test outcome to the list held in [OsiUnitTest::outcomes](#).

Definition at line 268 of file OsiUnitTests.hpp.

9.17.2.4 `#define OSIUNITTEST_ASSERT_SEVERITY_EXPECTED(condition, failurecode, component, testname, severity, expected)`

Value:

```
{ \
    if (!OsiUnitTestAssertSeverityExpected(condition, #condition, \
        __FILE__, __LINE__, component, testname, severity, expected)) { \
        failurecode; \
    } \
}
```

Test for a condition and record the result.

Test `condition` and record the result in [OsiUnitTest::outcomes](#). If it succeeds, record the result as [OsiUnitTest::TestOutcome::PASSED](#) and print a message for [OsiUnitTest::verbosity](#) ≥ 2 . If it fails, record the test as failed with `severity` and `expected` and react as specified by [OsiUnitTest::haltonerror](#).

`failurecode` is executed when failure is not fatal.

Definition at line 281 of file OsiUnitTests.hpp.

9.17.2.5 `#define OSIUNITTEST_ASSERT_ERROR(condition, failurecode, component, testname)`

Value:

```
OSIUNITTEST_ASSERT_SEVERITY_EXPECTED(condition, failurecode, component, \
    testname, \
    OsiUnitTest::TestOutcome::ERROR, false)
```

Perform a test with severity [OsiUnitTest::TestOutcome::ERROR](#), failure not expected.

Definition at line 293 of file OsiUnitTests.hpp.

9.17.2.6 `#define OSIUNITTEST_ASSERT_WARNING(condition, failurecode, component, testname)`

Value:


```
OSIUNITTEST_ASSERT_SEVERITY_EXPECTED(condition, failurecode, component,
    testname, \
    false)                                OsiUnitTest::TestOutcome::WARNING,
```

Perform a test with severity [OsiUnitTest::TestOutcome::WARNING](#), failure not expected.

Definition at line 300 of file OsiUnitTests.hpp.

```
9.17.2.7 #define OSIUNITTEST_CATCH_SEVERITY_EXPECTED( trycode, catchcode, component, testname, severity, expected
    )
```

Perform a test surrounded by a try/catch block.

`trycode` is executed in a try/catch block; if there's no throw the test is deemed to have succeeded and is recorded in [OsiUnitTest::outcomes](#) with status [OsiUnitTest::TestOutcome::PASSED](#). If the `trycode` throws a `CoinError`, the failure is recorded with status `severity` and `expected` and the value of [OsiUnitTest::haltonerror](#) is consulted. If the failure is not fatal, `catchcode` is executed. If any other error is thrown, the failure is recorded as for a `CoinError` and `catchcode` is executed (`haltonerror` is not consulted).

Definition at line 314 of file OsiUnitTests.hpp.

```
9.17.2.8 #define OSIUNITTEST_CATCH_ERROR( trycode, catchcode, component, testname ) OSIUNITTEST_CATCH-
    _SEVERITY_EXPECTED(trycode, catchcode, component, testname, OsiUnitTest::TestOutcome::ERROR,
    false)
```

Perform a try/catch test with severity [OsiUnitTest::TestOutcome::ERROR](#), failure not expected.

Definition at line 363 of file OsiUnitTests.hpp.

```
9.17.2.9 #define OSIUNITTEST_CATCH_WARNING( trycode, catchcode, component, testname ) OSIUNITTEST_CATCH_S-
    EVERITY_EXPECTED(trycode, catchcode, component, testname, OsiUnitTest::TestOutcome::WARNING,
    false)
```

Perform a try/catch test with severity [OsiUnitTest::TestOutcome::WARNING](#), failure not expected.

Definition at line 369 of file OsiUnitTests.hpp.

9.17.3 Function Documentation

```
9.17.3.1 void OsiSolverInterfaceMpsUnitTest ( const std::vector< OsiSolverInterface * > & vecEmptySiP, const std::string &
    mpsDir )
```

A function that tests that a lot of problems given in MPS files (mostly the NETLIB problems) solve properly with all the specified solvers.

The routine creates a vector of NetLib problems (problem name, objective, various other characteristics), and a vector of solvers to be tested.

Each solver is run on each problem. The run is deemed successful if the solver reports the correct problem size after loading and returns the correct objective value after optimization.

If multiple solvers are available, the results are compared pairwise against the results reported by adjacent solvers in the solver vector. Due to limitations of the volume solver, it must be the last solver in `vecEmptySiP`.

```
9.17.3.2 void OsiSolverInterfaceCommonUnitTest ( const OsiSolverInterface * emptySi, const std::string & mpsDir, const
    std::string & netlibDir )
```

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

Some time ago, if this method is compiled with optimization, the compilation took 10-15 minutes and the machine pages (has 256M core memory!)...

9.17.3.3 `void OsiColCutUnitTest (const OsiSolverInterface * baseSiP, const std::string & mpsDir)`

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

9.17.3.4 `void OsiRowCutUnitTest (const OsiSolverInterface * baseSiP, const std::string & mpsDir)`

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

9.17.3.5 `void OsiRowCutDebuggerUnitTest (const OsiSolverInterface * siP, const std::string & mpsDir)`

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

9.17.3.6 `void OsiCutsUnitTest ()`

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

9.18 /home/ted/COIN/trunk/Osi/src/OsiCpx/OsiCpxSolverInterface.hpp File Reference

```
#include "OsiSolverInterface.hpp"
#include "CoinWarmStartBasis.hpp"
#include "OsiColCut.hpp"
#include "OsiRowCut.hpp"
```

Classes

- class [OsiCpxSolverInterface](#)
CPLEX Solver Interface.

Typedefs

- typedef struct cpxlp * [CPXLPptr](#)
- typedef struct cpxenv * [CPXENVptr](#)

Functions

- void [OsiCpxSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiCpxSolverInterface](#) class.

9.18.1 Typedef Documentation

9.18.1.1 typedef struct cpxlp* [CPXLPptr](#)

Definition at line 21 of file [OsiCpxSolverInterface.hpp](#).

9.18.1.2 typedef struct cpxenv* [CPXENVptr](#)

Definition at line 22 of file [OsiCpxSolverInterface.hpp](#).

9.18.2 Function Documentation

9.18.2.1 void OsiCpxSolverInterfaceUnitTest (const std::string & mpsDir, const std::string & netlibDir)

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

9.19 /home/ted/COIN/trunk/Osi/src/OsiGlpk/OsiGlpkSolverInterface.hpp File Reference

```
#include <string>
#include "OsiSolverInterface.hpp"
#include "CoinPackedMatrix.hpp"
#include "CoinWarmStartBasis.hpp"
```

Classes

- struct [glp_prob](#)
- class [OsiGlpkSolverInterface](#)

Macros

- #define [LPX glp_prob](#)
GLPK Solver Interface.
- #define [GLP_PROB_DEFINED](#)

Functions

- void [OsiGlpkSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiGlpkSolverInterface](#) class.

9.19.1 Macro Definition Documentation

9.19.1.1 #define LPX glp_prob

GLPK Solver Interface.

Instantiation of [OsiGlpkSolverInterface](#) for GLPK

Definition at line 24 of file [OsiGlpkSolverInterface.hpp](#).

9.19.1.2 #define GLP_PROB_DEFINED

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

9.19.2 Function Documentation

9.19.2.1 void OsiGlpkSolverInterfaceUnitTest (const std::string & mpsDir, const std::string & netlibDir)

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

9.20 /home/ted/COIN/trunk/Osi/src/OsiGrb/OsiGrbSolverInterface.hpp File Reference

```
#include <string>
#include "OsiSolverInterface.hpp"
```

Classes

- class [OsiGrbSolverInterface](#)
Gurobi Solver Interface.

Typedefs

- typedef struct _GRBmodel [GRBmodel](#)
- typedef struct _GRBenv [GRBenv](#)

Functions

- void [OsiGrbSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiGrbSolverInterface](#) class.

9.20.1 Typedef Documentation

9.20.1.1 typedef struct _GRBmodel [GRBmodel](#)

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

9.20.1.2 typedef struct _GRBenv [GRBenv](#)

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

9.20.2 Function Documentation

9.20.2.1 void [OsiGrbSolverInterfaceUnitTest](#) (const std::string & *mpsDir*, const std::string & *netlibDir*)

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

9.21 /home/ted/COIN/trunk/Osi/src/OsiMsk/OsiMskSolverInterface.hpp File Reference

```
#include "OsiSolverInterface.hpp"
```

Classes

- class [OsiMskSolverInterface](#)

Typedefs

- typedef void * [MSKtask_t](#)
- typedef void * [MSKenv_t](#)

Functions

- void [OsiMskSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiMskSolverInterface](#) class.

9.21.1 Typedef Documentation

9.21.1.1 typedef void* [MSKtask_t](#)

Definition at line 16 of file [OsiMskSolverInterface.hpp](#).

9.21.1.2 typedef void* [MSKenv_t](#)

Definition at line 17 of file [OsiMskSolverInterface.hpp](#).

9.21.2 Function Documentation

9.21.2.1 void [OsiMskSolverInterfaceUnitTest](#) (const std::string & *mpsDir*, const std::string & *netlibDir*)

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

9.22 /home/ted/COIN/trunk/Osi/src/OsiSpx/OsiSpxSolverInterface.hpp File Reference

```
#include <string>
#include "OsiSolverInterface.hpp"
#include "CoinWarmStartBasis.hpp"
```

Classes

- class [OsiSpxSolverInterface](#)
SoPlex Solver Interface Instantiation of [OsiSpxSolverInterface](#) for SoPlex.

Namespaces

- [soplex](#)

Functions

- void [OsiSpxSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiSpxSolverInterface](#) class.

9.22.1 Function Documentation

9.22.1.1 void OsiSpxSolverInterfaceUnitTest (const std::string & mpsDir, const std::string & netlibDir)

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

9.23 /home/ted/COIN/trunk/Osi/src/OsiXpr/OsiXprSolverInterface.hpp File Reference

```
#include <string>
#include <cstdio>
#include "OsiSolverInterface.hpp"
```

Classes

- class [OsiXprSolverInterface](#)
XPRESS-MP Solver Interface.

Typedefs

- typedef struct xo_prob_struct * [XPRSprob](#)

Functions

- void [OsiXprSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)
A function that tests the methods in the [OsiXprSolverInterface](#) class.

9.23.1 Typedef Documentation

9.23.1.1 typedef struct xo_prob_struct* XPRSprob

Definition at line 13 of file OsiXprSolverInterface.hpp.

9.23.2 Function Documentation

9.23.2.1 void OsiXprSolverInterfaceUnitTest (const std::string & mpsDir, const std::string & netlibDir)

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

Index

Symbols

- ~ApplyCutsReturnCode
 - OsiSolverInterface::ApplyCutsReturnCode, [14](#)
- ~OsiAuxInfo
 - OsiAuxInfo, [22](#)
- ~OsiBabSolver
 - OsiBabSolver, [25](#)
- ~OsiBranchingInformation
 - OsiBranchingInformation, [31](#)
- ~OsiBranchingObject
 - OsiBranchingObject, [37](#)
- ~OsiChooseStrong
 - OsiChooseStrong, [41](#)
- ~OsiChooseVariable
 - OsiChooseVariable, [47](#)
- ~OsiColCut
 - OsiColCut, [54](#)
- ~OsiCpxSolverInterface
 - OsiCpxSolverInterface, [66](#)
- ~OsiCut
 - OsiCut, [84](#)
- ~OsiCuts
 - OsiCuts, [90](#)
- ~OsiGlpkSolverInterface
 - OsiGlpkSolverInterface, [103](#)
- ~OsiGrbSolverInterface
 - OsiGrbSolverInterface, [131](#)
- ~OsiHotInfo
 - OsiHotInfo, [152](#)
- ~OsiIntegerBranchingObject
 - OsiIntegerBranchingObject, [156](#)
- ~OsiLotsize
 - OsiLotsize, [159](#)
- ~OsiLotsizeBranchingObject
 - OsiLotsizeBranchingObject, [163](#)
- ~OsiMskSolverInterface
 - OsiMskSolverInterface, [172](#)
- ~OsiObject
 - OsiObject, [190](#)
- ~OsiObject2
 - OsiObject2, [195](#)
- ~OsiPresolve
 - OsiPresolve, [198](#)
- ~OsiPseudoCosts
 - OsiPseudoCosts, [203](#)
- ~OsiRowCut
 - OsiRowCut, [208](#)
- ~OsiRowCut2
 - OsiRowCut2, [212](#)
- ~OsiRowCutDebugger
 - OsiRowCutDebugger, [215](#)
- ~OsiSOS
 - OsiSOS, [273](#)
- ~OsiSOSBranchingObject
 - OsiSOSBranchingObject, [277](#)
- ~OsiSimpleInteger
 - OsiSimpleInteger, [219](#)
- ~OsiSolverBranch
 - OsiSolverBranch, [222](#)
- ~OsiSolverInterface
 - OsiSolverInterface, [237](#)
- ~OsiSolverResult
 - OsiSolverResult, [270](#)
- ~OsiSpxSolverInterface
 - OsiSpxSolverInterface, [284](#)
- ~OsiTwoWayBranchingObject
 - OsiTwoWayBranchingObject, [297](#)
- ~OsiXprSolverInterface
 - OsiXprSolverInterface, [306](#)
- ~const_iterator
 - OsiCuts::const_iterator, [18](#)
- ~iterator
 - OsiCuts::iterator, [20](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiAuxInfo.hpp, [326](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiBranchingObject.-
hpp, [326](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiChooseVariable.-
hpp, [327](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiColCut.hpp, [327](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiCollections.hpp,
[328](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiConfig.h, [328](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiCut.hpp, [329](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiCuts.hpp, [329](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiPresolve.hpp, [330](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiRowCut.hpp, [330](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiRowCutDebugger.-
hpp, [330](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiSolverBranch.hpp,
[331](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiSolverInterface.hpp,
[331](#)
- /home/ted/COIN/trunk/Osi/src/Osi/OsiSolverParameters.-
hpp, [331](#)
- /home/ted/COIN/trunk/Osi/src/Osi/config_default.h, [325](#)
- /home/ted/COIN/trunk/Osi/src/Osi/config_osi_default.h,
[325](#)
- /home/ted/COIN/trunk/Osi/src/Osi/CommonTest/OsiUnit-
Tests.hpp, [333](#)
- /home/ted/COIN/trunk/Osi/src/OsiCpx/OsiCpxSolver-
Interface.hpp, [338](#)

/home/ted/COIN/trunk/Osi/src/OsiGlpk/OsiGlpkSolver-
Interface.hpp, 339
/home/ted/COIN/trunk/Osi/src/OsiGrb/OsiGrbSolver-
Interface.hpp, 340
/home/ted/COIN/trunk/Osi/src/OsiMsk/OsiMskSolver-
Interface.hpp, 340
/home/ted/COIN/trunk/Osi/src/OsiSpx/OsiSpxSolver-
Interface.hpp, 341
/home/ted/COIN/trunk/Osi/src/OsiXpr/OsiXprSolver-
Interface.hpp, 342
_opaque_prob
glp_prob, 19

A

activate
OsiRowCutDebugger, 216
activateMe
OsiXprSolverInterface, 318
activateRowCutDebugger
OsiSolverInterface, 261
active
OsiRowCutDebugger, 216
add
OsiUnitTest::TestOutcomes, 324
addBranch
OsiSolverBranch, 223
addCol
OsiCpxSolverInterface, 74
OsiGlpkSolverInterface, 111
OsiGrbSolverInterface, 140
OsiMskSolverInterface, 180
OsiSolverInterface, 252
OsiSpxSolverInterface, 291
OsiXprSolverInterface, 314
addCols
OsiCpxSolverInterface, 74
OsiGlpkSolverInterface, 112
OsiGrbSolverInterface, 140
OsiMskSolverInterface, 181
OsiSolverInterface, 252
OsiXprSolverInterface, 314
addObjects
OsiSolverInterface, 261
addRow
OsiCpxSolverInterface, 74, 75
OsiGlpkSolverInterface, 112
OsiGrbSolverInterface, 140
OsiMskSolverInterface, 181
OsiSolverInterface, 253
OsiSpxSolverInterface, 291
OsiXprSolverInterface, 314
addRows
OsiCpxSolverInterface, 75
OsiGlpkSolverInterface, 112

OsiGrbSolverInterface, 141
OsiMskSolverInterface, 181
OsiSolverInterface, 253, 254
OsiXprSolverInterface, 314
alwaysTryCutsAtRootNode
OsiBabSolver, 26
appData_
OsiAuxInfo, 23
appDataEtc_
OsiSolverInterface, 267
applied_
OsiSolverInterface::ApplyCutsReturnCode, 16
applyBounds
OsiSolverBranch, 223
applyColCut
OsiCpxSolverInterface, 79
OsiGlpkSolverInterface, 115
OsiGrbSolverInterface, 145
OsiMskSolverInterface, 184
OsiSolverInterface, 265
OsiSpxSolverInterface, 293
OsiXprSolverInterface, 317
applyCuts
OsiGrbSolverInterface, 144
OsiSolverInterface, 255
ApplyCutsReturnCode
OsiSolverInterface::ApplyCutsReturnCode, 14
applyRowCut
OsiCpxSolverInterface, 79
OsiGlpkSolverInterface, 115
OsiGrbSolverInterface, 145
OsiMskSolverInterface, 184
OsiSolverInterface, 265
OsiSpxSolverInterface, 293
OsiXprSolverInterface, 317
applyRowCuts
OsiSolverInterface, 255
assignProblem
OsiCpxSolverInterface, 75, 76
OsiGlpkSolverInterface, 113
OsiGrbSolverInterface, 141, 142
OsiMskSolverInterface, 182
OsiSolverInterface, 256
OsiSpxSolverInterface, 292
OsiXprSolverInterface, 315, 316
auxcolind
OsiGrbSolverInterface, 150
auxcolindspace
OsiGrbSolverInterface, 149
auxcolspace
OsiGrbSolverInterface, 149

B

basis

- OsiSolverResult, 270
- basis_
 - OsiSolverResult, 271
- basisIsAvailable
 - OsiCpxSolverInterface, 78
 - OsiGrbSolverInterface, 144
 - OsiSolverInterface, 262
- bbWasLast_
 - OsiGlpkSolverInterface, 116
- beforeLower
 - OsiBabSolver, 27
- beforeLower_
 - OsiBabSolver, 28
- beforeUpper
 - OsiBabSolver, 28
- beforeUpper_
 - OsiBabSolver, 28
- begin
 - OsiCuts, 93
 - OsiCuts::const_iterator, 18
 - OsiCuts::iterator, 21
- bestObjectIndex
 - OsiChooseVariable, 48
- bestObjectIndex_
 - OsiChooseVariable, 51
- bestObjectiveValue
 - OsiBabSolver, 27
- bestObjectiveValue_
 - OsiBabSolver, 28
- bestSolution_
 - OsiBabSolver, 28
- bestWhichWay
 - OsiChooseVariable, 48
- bestWhichWay_
 - OsiChooseVariable, 51
- bound
 - OsiLotsize, 160
- bound_
 - OsiLotsize, 161
 - OsiSolverBranch, 224
- boundBranch
 - OsiBranchingObject, 38
 - OsiObject, 192
- bounds
 - OsiSolverBranch, 223
- branch
 - OsiBranchingObject, 37
 - OsiIntegerBranchingObject, 156
 - OsiLotsizeBranchingObject, 163
 - OsiSOSBranchingObject, 277
 - OsiTwoWayBranchingObject, 297
- branchAndBound
 - OsiCpxSolverInterface, 66
 - OsiGlpkSolverInterface, 103
- OsiGrbSolverInterface, 131
- OsiMskSolverInterface, 173
- OsiSolverInterface, 237
- OsiSpxSolverInterface, 284
- OsiXprSolverInterface, 306
- branchIndex
 - OsiBranchingObject, 38
- branchIndex_
 - OsiBranchingObject, 39
- branchingObject
 - OsiHotInfo, 153
- branchingObject_
 - OsiHotInfo, 154
- C
- COIN_HAS_COINUTILS
 - config_default.h, 325
- COIN_OSI_VERBOSITY
 - config_default.h, 325
- CPXENVptr
 - OsiCpxSolverInterface.hpp, 338
- CPXLPptr
 - OsiCpxSolverInterface.hpp, 338
- canDoHeuristics
 - OsiLotsize, 161
 - OsiObject, 192
 - OsiSOS, 275
- canDoSimplexInterface
 - OsiCpxSolverInterface, 77
 - OsiGrbSolverInterface, 143
 - OsiSolverInterface, 262
- canHandleShadowPrices
 - OsiLotsize, 161
 - OsiObject, 192
 - OsiSimpleInteger, 221
 - OsiSOS, 275
- canMoveToNearest
 - OsiObject, 192
- candidates
 - OsiChooseVariable, 49
- change
 - OsiHotInfo, 153
- changes_
 - OsiHotInfo, 154
- checkInfeasibility
 - OsiObject, 191
- checkIsCutoff
 - OsiBranchingObject, 38
- chooseVariable
 - OsiChooseStrong, 41
 - OsiChooseVariable, 47
- clearGoodSolution
 - OsiChooseVariable, 47
- clone

- OsiAuxInfo, 22
- OsiBabSolver, 25
- OsiBranchingInformation, 31
- OsiBranchingObject, 37
- OsiChooseStrong, 41
- OsiChooseVariable, 47
- OsiColCut, 56
- OsiCpxSolverInterface, 77
- OsiGlpkSolverInterface, 115
- OsiGrbSolverInterface, 143
- OsiHotInfo, 152
- OsiIntegerBranchingObject, 156
- OsiLotsize, 159
- OsiLotsizeBranchingObject, 163
- OsiMskSolverInterface, 184
- OsiObject, 191
- OsiRowCut, 210
- OsiRowCut2, 213
- OsiSimpleInteger, 219
- OsiSolverInterface, 265
- OsiSOS, 273
- OsiSOSBranchingObject, 277
- OsiSpxSolverInterface, 293
- OsiXprSolverInterface, 317
- colCut
 - OsiCuts, 92
- colCutIndex_
 - OsiCuts::const_iterator, 19
 - OsiCuts::iterator, 21
- colCutPtr
 - OsiCuts, 91, 92
- colCutPtrs_
 - OsiCuts, 94
- colNames_
 - OsiSolverInterface, 268
- collower_
 - OsiCpxSolverInterface, 80
 - OsiGlpkSolverInterface, 119
 - OsiGrbSolverInterface, 147
 - OsiMskSolverInterface, 186
 - OsiXprSolverInterface, 319
- colmap_G2O
 - OsiGrbSolverInterface, 149
- colmap_O2G
 - OsiGrbSolverInterface, 149
- colprice_
 - OsiXprSolverInterface, 320
- colsol_
 - OsiCpxSolverInterface, 81
 - OsiGlpkSolverInterface, 120
 - OsiGrbSolverInterface, 148
 - OsiMskSolverInterface, 187
 - OsiSpxSolverInterface, 295
 - OsiXprSolverInterface, 320
- colspace_
 - OsiGrbSolverInterface, 149
- coltype_
 - OsiCpxSolverInterface, 82
 - OsiGrbSolverInterface, 149
 - OsiMskSolverInterface, 188
- coltypesize_
 - OsiCpxSolverInterface, 82
 - OsiMskSolverInterface, 188
- columnLength_
 - OsiBranchingInformation, 34
- columnNumber
 - OsiBranchingObject, 38
 - OsiLotsize, 159
 - OsiObject, 192
 - OsiSimpleInteger, 220
- columnNumber_
 - OsiLotsize, 161
 - OsiSimpleInteger, 221
- columnStart_
 - OsiBranchingInformation, 33
- columnType
 - OsiSolverInterface, 244
- columnType_
 - OsiSolverInterface, 267
- colupper_
 - OsiCpxSolverInterface, 81
 - OsiGlpkSolverInterface, 119
 - OsiGrbSolverInterface, 148
 - OsiMskSolverInterface, 187
 - OsiXprSolverInterface, 319
- compareProblems
 - OsiUnitTest, 11
- component
 - OsiUnitTest::TestOutcome, 323
- config_osi_default.h
 - OSI_VERSION, 326
- consistent
 - OsiColCut, 55, 56
 - OsiCut, 86
 - OsiRowCut, 209
- const_iterator
 - OsiCuts::const_iterator, 18
- convertBoundToSense
 - OsiSolverInterface, 265
- convertSenseToBound
 - OsiSolverInterface, 266
- convertToNormalRow
 - OsiGrbSolverInterface, 146
- convertToRangedRow
 - OsiGrbSolverInterface, 146
- copyParameters
 - OsiSolverInterface, 239
- createBranch

- OsiLotsize, [159](#)
- OsiObject, [191](#)
- OsiSimpleInteger, [220](#)
- OsiSOS, [274](#)
- createResult
 - OsiSolverResult, [270](#)
- ctype_
 - OsiGlpkSolverInterface, [119](#)
- cutP_
 - OsiCuts::const_iterator, [19](#)
 - OsiCuts::iterator, [21](#)
- cutoff_
 - OsiBranchingInformation, [32](#)
- cuts_
 - OsiCuts::iterator, [21](#)
- cutsPtr_
 - OsiCuts::const_iterator, [18](#)
- D
- dblParam_
 - OsiSolverInterface, [267](#)
- decrementInstanceCounter
 - OsiGlpkSolverInterface, [115](#)
 - OsiGrbSolverInterface, [143](#)
 - OsiMskSolverInterface, [183](#)
 - OsiXprSolverInterface, [317](#)
- decrementNumberBranchesLeft
 - OsiBranchingObject, [37](#)
- defaultDual_
 - OsiBranchingInformation, [32](#)
- defaultHandler
 - OsiSolverInterface, [260](#)
- defaultHandler_
 - OsiSolverInterface, [267](#)
- definedSolution
 - OsiMskSolverInterface, [184](#)
- deleteBranchingInfo
 - OsiSolverInterface, [255](#)
- deleteColNames
 - OsiSolverInterface, [251](#)
- deleteCols
 - OsiCpxSolverInterface, [74](#)
 - OsiGlpkSolverInterface, [112](#)
 - OsiGrbSolverInterface, [140](#)
 - OsiMskSolverInterface, [181](#)
 - OsiSolverInterface, [252](#)
 - OsiSpxSolverInterface, [291](#)
 - OsiXprSolverInterface, [314](#)
- deleteObjects
 - OsiSolverInterface, [261](#)
- deleteRowNames
 - OsiSolverInterface, [250](#)
- deleteRows
 - OsiCpxSolverInterface, [75](#)
 - OsiGlpkSolverInterface, [112](#)
 - OsiGrbSolverInterface, [141](#)
 - OsiMskSolverInterface, [181](#)
 - OsiSolverInterface, [254](#)
 - OsiSpxSolverInterface, [291](#)
 - OsiXprSolverInterface, [315](#)
- depth_
 - OsiBranchingInformation, [34](#)
- dfltRowColName
 - OsiSolverInterface, [250](#)
- difference_type
 - OsiCuts::const_iterator, [17](#)
- direction_
 - OsiBranchingInformation, [32](#)
- disableFactorization
 - OsiCpxSolverInterface, [78](#)
 - OsiGrbSolverInterface, [144](#)
 - OsiSolverInterface, [262](#)
- disableSimplexInterface
 - OsiCpxSolverInterface, [77](#)
 - OsiGrbSolverInterface, [144](#)
 - OsiSolverInterface, [264](#)
- disableadvbasis
 - OsiCpxSolverInterface, [82](#)
- doStrongBranching
 - OsiChooseStrong, [42](#)
- domipstart
 - OsiCpxSolverInterface, [82](#)
 - OsiGrbSolverInterface, [149](#)
 - OsiXprSolverInterface, [321](#)
- down_
 - OsiIntegerBranchingObject, [156](#)
 - OsiLotsizeBranchingObject, [163](#)
- downChange
 - OsiChooseVariable, [48](#)
 - OsiHotInfo, [152](#)
- downChange_
 - OsiChooseVariable, [50](#)
- downEstimate
 - OsiLotsize, [160](#)
 - OsiObject, [193](#)
 - OsiSimpleInteger, [221](#)
 - OsiSOS, [274](#)
- downIterationCount
 - OsiHotInfo, [153](#)
- downNumber
 - OsiPseudoCosts, [204](#)
- downNumber_
 - OsiPseudoCosts, [205](#)
- downStatus
 - OsiHotInfo, [153](#)
- downTotalChange
 - OsiPseudoCosts, [204](#)
- downTotalChange_

- OsiPseudoCosts, 205
- dualObjectiveLimit_
 - OsiGlpkSolverInterface, 117
- dualPivotResult
 - OsiSolverInterface, 265
- dualSolution
 - OsiSolverResult, 270
- dualSolution_
 - OsiSolverResult, 271
- dualTolerance_
 - OsiGlpkSolverInterface, 117
- dumpCuts
 - OsiCuts, 93
- E
- ERROR
 - OsiUnitTest::TestOutcome, 322
- effectiveness
 - OsiCut, 84
- effectiveness_
 - OsiCut, 86
- elementByColumn_
 - OsiBranchingInformation, 33
- enableFactorization
 - OsiCpxSolverInterface, 77
 - OsiGrbSolverInterface, 144
 - OsiSolverInterface, 262
- enableSimplexInterface
 - OsiCpxSolverInterface, 77
 - OsiGrbSolverInterface, 144
 - OsiSolverInterface, 264
- end
 - OsiCuts, 93
 - OsiCuts::const_iterator, 18
 - OsiCuts::iterator, 21
- env_
 - OsiCpxSolverInterface, 80
 - OsiMskSolverInterface, 185
- equivalentVectors
 - OsiUnitTest, 11
- eraseAndDumpCuts
 - OsiCuts, 93
- eraseColCut
 - OsiCuts, 92
- eraseRowCut
 - OsiCuts, 92
- expected
 - OsiUnitTest::TestOutcome, 323
- extInconsistent_
 - OsiSolverInterface::ApplyCutsReturnCode, 16
- extraCharacteristics
 - OsiBabSolver, 27
- extraCharacteristics_
 - OsiBabSolver, 29

- F
- FREECACHED_COLUMN
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- FREECACHED_MATRIX
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- FREECACHED_RESULTS
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- FREECACHED_ROW
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- failureMessage
 - OsiUnitTest, 11
- feasibleOneWay
 - OsiSolverBranch, 223
- feasibleRegion
 - OsiLotsize, 159
 - OsiObject, 191
 - OsiSimpleInteger, 219
 - OsiSOS, 273
- feasibleSolution
 - OsiChooseVariable, 47
- filename
 - OsiUnitTest::TestOutcome, 323
- fillColBounds
 - OsiGlpkSolverInterface, 116
- findIntegers
 - OsiSolverInterface, 260
- findIntegersAndSOS
 - OsiSolverInterface, 260
- findRange
 - OsiLotsize, 160
- firstBranch
 - OsiTwoWayBranchingObject, 297
- firstBranch_
 - OsiTwoWayBranchingObject, 298
- firstForcedObjectIndex
 - OsiChooseVariable, 48
- firstForcedObjectIndex_
 - OsiChooseVariable, 51

- firstForcedWhichWay
 - OsiChooseVariable, [49](#)
- firstForcedWhichWay_
 - OsiChooseVariable, [51](#)
- fixed
 - OsiSolverResult, [270](#)
- fixed_
 - OsiSolverResult, [271](#)
- floorCeiling
 - OsiLotsize, [160](#)
- forceFeasible
 - OsiSolverInterface, [261](#)
- forceIntoRange
 - OsiSolverInterface, [266](#)
- freeAllMemory
 - OsiCpxSolverInterface, [80](#)
 - OsiGlpkSolverInterface, [116](#)
 - OsiGrbSolverInterface, [146](#)
 - OsiMskSolverInterface, [185](#)
 - OsiSpxSolverInterface, [294](#)
- freeCachedColRim
 - OsiCpxSolverInterface, [79](#)
 - OsiGlpkSolverInterface, [116](#)
 - OsiGrbSolverInterface, [146](#)
 - OsiMskSolverInterface, [185](#)
 - OsiSpxSolverInterface, [294](#)
- freeCachedData
 - OsiCpxSolverInterface, [79](#)
 - OsiGlpkSolverInterface, [116](#)
 - OsiGrbSolverInterface, [146](#)
 - OsiMskSolverInterface, [185](#)
 - OsiSpxSolverInterface, [294](#)
- freeCachedMatrix
 - OsiCpxSolverInterface, [79](#)
 - OsiGlpkSolverInterface, [116](#)
 - OsiGrbSolverInterface, [146](#)
 - OsiMskSolverInterface, [185](#)
 - OsiSpxSolverInterface, [294](#)
- freeCachedResults
 - OsiCpxSolverInterface, [79](#)
 - OsiGlpkSolverInterface, [116](#)
 - OsiGrbSolverInterface, [146](#)
 - OsiMskSolverInterface, [185](#)
 - OsiSpxSolverInterface, [294](#)
 - OsiXprSolverInterface, [318](#)
- freeCachedRowRim
 - OsiCpxSolverInterface, [79](#)
 - OsiGlpkSolverInterface, [116](#)
 - OsiGrbSolverInterface, [146](#)
 - OsiMskSolverInterface, [185](#)
 - OsiSpxSolverInterface, [294](#)
- freeColSpace
 - OsiGrbSolverInterface, [145](#)
- freeColType
 - OsiCpxSolverInterface, [79](#)
 - OsiMskSolverInterface, [184](#)
- freeSolution
 - OsiXprSolverInterface, [318](#)
- G
 - GLP_PROB_DEFINED
 - OsiGlpkSolverInterface.hpp, [339](#)
- GRBenv
 - OsiGrbSolverInterface.hpp, [340](#)
- GRBmodel
 - OsiGrbSolverInterface.hpp, [340](#)
- getApplicationData
 - OsiAuxInfo, [22](#)
 - OsiSolverInterface, [259](#)
- getAuxiliaryInfo
 - OsiSolverInterface, [260](#)
- getBlInvACol
 - OsiCpxSolverInterface, [78](#)
 - OsiSolverInterface, [264](#)
- getBlInvARow
 - OsiCpxSolverInterface, [78](#)
 - OsiSolverInterface, [264](#)
- getBlInvCol
 - OsiCpxSolverInterface, [78](#)
 - OsiSolverInterface, [264](#)
- getBlInvRow
 - OsiCpxSolverInterface, [78](#)
 - OsiSolverInterface, [264](#)
- getBasics
 - OsiCpxSolverInterface, [78](#)
 - OsiSolverInterface, [264](#)
- getBasisStatus
 - OsiCpxSolverInterface, [78](#)
 - OsiGrbSolverInterface, [144](#)
 - OsiSolverInterface, [263](#)
- getColLower
 - OsiCpxSolverInterface, [68](#)
 - OsiGlpkSolverInterface, [106](#)
 - OsiGrbSolverInterface, [134](#)
 - OsiMskSolverInterface, [175](#)
 - OsiSolverInterface, [242](#)
 - OsiSpxSolverInterface, [287](#)
 - OsiXprSolverInterface, [308](#)
- getColName
 - OsiSolverInterface, [251](#)
- getColNames
 - OsiSolverInterface, [251](#)
- getColSolution
 - OsiCpxSolverInterface, [70](#)
 - OsiGlpkSolverInterface, [108](#)
 - OsiGrbSolverInterface, [136](#)
 - OsiMskSolverInterface, [177](#)
 - OsiSolverInterface, [245](#)

- OsiSpxSolverInterface, 288
- OsiXprSolverInterface, 310
- getColType
 - OsiSolverInterface, 244
- getColUpper
 - OsiCpxSolverInterface, 69
 - OsiGlpkSolverInterface, 106
 - OsiGrbSolverInterface, 134
 - OsiMskSolverInterface, 175
 - OsiSolverInterface, 242
 - OsiSpxSolverInterface, 287
 - OsiXprSolverInterface, 308
- getCountBySeverity
 - OsiUnitTest::TestOutcomes, 325
- getCtype
 - OsiCpxSolverInterface, 77
 - OsiGrbSolverInterface, 143
 - OsiMskSolverInterface, 183
- getDbParam
 - OsiCpxSolverInterface, 66
 - OsiGlpkSolverInterface, 104
 - OsiGrbSolverInterface, 132
 - OsiMskSolverInterface, 173
 - OsiSolverInterface, 238
 - OsiSpxSolverInterface, 285
 - OsiXprSolverInterface, 306
- getDualRays
 - OsiCpxSolverInterface, 71
 - OsiGlpkSolverInterface, 108
 - OsiGrbSolverInterface, 136
 - OsiMskSolverInterface, 177
 - OsiSolverInterface, 245
 - OsiSpxSolverInterface, 289
 - OsiXprSolverInterface, 311
- getEmptyWarmStart
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 105
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 174
 - OsiSolverInterface, 240
 - OsiSpxSolverInterface, 286
 - OsiXprSolverInterface, 307
- getEnvironmentPtr
 - OsiCpxSolverInterface, 77
 - OsiGrbSolverInterface, 143
 - OsiMskSolverInterface, 183
- getFractionalIndices
 - OsiSolverInterface, 246
- getHintParam
 - OsiGrbSolverInterface, 132
 - OsiSolverInterface, 239
- getInfinity
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 108
- OsiGrbSolverInterface, 136
- OsiMskSolverInterface, 177
- OsiSolverInterface, 244
- OsiSpxSolverInterface, 288
- OsiXprSolverInterface, 310
- getIntParam
 - OsiCpxSolverInterface, 66
 - OsiGlpkSolverInterface, 104
 - OsiGrbSolverInterface, 132
 - OsiMskSolverInterface, 173
 - OsiSolverInterface, 238
 - OsiSpxSolverInterface, 285
 - OsiXprSolverInterface, 306
- getIntegerTolerance
 - OsiSolverInterface, 239
- getIterationCount
 - OsiCpxSolverInterface, 71
 - OsiGlpkSolverInterface, 108
 - OsiGrbSolverInterface, 136
 - OsiMskSolverInterface, 177
 - OsiSolverInterface, 245
 - OsiSpxSolverInterface, 289
 - OsiXprSolverInterface, 310
- getLogFilePtr
 - OsiXprSolverInterface, 317
- getLpPtr
 - OsiCpxSolverInterface, 77
 - OsiGrbSolverInterface, 142
 - OsiMskSolverInterface, 183
 - OsiXprSolverInterface, 317
- getMatrixByCol
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 107
 - OsiGrbSolverInterface, 136
 - OsiMskSolverInterface, 176
 - OsiSolverInterface, 244
 - OsiSpxSolverInterface, 288
 - OsiXprSolverInterface, 310
- getMatrixByRow
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 107
 - OsiGrbSolverInterface, 136
 - OsiMskSolverInterface, 176
 - OsiSolverInterface, 244
 - OsiSpxSolverInterface, 288
 - OsiXprSolverInterface, 310
- getMipStart
 - OsiCpxSolverInterface, 67
 - OsiGrbSolverInterface, 132
 - OsiXprSolverInterface, 307
- getModelPtr
 - OsiGlpkSolverInterface, 114
- getMutableLpPtr
 - OsiCpxSolverInterface, 79

- OsiGrbSolverInterface, 145
- OsiMskSolverInterface, 184
- getMutableMatrixByCol
 - OsiSolverInterface, 244
- getMutableMatrixByRow
 - OsiSolverInterface, 244
- getMutableModelPtr
 - OsiGlpkSolverInterface, 115
- getNumApplied
 - OsiSolverInterface::ApplyCutsReturnCode, 15
- getNumCols
 - OsiCpxSolverInterface, 68
 - OsiGlpkSolverInterface, 106
 - OsiGrbSolverInterface, 134
 - OsiMskSolverInterface, 175
 - OsiPresolve, 199
 - OsiSolverInterface, 241
 - OsiSpxSolverInterface, 286
 - OsiXprSolverInterface, 308
- getNumElements
 - OsiCpxSolverInterface, 68
 - OsiGlpkSolverInterface, 106
 - OsiGrbSolverInterface, 134
 - OsiMskSolverInterface, 175
 - OsiSolverInterface, 242
 - OsiSpxSolverInterface, 286
 - OsiXprSolverInterface, 308
- getNumInconsistent
 - OsiSolverInterface::ApplyCutsReturnCode, 14
- getNumInconsistentWrtIntegerModel
 - OsiSolverInterface::ApplyCutsReturnCode, 14
- getNumIneffective
 - OsiSolverInterface::ApplyCutsReturnCode, 15
- getNumInfeasible
 - OsiSolverInterface::ApplyCutsReturnCode, 14
- getNumInstances
 - OsiGlpkSolverInterface, 115
 - OsiGrbSolverInterface, 143
 - OsiMskSolverInterface, 184
 - OsiXprSolverInterface, 317
- getNumIntVars
 - OsiXprSolverInterface, 318
- getNumIntegers
 - OsiSolverInterface, 242
- getNumRows
 - OsiCpxSolverInterface, 68
 - OsiGlpkSolverInterface, 106
 - OsiGrbSolverInterface, 134
 - OsiMskSolverInterface, 175
 - OsiPresolve, 199
 - OsiSolverInterface, 241
 - OsiSpxSolverInterface, 286
 - OsiXprSolverInterface, 308
- getObjCoefficients
 - OsiCpxSolverInterface, 69
 - OsiGlpkSolverInterface, 107
 - OsiGrbSolverInterface, 135
 - OsiMskSolverInterface, 176
 - OsiSolverInterface, 243
 - OsiSpxSolverInterface, 288
 - OsiXprSolverInterface, 309
- getObjName
 - OsiSolverInterface, 250
- getObjSense
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 107
 - OsiGrbSolverInterface, 135
 - OsiMskSolverInterface, 176
 - OsiSolverInterface, 243
 - OsiSpxSolverInterface, 288
 - OsiXprSolverInterface, 309
- getObjValue
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 108
 - OsiGrbSolverInterface, 136
 - OsiMskSolverInterface, 177
 - OsiSolverInterface, 245
 - OsiSpxSolverInterface, 289
 - OsiXprSolverInterface, 310
- getPointerToWarmStart
 - OsiSolverInterface, 241
- getPrimalRays
 - OsiCpxSolverInterface, 71
 - OsiGlpkSolverInterface, 108
 - OsiGrbSolverInterface, 137
 - OsiMskSolverInterface, 178
 - OsiSolverInterface, 246
 - OsiSpxSolverInterface, 289
 - OsiXprSolverInterface, 311
- getReducedCost
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 108
 - OsiGrbSolverInterface, 136
 - OsiMskSolverInterface, 177
 - OsiSolverInterface, 245
 - OsiSpxSolverInterface, 288
 - OsiXprSolverInterface, 310
- getReducedGradient
 - OsiSolverInterface, 263
- getRescode
 - OsiMskSolverInterface, 174
- getRightHandSide
 - OsiCpxSolverInterface, 69
 - OsiGlpkSolverInterface, 106
 - OsiGrbSolverInterface, 135
 - OsiMskSolverInterface, 175
 - OsiSolverInterface, 242
 - OsiSpxSolverInterface, 287

- OsiXprSolverInterface, 309
- getRowActivity
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 108
 - OsiGrbSolverInterface, 136
 - OsiMskSolverInterface, 177
 - OsiSolverInterface, 245
 - OsiSpxSolverInterface, 289
 - OsiXprSolverInterface, 310
- getRowCutDebugger
 - OsiSolverInterface, 262
- getRowCutDebuggerAlways
 - OsiSolverInterface, 262
- getRowLower
 - OsiCpxSolverInterface, 69
 - OsiGlpkSolverInterface, 107
 - OsiGrbSolverInterface, 135
 - OsiMskSolverInterface, 176
 - OsiSolverInterface, 243
 - OsiSpxSolverInterface, 287
 - OsiXprSolverInterface, 309
- getRowName
 - OsiSolverInterface, 250
- getRowNames
 - OsiSolverInterface, 250
- getRowPrice
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 108
 - OsiGrbSolverInterface, 136
 - OsiMskSolverInterface, 177
 - OsiSolverInterface, 245
 - OsiSpxSolverInterface, 288
 - OsiXprSolverInterface, 310
- getRowRange
 - OsiCpxSolverInterface, 69
 - OsiGlpkSolverInterface, 107
 - OsiGrbSolverInterface, 135
 - OsiMskSolverInterface, 176
 - OsiSolverInterface, 242
 - OsiSpxSolverInterface, 287
 - OsiXprSolverInterface, 309
- getRowSense
 - OsiCpxSolverInterface, 69
 - OsiGlpkSolverInterface, 106
 - OsiGrbSolverInterface, 134
 - OsiMskSolverInterface, 175
 - OsiSolverInterface, 242
 - OsiSpxSolverInterface, 287
 - OsiXprSolverInterface, 308
- getRowUpper
 - OsiCpxSolverInterface, 69
 - OsiGlpkSolverInterface, 107
 - OsiGrbSolverInterface, 135
 - OsiMskSolverInterface, 176
- OsiSolverInterface, 243
- OsiSpxSolverInterface, 288
- OsiXprSolverInterface, 309
- getStrParam
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 104
 - OsiGrbSolverInterface, 132
 - OsiMskSolverInterface, 173
 - OsiSolverInterface, 238
 - OsiSpxSolverInterface, 285
 - OsiXprSolverInterface, 306
- getStrictColSolution
 - OsiSolverInterface, 245
- getTimeLimit
 - OsiSpxSolverInterface, 285
- getVarTypes
 - OsiXprSolverInterface, 318
- getWarmStart
 - OsiCpxSolverInterface, 68
 - OsiGlpkSolverInterface, 105
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 174
 - OsiSolverInterface, 240
 - OsiSpxSolverInterface, 286
 - OsiXprSolverInterface, 307
- globalenv_
 - OsiGrbSolverInterface, 146
- globalenv_is_ours
 - OsiGrbSolverInterface, 146
- globallyValid
 - OsiCut, 85
- globallyValid_
 - OsiCut, 86
- globallyValidAsInteger
 - OsiCut, 85
- glp_prob, 19
 - _opaque_prob, 19
- goodObjectiveValue
 - OsiChooseVariable, 48
- goodObjectiveValue_
 - OsiChooseVariable, 50
- goodSolution
 - OsiChooseVariable, 48
- goodSolution_
 - OsiChooseVariable, 51
- gutsOfConstructor
 - OsiCpxSolverInterface, 79
 - OsiGlpkSolverInterface, 115
 - OsiGrbSolverInterface, 145
 - OsiMskSolverInterface, 185
 - OsiXprSolverInterface, 318
- gutsOfCopy
 - OsiCpxSolverInterface, 79
 - OsiCuts, 93

- OsiGlpkSolverInterface, 115
- OsiGrbSolverInterface, 145
- OsiMskSolverInterface, 185
- OsiPseudoCosts, 203
- OsiXprSolverInterface, 317
- gutsOfDelete
 - OsiPseudoCosts, 203
- gutsOfDestroy
 - OsiPresolve, 200
- gutsOfDestructor
 - OsiCpxSolverInterface, 79
 - OsiCuts, 93
 - OsiGlpkSolverInterface, 115
 - OsiGrbSolverInterface, 146
 - OsiMskSolverInterface, 185
 - OsiXprSolverInterface, 318
- H
- haltonerror
 - OsiUnitTest, 12
- handler_
 - OsiSolverInterface, 266
- hasSolution
 - OsiBabSolver, 26
- hintParam_
 - OsiSolverInterface, 268
- hintStrength_
 - OsiSolverInterface, 268
- hotStartCDualVal_
 - OsiGlpkSolverInterface, 118
- hotStartCStat_
 - OsiCpxSolverInterface, 80
 - OsiGlpkSolverInterface, 118
 - OsiGrbSolverInterface, 147
 - OsiMskSolverInterface, 186
 - OsiSpxSolverInterface, 294
- hotStartCStatSize_
 - OsiCpxSolverInterface, 80
 - OsiGlpkSolverInterface, 118
 - OsiGrbSolverInterface, 147
 - OsiMskSolverInterface, 186
 - OsiSpxSolverInterface, 294
- hotStartCVal_
 - OsiGlpkSolverInterface, 118
- hotStartMaxIteration_
 - OsiCpxSolverInterface, 80
 - OsiGlpkSolverInterface, 117
 - OsiGrbSolverInterface, 147
 - OsiMskSolverInterface, 186
 - OsiSpxSolverInterface, 295
- hotStartRDualVal_
 - OsiGlpkSolverInterface, 118
- hotStartRStat_
 - OsiCpxSolverInterface, 80
- OsiGlpkSolverInterface, 118
- OsiGrbSolverInterface, 147
- OsiMskSolverInterface, 186
- OsiSpxSolverInterface, 294
- hotStartRStatSize_
 - OsiCpxSolverInterface, 80
 - OsiGlpkSolverInterface, 118
 - OsiGrbSolverInterface, 147
 - OsiMskSolverInterface, 186
 - OsiSpxSolverInterface, 295
- hotStartRVal_
 - OsiGlpkSolverInterface, 118
- hotstartSolution_
 - OsiBranchingInformation, 33
- I
- iXprCallCount_
 - OsiXprSolverInterface, 318
- incrementApplied
 - OsiSolverInterface::ApplyCutsReturnCode, 15
- incrementExternallyInconsistent
 - OsiSolverInterface::ApplyCutsReturnCode, 15
- incrementIneffective
 - OsiSolverInterface::ApplyCutsReturnCode, 15
- incrementInfeasible
 - OsiSolverInterface::ApplyCutsReturnCode, 15
- incrementInstanceCounter
 - OsiGlpkSolverInterface, 114
 - OsiGrbSolverInterface, 143
 - OsiMskSolverInterface, 183
 - OsiXprSolverInterface, 316
- incrementInternallyInconsistent
 - OsiSolverInterface::ApplyCutsReturnCode, 15
- incrementNumberBranchesLeft
 - OsiBranchingObject, 37
- indexRegion_
 - OsiBranchingInformation, 34
- indices_
 - OsiSolverBranch, 223
- ineffective_
 - OsiSolverInterface::ApplyCutsReturnCode, 16
- infeasibility
 - OsiLotsize, 159
 - OsiObject, 191, 193
 - OsiSimpleInteger, 219
 - OsiSOS, 273
- infeasibility_
 - OsiObject, 194
- infeasible
 - OsiColCut, 56
 - OsiCut, 86
 - OsiRowCut, 209
- infeasible_
 - OsiSolverInterface::ApplyCutsReturnCode, 16

- info_
 - OsiGlpkSolverInterface, 117
- initialSolve
 - OsiCpxSolverInterface, 66
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 131
 - OsiMskSolverInterface, 172
 - OsiSolverInterface, 237
 - OsiSpxSolverInterface, 284
 - OsiXprSolverInterface, 306
- InitialSolver
 - OsiMskSolverInterface, 186
- initialize
 - OsiPseudoCosts, 204
- insert
 - OsiCuts, 90, 91
- insertIfNotDuplicate
 - OsiCuts, 90
- intInconsistent_
 - OsiSolverInterface::ApplyCutsReturnCode, 16
- intParam_
 - OsiSolverInterface, 267
- integerTolerance_
 - OsiBranchingInformation, 32
- integerValued_
 - OsiSOS, 276
- integerVariable_
 - OsiRowCutDebugger, 217
- invalidCut
 - OsiRowCutDebugger, 215
- isAbandoned
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 104
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 173
 - OsiSolverInterface, 240
 - OsiSpxSolverInterface, 285
 - OsiXprSolverInterface, 307
- isAbandoned_
 - OsiGlpkSolverInterface, 118
- isBinary
 - OsiSolverInterface, 243
- isContinuous
 - OsiCpxSolverInterface, 70
 - OsiGlpkSolverInterface, 107
 - OsiGrbSolverInterface, 135
 - OsiMskSolverInterface, 176
 - OsiSolverInterface, 243
 - OsiSpxSolverInterface, 288
 - OsiXprSolverInterface, 310
- isDataLoaded
 - OsiXprSolverInterface, 318
- isDemoLicense
 - OsiGrbSolverInterface, 143
- isDualInfeasible_
 - OsiGlpkSolverInterface, 119
- isDualObjectiveLimitReached
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 105
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 174
 - OsiSolverInterface, 240
 - OsiSpxSolverInterface, 285
 - OsiXprSolverInterface, 307
- isEquivalent
 - OsiUnitTest, 11
- isFeasible
 - OsiGlpkSolverInterface, 105
- isFeasible_
 - OsiGlpkSolverInterface, 119
- isFreeBinary
 - OsiSolverInterface, 244
- isInteger
 - OsiSolverInterface, 243
- isIntegerNonBinary
 - OsiSolverInterface, 243
- isIterationLimitReached
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 105
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 174
 - OsiSolverInterface, 240
 - OsiSpxSolverInterface, 286
 - OsiXprSolverInterface, 307
- isIterationLimitReached_
 - OsiGlpkSolverInterface, 118
- isLicenseError
 - OsiMskSolverInterface, 174
- isObjLowerLimitReached_
 - OsiGlpkSolverInterface, 119
- isObjUpperLimitReached_
 - OsiGlpkSolverInterface, 119
- isPrimInfeasible_
 - OsiGlpkSolverInterface, 119
- isPrimalObjectiveLimitReached
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 105
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 174
 - OsiSolverInterface, 240
 - OsiXprSolverInterface, 307
- isProvenDualInfeasible
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 105
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 174
 - OsiSolverInterface, 240
 - OsiSpxSolverInterface, 285

- OsiXprSolverInterface, 307
- isProvenOptimal
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 104
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 173
 - OsiSolverInterface, 240
 - OsiSpxSolverInterface, 285
 - OsiXprSolverInterface, 307
- isProvenPrimalInfeasible
 - OsiCpxSolverInterface, 67
 - OsiGlpkSolverInterface, 104
 - OsiGrbSolverInterface, 133
 - OsiMskSolverInterface, 173
 - OsiSolverInterface, 240
 - OsiSpxSolverInterface, 285
 - OsiXprSolverInterface, 307
- isTimeLimitReached
 - OsiGlpkSolverInterface, 105
 - OsiSpxSolverInterface, 286
- isTimeLimitReached_
 - OsiGlpkSolverInterface, 118
- iter_used_
 - OsiGlpkSolverInterface, 119
- iterationCount
 - OsiHotInfo, 153
- iterationCounts_
 - OsiHotInfo, 154
- iterator
 - OsiCuts::iterator, 20
- iterator_category
 - OsiCuts::const_iterator, 17
- ivarind_
 - OsiXprSolverInterface, 320
- ivartype_
 - OsiXprSolverInterface, 321
- K
- KEEPCACHED_ALL
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- KEEPCACHED_COLUMN
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- KEEPCACHED_MATRIX
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
- OsiMskSolverInterface, 172
- OsiSpxSolverInterface, 284
- KEEPCACHED_NONE
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- KEEPCACHED_PROBLEM
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- KEEPCACHED_RESULTS
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- KEEPCACHED_ROW
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- keepCachedFlag
 - OsiCpxSolverInterface, 65
 - OsiGlpkSolverInterface, 103
 - OsiGrbSolverInterface, 130
 - OsiMskSolverInterface, 172
 - OsiSpxSolverInterface, 284
- knownSolution_
 - OsiRowCutDebugger, 217
- knownValue_
 - OsiRowCutDebugger, 217
- L
- LAST
 - OsiUnitTest::TestOutcome, 322
- LPX
 - OsiGlpkSolverInterface.hpp, 339
- largestGap_
 - OsiLotsize, 161
- lastsolvewsmip
 - OsiXprSolverInterface, 321
- lb
 - OsiRowCut, 208
- lb_
 - OsiRowCut, 211
- lbs
 - OsiColCut, 55
- lbs_
 - OsiColCut, 57

- linenumber
 - OsiUnitTest::TestOutcome, [323](#)
- list_
 - OsiChooseVariable, [51](#)
- loadFromCoinModel
 - OsiSolverInterface, [257](#)
- loadProblem
 - OsiCpxSolverInterface, [75](#), [76](#)
 - OsiGlpkSolverInterface, [113](#), [114](#)
 - OsiGrbSolverInterface, [141](#), [142](#)
 - OsiMskSolverInterface, [182](#), [183](#)
 - OsiSolverInterface, [255–257](#)
 - OsiSpxSolverInterface, [292](#), [293](#)
 - OsiXprSolverInterface, [315](#), [316](#)
- localenv_
 - OsiGrbSolverInterface, [147](#)
- logFileName_
 - OsiXprSolverInterface, [318](#)
- logFilePtr_
 - OsiXprSolverInterface, [318](#)
- lower_
 - OsiBranchingInformation, [33](#)
- lp_
 - OsiCpxSolverInterface, [80](#)
 - OsiGlpkSolverInterface, [116](#)
 - OsiGrbSolverInterface, [147](#)
- M
- MSKenv_t
 - OsiMskSolverInterface.hpp, [341](#)
- MSKsolverused_
 - OsiMskSolverInterface, [186](#)
- MSKtask_t
 - OsiMskSolverInterface.hpp, [341](#)
- markHotStart
 - OsiCpxSolverInterface, [68](#)
 - OsiGlpkSolverInterface, [105](#)
 - OsiGrbSolverInterface, [134](#)
 - OsiMskSolverInterface, [174](#)
 - OsiSolverInterface, [241](#)
 - OsiSpxSolverInterface, [286](#)
 - OsiXprSolverInterface, [308](#)
- matrixByCol_
 - OsiCpxSolverInterface, [82](#)
 - OsiGlpkSolverInterface, [121](#)
 - OsiGrbSolverInterface, [149](#)
 - OsiMskSolverInterface, [188](#)
 - OsiSpxSolverInterface, [296](#)
 - OsiXprSolverInterface, [319](#)
- matrixByRow_
 - OsiCpxSolverInterface, [82](#)
 - OsiGlpkSolverInterface, [120](#)
 - OsiGrbSolverInterface, [148](#)
 - OsiMskSolverInterface, [187](#)
 - OsiSpxSolverInterface, [295](#)
 - OsiXprSolverInterface, [319](#)
- maxIteration_
 - OsiGlpkSolverInterface, [117](#)
- members
 - OsiSOS, [274](#)
- members_
 - OsiSOS, [275](#)
- messageHandler
 - OsiSolverInterface, [260](#)
- messages
 - OsiSolverInterface, [260](#)
- messages_
 - OsiSolverInterface, [267](#)
- messagesPointer
 - OsiSolverInterface, [260](#)
- mipBound
 - OsiBabSolver, [27](#)
- mipBound_
 - OsiBabSolver, [28](#)
- mipFeasible
 - OsiBabSolver, [27](#)
- model
 - OsiPresolve, [199](#)
- modifiableObject
 - OsiSolverInterface, [261](#)
- mostEffectiveCutPtr
 - OsiCuts, [92](#)
- Mskerr
 - OsiMskSolverInterface, [186](#)
- mutableMembers
 - OsiSOS, [275](#)
- mutableRow
 - OsiRowCut, [209](#)
- mutableWeights
 - OsiSOS, [275](#)
- N
- NOTE
 - OsiUnitTest::TestOutcome, [322](#)
- nameDisc_
 - OsiGlpkSolverInterface, [117](#)
 - OsiGrbSolverInterface, [147](#)
- nauxcols
 - OsiGrbSolverInterface, [149](#)
- ncols_
 - OsiPresolve, [201](#)
- nelems_
 - OsiPresolve, [201](#)
- newLanguage
 - OsiSolverInterface, [260](#)
- nonLinearValue
 - OsiPresolve, [200](#)
- nonLinearValue_

- OsiPresolve, [201](#)
- nrows_
 - OsiPresolve, [201](#)
- numInstances_
 - OsiGlpkSolverInterface, [116](#)
 - OsiGrbSolverInterface, [147](#)
 - OsiMskSolverInterface, [185](#)
 - OsiXprSolverInterface, [319](#)
- numResults_
 - OsiChooseStrong, [43](#)
- numberBeforeTrusted
 - OsiChooseStrong, [42](#)
 - OsiPseudoCosts, [203](#)
- numberBeforeTrusted_
 - OsiPseudoCosts, [205](#)
- numberBranches
 - OsiBranchingObject, [37](#)
- numberBranches_
 - OsiBranchingObject, [39](#)
- numberBranchesLeft
 - OsiBranchingObject, [37](#)
- numberBranchingSolutions_
 - OsiBranchingInformation, [34](#)
- numberColumns
 - OsiRowCutDebugger, [216](#)
- numberColumns_
 - OsiBranchingInformation, [32](#)
 - OsiRowCutDebugger, [217](#)
- numberIntegers_
 - OsiSolverInterface, [267](#)
- numberMembers
 - OsiSOS, [274](#)
- numberMembers_
 - OsiSOS, [276](#)
- numberObjects
 - OsiChooseStrong, [42](#)
 - OsiPseudoCosts, [204](#)
 - OsiSolverInterface, [261](#)
- numberObjects_
 - OsiPseudoCosts, [205](#)
 - OsiSolverInterface, [267](#)
- numberOnList
 - OsiChooseVariable, [49](#)
- numberOnList_
 - OsiChooseVariable, [52](#)
- numberPasses_
 - OsiPresolve, [202](#)
- numberRanges
 - OsiLotsize, [160](#)
- numberRanges_
 - OsiLotsize, [161](#)
- numberSolutions_
 - OsiBranchingInformation, [34](#)
- numberStrong
 - OsiChooseVariable, [49](#)
- numberStrong_
 - OsiChooseVariable, [52](#)
- numberStrongDone
 - OsiChooseVariable, [49](#)
- numberStrongDone_
 - OsiChooseVariable, [52](#)
- numberStrongFixed
 - OsiChooseVariable, [49](#)
- numberStrongFixed_
 - OsiChooseVariable, [52](#)
- numberStrongIterations
 - OsiChooseVariable, [49](#)
- numberStrongIterations_
 - OsiChooseVariable, [52](#)
- numberUnsatisfied
 - OsiChooseVariable, [49](#)
- numberUnsatisfied_
 - OsiChooseVariable, [51](#)
- numberWays
 - OsiObject, [192](#)
- numberWays_
 - OsiObject, [194](#)
- O
- OSI_VERSION
 - config_osi_default.h, [326](#)
- OSI_VERSION_MAJOR
 - config_osi_default.h, [326](#)
- OSI_VERSION_MINOR
 - config_osi_default.h, [326](#)
- OSIUNITTEST_QUOTEME
 - OsiUnitTests.hpp, [336](#)
- obj_
 - OsiCpxSolverInterface, [80](#)
 - OsiGlpkSolverInterface, [119](#)
 - OsiGrbSolverInterface, [147](#)
 - OsiMskSolverInterface, [186](#)
 - OsiSpxSolverInterface, [295](#)
- objName_
 - OsiSolverInterface, [268](#)
- ObjOffset_
 - OsiMskSolverInterface, [186](#)
- objOffset_
 - OsiGlpkSolverInterface, [117](#)
- objcoeffs_
 - OsiXprSolverInterface, [320](#)
- object
 - OsiSolverInterface, [261](#)
- object_
 - OsiSolverInterface, [267](#)
- objective_
 - OsiBranchingInformation, [33](#)
- objectiveValue

- OsiSolverResult, 270
- objectiveValue_
 - OsiBranchingInformation, 32
 - OsiSolverResult, 271
- objects
 - OsiSolverInterface, 261
- objsense_
 - OsiXprSolverInterface, 320
- onOptimalPath
 - OsiRowCutDebugger, 215
- operator<
 - OsiCut, 85
 - OsiCuts::const_iterator, 18
 - OsiCuts::iterator, 21
- operator>
 - OsiCut, 85
- operator*
 - OsiCuts::const_iterator, 18
 - OsiCuts::iterator, 20
- operator*=
 - OsiRowCut, 210
- operator()
 - OsiCuts::OsiCutCompare, 87
- operator++
 - OsiCuts::const_iterator, 18
 - OsiCuts::iterator, 20, 21
- operator+=
 - OsiRowCut, 210
- operator-=
 - OsiRowCut, 210
- operator/=
 - OsiRowCut, 210
- operator=
 - OsiAuxInfo, 22
 - OsiBabSolver, 25
 - OsiBranchingInformation, 31
 - OsiBranchingObject, 37
 - OsiChooseStrong, 41
 - OsiChooseVariable, 47
 - OsiColCut, 56
 - OsiCpxSolverInterface, 77
 - OsiCut, 86
 - OsiCuts, 93
 - OsiCuts::const_iterator, 18
 - OsiCuts::iterator, 20
 - OsiGlpkSolverInterface, 115
 - OsiGrbSolverInterface, 143
 - OsiHotInfo, 152
 - OsiIntegerBranchingObject, 156
 - OsiLotsize, 159
 - OsiLotsizeBranchingObject, 163
 - OsiMskSolverInterface, 184
 - OsiObject, 191
 - OsiObject2, 195
 - OsiPseudoCosts, 203
 - OsiRowCut, 210
 - OsiRowCut2, 213
 - OsiRowCutDebugger, 216
 - OsiSimpleInteger, 219
 - OsiSolverBranch, 223
 - OsiSolverInterface, 265
 - OsiSolverInterface::ApplyCutsReturnCode, 14
 - OsiSolverResult, 270
 - OsiSOS, 273
 - OsiSOSBranchingObject, 277
 - OsiSpxSolverInterface, 293
 - OsiTwoWayBranchingObject, 297
 - OsiXprSolverInterface, 317
- operator==
 - OsiColCut, 55
 - OsiCut, 85
 - OsiCuts::const_iterator, 18
 - OsiCuts::iterator, 21
 - OsiRowCut, 209
- optimalBasisIsAvailable
 - OsiSolverInterface, 263
- optimalSolution
 - OsiRowCutDebugger, 216
- optimalValue
 - OsiRowCutDebugger, 216
- originalColumn_
 - OsiPresolve, 201
- originalColumns
 - OsiPresolve, 199
- originalLower_
 - OsiSimpleInteger, 221
- originalLowerBound
 - OsiLotsize, 160
 - OsiSimpleInteger, 220
- originalModel
 - OsiPresolve, 199
- originalModel_
 - OsiPresolve, 200
- originalObject
 - OsiBranchingObject, 38
- originalObject_
 - OsiBranchingObject, 39
- originalObjectiveValue
 - OsiHotInfo, 152
- originalObjectiveValue_
 - OsiHotInfo, 154
- originalRow_
 - OsiPresolve, 201
- originalRows
 - OsiPresolve, 199
- originalUpper_
 - OsiSimpleInteger, 221
- originalUpperBound

- OsiLotsize, [160](#)
- OsiSimpleInteger, [220](#)
- OsiCpxSolverInterface
 - FREECACHED_COLUMN, [65](#)
 - FREECACHED_MATRIX, [65](#)
 - FREECACHED_RESULTS, [65](#)
 - FREECACHED_ROW, [65](#)
 - KEEPCACHED_ALL, [65](#)
 - KEEPCACHED_COLUMN, [65](#)
 - KEEPCACHED_MATRIX, [65](#)
 - KEEPCACHED_NONE, [65](#)
 - KEEPCACHED_PROBLEM, [65](#)
 - KEEPCACHED_RESULTS, [65](#)
 - KEEPCACHED_ROW, [65](#)
- OsiDoCrash
 - OsiSolverParameters.hpp, [333](#)
- OsiDoDualInInitial
 - OsiSolverParameters.hpp, [333](#)
- OsiDoDualInResolve
 - OsiSolverParameters.hpp, [333](#)
- OsiDoInBranchAndCut
 - OsiSolverParameters.hpp, [333](#)
- OsiDoPresolveInInitial
 - OsiSolverParameters.hpp, [333](#)
- OsiDoPresolveInResolve
 - OsiSolverParameters.hpp, [333](#)
- OsiDoReducePrint
 - OsiSolverParameters.hpp, [333](#)
- OsiDoScale
 - OsiSolverParameters.hpp, [333](#)
- OsiDualObjectiveLimit
 - OsiSolverParameters.hpp, [332](#)
- OsiDualTolerance
 - OsiSolverParameters.hpp, [332](#)
- OsiForceDo
 - OsiSolverParameters.hpp, [333](#)
- OsiGlpkSolverInterface
 - FREECACHED_COLUMN, [103](#)
 - FREECACHED_MATRIX, [103](#)
 - FREECACHED_RESULTS, [103](#)
 - FREECACHED_ROW, [103](#)
 - KEEPCACHED_ALL, [103](#)
 - KEEPCACHED_COLUMN, [103](#)
 - KEEPCACHED_MATRIX, [103](#)
 - KEEPCACHED_NONE, [103](#)
 - KEEPCACHED_PROBLEM, [103](#)
 - KEEPCACHED_RESULTS, [103](#)
 - KEEPCACHED_ROW, [103](#)
- OsiGrbSolverInterface
 - FREECACHED_COLUMN, [130](#)
 - FREECACHED_MATRIX, [130](#)
 - FREECACHED_RESULTS, [130](#)
 - FREECACHED_ROW, [130](#)
 - KEEPCACHED_ALL, [130](#)
 - KEEPCACHED_COLUMN, [130](#)
 - KEEPCACHED_MATRIX, [130](#)
 - KEEPCACHED_NONE, [130](#)
 - KEEPCACHED_PROBLEM, [130](#)
 - KEEPCACHED_RESULTS, [130](#)
 - KEEPCACHED_ROW, [130](#)
- OsiHintDo
 - OsiSolverParameters.hpp, [333](#)
- OsiHintIgnore
 - OsiSolverParameters.hpp, [333](#)
- OsiHintTry
 - OsiSolverParameters.hpp, [333](#)
- OsiLastDblParam
 - OsiSolverParameters.hpp, [332](#)
- OsiLastHintParam
 - OsiSolverParameters.hpp, [333](#)
- OsiLastIntParam
 - OsiSolverParameters.hpp, [332](#)
- OsiLastStrParam
 - OsiSolverParameters.hpp, [333](#)
- OsiMaxNumIteration
 - OsiSolverParameters.hpp, [332](#)
- OsiMaxNumIterationHotStart
 - OsiSolverParameters.hpp, [332](#)
- OsiMskSolverInterface
 - FREECACHED_COLUMN, [172](#)
 - FREECACHED_MATRIX, [172](#)
 - FREECACHED_RESULTS, [172](#)
 - FREECACHED_ROW, [172](#)
 - KEEPCACHED_ALL, [172](#)
 - KEEPCACHED_COLUMN, [172](#)
 - KEEPCACHED_MATRIX, [172](#)
 - KEEPCACHED_NONE, [172](#)
 - KEEPCACHED_PROBLEM, [172](#)
 - KEEPCACHED_RESULTS, [172](#)
 - KEEPCACHED_ROW, [172](#)
- OsiNameDiscipline
 - OsiSolverParameters.hpp, [332](#)
- OsiObjOffset
 - OsiSolverParameters.hpp, [332](#)
- OsiPrimalObjectiveLimit
 - OsiSolverParameters.hpp, [332](#)
- OsiPrimalTolerance
 - OsiSolverParameters.hpp, [332](#)
- OsiProbName
 - OsiSolverParameters.hpp, [333](#)
- OsiSolverName
 - OsiSolverParameters.hpp, [333](#)
- OsiSolverParameters.hpp
 - OsiDoCrash, [333](#)
 - OsiDoDualInInitial, [333](#)
 - OsiDoDualInResolve, [333](#)
 - OsiDoInBranchAndCut, [333](#)
 - OsiDoPresolveInInitial, [333](#)

- OsiDoPresolveInResolve, [333](#)
- OsiDoReducePrint, [333](#)
- OsiDoScale, [333](#)
- OsiDualObjectiveLimit, [332](#)
- OsiDualTolerance, [332](#)
- OsiForceDo, [333](#)
- OsiHintDo, [333](#)
- OsiHintIgnore, [333](#)
- OsiHintTry, [333](#)
- OsiLastDbIParam, [332](#)
- OsiLastHintParam, [333](#)
- OsiLastIntParam, [332](#)
- OsiLastStrParam, [333](#)
- OsiMaxNumIteration, [332](#)
- OsiMaxNumIterationHotStart, [332](#)
- OsiNameDiscipline, [332](#)
- OsiObjOffset, [332](#)
- OsiPrimalObjectiveLimit, [332](#)
- OsiPrimalTolerance, [332](#)
- OsiProbName, [333](#)
- OsiSolverName, [333](#)
- OsiSpxSolverInterface
 - FREECACHED_COLUMN, [284](#)
 - FREECACHED_MATRIX, [284](#)
 - FREECACHED_RESULTS, [284](#)
 - FREECACHED_ROW, [284](#)
 - KEEPCACHED_ALL, [284](#)
 - KEEPCACHED_COLUMN, [284](#)
 - KEEPCACHED_MATRIX, [284](#)
 - KEEPCACHED_NONE, [284](#)
 - KEEPCACHED_PROBLEM, [284](#)
 - KEEPCACHED_RESULTS, [284](#)
 - KEEPCACHED_ROW, [284](#)
- OsiUnitTest::TestOutcome
 - ERROR, [322](#)
 - LAST, [322](#)
 - NOTE, [322](#)
 - PASSED, [322](#)
 - WARNING, [322](#)
- OsiAuxInfo, [21](#)
 - ~OsiAuxInfo, [22](#)
 - appData_, [23](#)
 - clone, [22](#)
 - getApplicationData, [22](#)
 - operator=, [22](#)
 - OsiAuxInfo, [22](#)
 - OsiAuxInfo, [22](#)
- OsiBabSolver, [23](#)
 - ~OsiBabSolver, [25](#)
 - alwaysTryCutsAtRootNode, [26](#)
 - beforeLower, [27](#)
 - beforeLower_, [28](#)
 - beforeUpper, [28](#)
 - beforeUpper_, [28](#)
 - bestObjectiveValue, [27](#)
 - bestObjectiveValue_, [28](#)
 - bestSolution_, [28](#)
 - clone, [25](#)
 - extraCharacteristics, [27](#)
 - extraCharacteristics_, [29](#)
 - hasSolution, [26](#)
 - mipBound, [27](#)
 - mipBound_, [28](#)
 - mipFeasible, [27](#)
 - operator=, [25](#)
 - OsiBabSolver, [25](#)
 - OsiBabSolver, [25](#)
 - reducedCostsAccurate, [27](#)
 - setBeforeLower, [28](#)
 - setBeforeUpper, [28](#)
 - setExtraCharacteristics, [27](#)
 - setMipBound, [27](#)
 - setSolution, [26](#)
 - setSolver, [25](#)
 - setSolverType, [26](#)
 - sizeSolution_, [29](#)
 - solution, [25](#)
 - solutionAddsCuts, [26](#)
 - solver_, [28](#)
 - solverAccurate, [26](#)
 - solverType, [26](#)
 - solverType_, [28](#)
 - tryCuts, [27](#)
 - warmStart, [27](#)
- OsiBranchingInformation, [29](#)
 - ~OsiBranchingInformation, [31](#)
 - clone, [31](#)
 - columnLength_, [34](#)
 - columnStart_, [33](#)
 - cutoff_, [32](#)
 - defaultDual_, [32](#)
 - depth_, [34](#)
 - direction_, [32](#)
 - elementByColumn_, [33](#)
 - hotstartSolution_, [33](#)
 - indexRegion_, [34](#)
 - integerTolerance_, [32](#)
 - lower_, [33](#)
 - numberBranchingSolutions_, [34](#)
 - numberColumns_, [32](#)
 - numberSolutions_, [34](#)
 - objective_, [33](#)
 - objectiveValue_, [32](#)
 - operator=, [31](#)
 - OsiBranchingInformation, [31](#)
 - OsiBranchingInformation, [31](#)
 - owningSolution_, [34](#)
 - pi_, [33](#)

- primalTolerance_, 32
- row_, 34
- rowActivity_, 33
- rowLower_, 33
- rowUpper_, 33
- solution_, 33
- solver_, 32
- stateOfSearch_, 32
- timeRemaining_, 32
- upper_, 33
- usefulRegion_, 34
- OsiBranchingObject, 35
 - ~OsiBranchingObject, 37
 - boundBranch, 38
 - branch, 37
 - branchIndex, 38
 - branchIndex_, 39
 - checkIsCutoff, 38
 - clone, 37
 - columnNumber, 38
 - decrementNumberBranchesLeft, 37
 - incrementNumberBranchesLeft, 37
 - numberBranches, 37
 - numberBranches_, 39
 - numberBranchesLeft, 37
 - operator=, 37
 - originalObject, 38
 - originalObject_, 39
 - OsiBranchingObject, 36
 - OsiBranchingObject, 36
 - print, 38
 - setBranchingIndex, 38
 - setNumberBranchesLeft, 37
 - setOriginalObject, 38
 - value, 38
 - value_, 39
- OsiChooseStrong, 39
 - ~OsiChooseStrong, 41
 - chooseVariable, 41
 - clone, 41
 - doStrongBranching, 42
 - numResults_, 43
 - numberBeforeTrusted, 42
 - numberObjects, 42
 - operator=, 41
 - OsiChooseStrong, 41
 - OsiChooseStrong, 41
 - pseudoCosts, 42
 - pseudoCosts_, 43
 - resetResults, 42
 - results_, 43
 - setNumberBeforeTrusted, 42
 - setShadowPriceMode, 42
 - setupList, 41
 - shadowPriceMode, 42
 - shadowPriceMode_, 42
- OsiChooseVariable, 43
 - ~OsiChooseVariable, 47
 - bestObjectIndex, 48
 - bestObjectIndex_, 51
 - bestWhichWay, 48
 - bestWhichWay_, 51
 - candidates, 49
 - chooseVariable, 47
 - clearGoodSolution, 47
 - clone, 47
 - downChange, 48
 - downChange_, 50
 - feasibleSolution, 47
 - firstForcedObjectIndex, 48
 - firstForcedObjectIndex_, 51
 - firstForcedWhichWay, 49
 - firstForcedWhichWay_, 51
 - goodObjectiveValue, 48
 - goodObjectiveValue_, 50
 - goodSolution, 48
 - goodSolution_, 51
 - list_, 51
 - numberOnList, 49
 - numberOnList_, 52
 - numberStrong, 49
 - numberStrong_, 52
 - numberStrongDone, 49
 - numberStrongDone_, 52
 - numberStrongFixed, 49
 - numberStrongFixed_, 52
 - numberStrongIterations, 49
 - numberStrongIterations_, 52
 - numberUnsatisfied, 49
 - numberUnsatisfied_, 51
 - operator=, 47
 - OsiChooseVariable, 46
 - OsiChooseVariable, 46
 - saveSolution, 47
 - setBestObjectIndex, 48
 - setBestWhichWay, 48
 - setFirstForcedObjectIndex, 48
 - setFirstForcedWhichWay, 49
 - setNumberStrong, 49
 - setSolver, 50
 - setStatus, 50
 - setTrustStrongForBound, 50
 - setTrustStrongForSolution, 50
 - setupList, 47
 - solver_, 51
 - status, 50
 - status_, 51
 - trustStrongForBound, 49

- trustStrongForBound_, 52
- trustStrongForSolution, 50
- trustStrongForSolution_, 52
- upChange, 48
- upChange_, 50
- updateInformation, 47
- useful_, 51
- OsiClpSolverInterface
 - OsiSolverInterface::ApplyCutsReturnCode, 15
- OsiColCut, 52
 - ~OsiColCut, 54
 - clone, 56
 - consistent, 55, 56
 - infeasible, 56
 - lbs, 55
 - lbs_, 57
 - operator=, 56
 - operator==, 55
 - OsiColCut, 54
 - OsiColCutUnitTest, 56
 - OsiColCut, 54
 - print, 56
 - setLbs, 55
 - setUbs, 55
 - ubs, 55
 - ubs_, 57
 - violated, 56
- OsiColCutUnitTest
 - OsiColCut, 56
 - OsiUnitTests.hpp, 338
- OsiCollections.hpp
 - OsiVectorColCutPtr, 328
 - OsiVectorCutPtr, 328
 - OsiVectorDouble, 328
 - OsiVectorInt, 328
 - OsiVectorRowCutPtr, 328
- OsiCpxSolverInterface, 57
 - ~OsiCpxSolverInterface, 66
 - addCol, 74
 - addCols, 74
 - addRow, 74, 75
 - addRows, 75
 - applyColCut, 79
 - applyRowCut, 79
 - assignProblem, 75, 76
 - basisIsAvailable, 78
 - branchAndBound, 66
 - canDoSimplexInterface, 77
 - clone, 77
 - collower_, 80
 - colsol_, 81
 - coltype_, 82
 - coltypesize_, 82
 - colupper_, 81
 - deleteCols, 74
 - deleteRows, 75
 - disableFactorization, 78
 - disableSimplexInterface, 77
 - disableadvbasis, 82
 - domipstart, 82
 - enableFactorization, 77
 - enableSimplexInterface, 77
 - env_, 80
 - freeAllMemory, 80
 - freeCachedColRim, 79
 - freeCachedData, 79
 - freeCachedMatrix, 79
 - freeCachedResults, 79
 - freeCachedRowRim, 79
 - freeColType, 79
 - getBInvACol, 78
 - getBInvARow, 78
 - getBInvCol, 78
 - getBInvRow, 78
 - getBasics, 78
 - getBasisStatus, 78
 - getColLower, 68
 - getColSolution, 70
 - getColUpper, 69
 - getCtype, 77
 - getDbiParam, 66
 - getDualRays, 71
 - getEmptyWarmStart, 67
 - getEnvironmentPtr, 77
 - getInfinity, 70
 - getIntParam, 66
 - getIterationCount, 71
 - getLpPtr, 77
 - getMatrixByCol, 70
 - getMatrixByRow, 70
 - getMipStart, 67
 - getMutableLpPtr, 79
 - getNumCols, 68
 - getNumElements, 68
 - getNumRows, 68
 - getObjCoefficients, 69
 - getObjSense, 70
 - getObjValue, 70
 - getPrimalRays, 71
 - getReducedCost, 70
 - getRightHandSide, 69
 - getRowActivity, 70
 - getRowLower, 69
 - getRowPrice, 70
 - getRowRange, 69
 - getRowSense, 69
 - getRowUpper, 69
 - getStrParam, 67

- getWarmStart, 68
- gutsOfConstructor, 79
- gutsOfCopy, 79
- gutsOfDestructor, 79
- hotStartCStat_, 80
- hotStartCStatSize_, 80
- hotStartMaxIteration_, 80
- hotStartRStat_, 80
- hotStartRStatSize_, 80
- initialSolve, 66
- isAbandoned, 67
- isContinuous, 70
- isDualObjectiveLimitReached, 67
- isIterationLimitReached, 67
- isPrimalObjectiveLimitReached, 67
- isProvenDualInfeasible, 67
- isProvenOptimal, 67
- isProvenPrimalInfeasible, 67
- keepCachedFlag, 65
- loadProblem, 75, 76
- lp_, 80
- markHotStart, 68
- matrixByCol_, 82
- matrixByRow_, 82
- obj_, 80
- operator=, 77
- OsiCpxSolverInterface, 66
- OsiCpxSolverInterfaceUnitTest, 80
- OsiCpxSolverInterface, 66
- passInMessageHandler, 77
- probtypemip_, 82
- readMps, 76
- redcost_, 81
- reset, 77
- resizeColType, 79
- resolve, 66
- rhs_, 81
- rowact_, 81
- rowlower_, 81
- rowrange_, 81
- rowsense_, 81
- rowsol_, 81
- rowupper_, 81
- setColBounds, 72
- setColLower, 71
- setColSetBounds, 72
- setColSolution, 74
- setColUpper, 72
- setContinuous, 73
- setDbIParam, 66
- setIntParam, 66
- setInteger, 73
- setMipStart, 67
- setObjCoeff, 71
- setObjCoeffSet, 71
- setObjSense, 74
- setRowBounds, 72
- setRowLower, 72
- setRowPrice, 74
- setRowSetBounds, 73
- setRowSetTypes, 73
- setRowType, 72
- setRowUpper, 72
- setStrParam, 66
- setWarmStart, 68
- solveFromHotStart, 68
- switchToLP, 78
- switchToMIP, 78
- unmarkHotStart, 68
- writeMps, 76
- OsiCpxSolverInterface.hpp
 - CPXENVptr, 338
 - CPXLPptr, 338
 - OsiCpxSolverInterfaceUnitTest, 339
- OsiCpxSolverInterfaceUnitTest
 - OsiCpxSolverInterface, 80
 - OsiCpxSolverInterface.hpp, 339
- OsiCut, 82
 - ~OsiCut, 84
 - consistent, 86
 - effectiveness, 84
 - effectiveness_, 86
 - globallyValid, 85
 - globallyValid_, 86
 - globallyValidAsInteger, 85
 - infeasible, 86
 - operator<, 85
 - operator>, 85
 - operator=, 86
 - operator==, 85
 - OsiCut, 84
 - OsiCut, 84
 - print, 85
 - setEffectiveness, 84
 - setGloballyValid, 84, 85
 - setGloballyValidAsInteger, 85
 - setNotGloballyValid, 85
 - violated, 86
- OsiCuts, 87
 - ~OsiCuts, 90
 - begin, 93
 - colCut, 92
 - colCutPtr, 91, 92
 - colCutPtrs_, 94
 - dumpCuts, 93
 - end, 93
 - eraseAndDumpCuts, 93
 - eraseColCut, 92

- eraseRowCut, [92](#)
- gutsOfCopy, [93](#)
- gutsOfDestructor, [93](#)
- insert, [90](#), [91](#)
- insertIfNotDuplicate, [90](#)
- mostEffectiveCutPtr, [92](#)
- operator=, [93](#)
- OsiCuts, [90](#)
- OsiCutsUnitTest, [94](#)
- OsiCuts, [90](#)
- OsiCuts::const_iterator, [18](#)
- OsiCuts::iterator, [21](#)
- printCuts, [91](#)
- rowCut, [92](#)
- rowCutPtr, [91](#)
- rowCutPtrAndZap, [92](#)
- rowCutPtrs_, [94](#)
- sizeColCuts, [91](#)
- sizeCuts, [91](#)
- sizeRowCuts, [91](#)
- sort, [93](#)
- OsiCuts::OsiCutCompare, [87](#)
 - operator(), [87](#)
- OsiCuts::const_iterator, [16](#)
 - ~const_iterator, [18](#)
 - begin, [18](#)
 - colCutIndex_, [19](#)
 - const_iterator, [18](#)
 - cutP_, [19](#)
 - cutsPtr_, [18](#)
 - difference_type, [17](#)
 - end, [18](#)
 - iterator_category, [17](#)
 - operator<, [18](#)
 - operator*, [18](#)
 - operator++, [18](#)
 - operator=, [18](#)
 - operator==, [18](#)
 - OsiCuts, [18](#)
 - pointer, [17](#)
 - reference, [17](#)
 - rowCutIndex_, [18](#)
 - value_type, [17](#)
- OsiCuts::iterator, [19](#)
 - ~iterator, [20](#)
 - begin, [21](#)
 - colCutIndex_, [21](#)
 - cutP_, [21](#)
 - cuts_, [21](#)
 - end, [21](#)
 - iterator, [20](#)
 - operator<, [21](#)
 - operator*, [20](#)
 - operator++, [20](#), [21](#)
 - operator=, [20](#)
 - operator==, [21](#)
 - OsiCuts, [21](#)
 - rowCutIndex_, [21](#)
- OsiCutsUnitTest
 - OsiCuts, [94](#)
 - OsiUnitTests.hpp, [338](#)
- OsiDbiParam
 - OsiSolverParameters.hpp, [332](#)
- OsiGlpkSolverInterface, [94](#)
 - ~OsiGlpkSolverInterface, [103](#)
 - addCol, [111](#)
 - addCols, [112](#)
 - addRow, [112](#)
 - addRows, [112](#)
 - applyColCut, [115](#)
 - applyRowCut, [115](#)
 - assignProblem, [113](#)
 - bbWasLast_, [116](#)
 - branchAndBound, [103](#)
 - clone, [115](#)
 - collower_, [119](#)
 - colsol_, [120](#)
 - colupper_, [119](#)
 - ctype_, [119](#)
 - decrementInstanceCounter, [115](#)
 - deleteCols, [112](#)
 - deleteRows, [112](#)
 - dualObjectiveLimit_, [117](#)
 - dualTolerance_, [117](#)
 - fillColBounds, [116](#)
 - freeAllMemory, [116](#)
 - freeCachedColRim, [116](#)
 - freeCachedData, [116](#)
 - freeCachedMatrix, [116](#)
 - freeCachedResults, [116](#)
 - freeCachedRowRim, [116](#)
 - getColLower, [106](#)
 - getColSolution, [108](#)
 - getColUpper, [106](#)
 - getDbiParam, [104](#)
 - getDualRays, [108](#)
 - getEmptyWarmStart, [105](#)
 - getInfinity, [108](#)
 - getIntParam, [104](#)
 - getIterationCount, [108](#)
 - getMatrixByCol, [107](#)
 - getMatrixByRow, [107](#)
 - getModelPtr, [114](#)
 - getMutableModelPtr, [115](#)
 - getNumCols, [106](#)
 - getNumElements, [106](#)
 - getNumInstances, [115](#)
 - getNumRows, [106](#)

getObjCoefficients, 107
getObjSense, 107
getObjValue, 108
getPrimalRays, 108
getReducedCost, 108
getRightHandSide, 106
getRowActivity, 108
getRowLower, 107
getRowPrice, 108
getRowRange, 107
getRowSense, 106
getRowUpper, 107
getStrParam, 104
getWarmStart, 105
gutsOfConstructor, 115
gutsOfCopy, 115
gutsOfDestructor, 115
hotStartCDualVal_, 118
hotStartCStat_, 118
hotStartCStatSize_, 118
hotStartCVal_, 118
hotStartMaxIteration_, 117
hotStartRDualVal_, 118
hotStartRStat_, 118
hotStartRStatSize_, 118
hotStartRVal_, 118
incrementInstanceCounter, 114
info_, 117
initialSolve, 103
isAbandoned, 104
isAbandoned_, 118
isContinuous, 107
isDualInfeasible_, 119
isDualObjectiveLimitReached, 105
isFeasible, 105
isFeasible_, 119
isIterationLimitReached, 105
isIterationLimitReached_, 118
isObjLowerLimitReached_, 119
isObjUpperLimitReached_, 119
isPrimInfeasible_, 119
isPrimalObjectiveLimitReached, 105
isProvenDualInfeasible, 105
isProvenOptimal, 104
isProvenPrimalInfeasible, 104
isTimeLimitReached, 105
isTimeLimitReached_, 118
iter_used_, 119
keepCachedFlag, 103
loadProblem, 113, 114
lp_, 116
markHotStart, 105
matrixByCol_, 121
matrixByRow_, 120

maxIteration_, 117
nameDisc_, 117
numInstances_, 116
obj_, 119
objOffset_, 117
operator=, 115
OsiGlpkSolverInterface, 103
OsiGlpkSolverInterfaceUnitTest, 116
OsiGlpkSolverInterface, 103
primalObjectiveLimit_, 117
primalTolerance_, 117
printBounds, 116
probName_, 117
readMps, 114
redcost_, 120
reset, 115
resolve, 103
rhs_, 120
rowact_, 120
rowlower_, 120
rowrange_, 120
rowsense_, 120
rowsol_, 120
rowupper_, 120
setColBounds, 109
setColLower, 109
setColName, 114
setColSetBounds, 109
setColSolution, 111
setColUpper, 109
setContinuous, 111
setDbIParam, 104
setHintParam, 104
setIntParam, 103
setInteger, 111
setObjCoeff, 109
setObjName, 114
setObjSense, 111
setRowBounds, 110
setRowLower, 110
setRowName, 114
setRowPrice, 111
setRowSetBounds, 110
setRowSetTypes, 110
setRowType, 110
setRowUpper, 110
setStrParam, 104
setWarmStart, 105
solveFromHotStart, 106
unmarkHotStart, 106
writeMps, 114
OsiGlpkSolverInterface.hpp
LPX, 339
OsiGlpkSolverInterfaceUnitTest, 339

OsiGlpkSolverInterfaceUnitTest
 OsiGlpkSolverInterface, 116
 OsiGlpkSolverInterface.hpp, 339
 OsiGrbSolverInterface, 121
 ~OsiGrbSolverInterface, 131
 addCol, 140
 addCols, 140
 addRow, 140
 addRows, 141
 applyColCut, 145
 applyCuts, 144
 applyRowCut, 145
 assignProblem, 141, 142
 auxcolind, 150
 auxcolindspace, 149
 auxcolspace, 149
 basisIsAvailable, 144
 branchAndBound, 131
 canDoSimplexInterface, 143
 clone, 143
 collower_, 147
 colmap_G2O, 149
 colmap_O2G, 149
 colsol_, 148
 colspace_, 149
 coltype_, 149
 colupper_, 148
 convertToNormalRow, 146
 convertToRangedRow, 146
 decrementInstanceCounter, 143
 deleteCols, 140
 deleteRows, 141
 disableFactorization, 144
 disableSimplexInterface, 144
 domipstart, 149
 enableFactorization, 144
 enableSimplexInterface, 144
 freeAllMemory, 146
 freeCachedColRim, 146
 freeCachedData, 146
 freeCachedMatrix, 146
 freeCachedResults, 146
 freeCachedRowRim, 146
 freeColSpace, 145
 getBasisStatus, 144
 getColLower, 134
 getColSolution, 136
 getColUpper, 134
 getCtype, 143
 getDbIParam, 132
 getDualRays, 136
 getEmptyWarmStart, 133
 getEnvironmentPtr, 143
 getHintParam, 132
 getInfinity, 136
 getIntParam, 132
 getIterationCount, 136
 getLpPtr, 142
 getMatrixByCol, 136
 getMatrixByRow, 136
 getMipStart, 132
 getMutableLpPtr, 145
 getNumCols, 134
 getNumElements, 134
 getNumInstances, 143
 getNumRows, 134
 getObjCoefficients, 135
 getObjSense, 135
 getObjValue, 136
 getPrimalRays, 137
 getReducedCost, 136
 getRightHandSide, 135
 getRowActivity, 136
 getRowLower, 135
 getRowPrice, 136
 getRowRange, 135
 getRowSense, 134
 getRowUpper, 135
 getStrParam, 132
 getWarmStart, 133
 globalenv_, 146
 globalenv_is_ours, 146
 gutsOfConstructor, 145
 gutsOfCopy, 145
 gutsOfDestructor, 146
 hotStartCStat_, 147
 hotStartCStatSize_, 147
 hotStartMaxIteration_, 147
 hotStartRStat_, 147
 hotStartRStatSize_, 147
 incrementInstanceCounter, 143
 initialSolve, 131
 isAbandoned, 133
 isContinuous, 135
 isDemoLicense, 143
 isDualObjectiveLimitReached, 133
 isIterationLimitReached, 133
 isPrimalObjectiveLimitReached, 133
 isProvenDualInfeasible, 133
 isProvenOptimal, 133
 isProvenPrimalInfeasible, 133
 keepCachedFlag, 130
 loadProblem, 141, 142
 localenv_, 147
 lp_, 147
 markHotStart, 134
 matrixByCol_, 149
 matrixByRow_, 148

- nameDisc_, 147
- nauxcols, 149
- numInstances_, 147
- obj_, 147
- operator=, 143
- OsiGrbSolverInterface, 131
- OsiGrbSolverInterfaceUnitTest, 146
- OsiGrbSolverInterface, 131
- OsiSolverInterface::ApplyCutsReturnCode, 15
- probtymemp_, 149
- readMps, 142
- redcost_, 148
- reset, 143
- resizeAuxColIndSpace, 145
- resizeAuxColSpace, 145
- resizeColSpace, 145
- resolve, 131
- rhs_, 148
- rowact_, 148
- rowlower_, 148
- rowrange_, 148
- rowsense_, 148
- rowsol_, 148
- rowupper_, 148
- setColBounds, 137
- setColLower, 137
- setColName, 139
- setColSetBounds, 138
- setColSolution, 140
- setColUpper, 137
- setContinuous, 139
- setDbIParam, 131
- setEnvironment, 143
- setHintParam, 131
- setIntParam, 131
- setInteger, 139
- setMipStart, 132
- setObjCoeff, 137
- setObjCoeffSet, 137
- setObjSense, 140
- setRowBounds, 138
- setRowLower, 138
- setRowName, 139
- setRowPrice, 140
- setRowSetBounds, 138
- setRowSetTypes, 139
- setRowType, 138
- setRowUpper, 138
- setStrParam, 131
- setWarmStart, 133
- solveFromHotStart, 134
- switchToLP, 144
- switchToMIP, 144
- unmarkHotStart, 134
- writeMps, 142
- OsiGrbSolverInterface.hpp
 - GRBEnv, 340
 - GRBmodel, 340
 - OsiGrbSolverInterfaceUnitTest, 340
- OsiGrbSolverInterfaceUnitTest
 - OsiGrbSolverInterface, 146
 - OsiGrbSolverInterface.hpp, 340
- OsiHintParam
 - OsiSolverParameters.hpp, 333
- OsiHintStrength
 - OsiSolverParameters.hpp, 333
- OsiHotInfo, 150
 - ~OsiHotInfo, 152
 - branchingObject, 153
 - branchingObject_, 154
 - change, 153
 - changes_, 154
 - clone, 152
 - downChange, 152
 - downIterationCount, 153
 - downStatus, 153
 - iterationCount, 153
 - iterationCounts_, 154
 - operator=, 152
 - originalObjectiveValue, 152
 - originalObjectiveValue_, 154
 - OsiHotInfo, 152
 - OsiHotInfo, 152
 - setDownChange, 152
 - setDownStatus, 153
 - setUpChange, 152
 - setUpStatus, 153
 - status, 153
 - statuses_, 154
 - upChange, 152
 - upIterationCount, 153
 - upStatus, 153
 - updateInformation, 152
 - whichObject, 154
 - whichObject_, 154
- OsiIntParam
 - OsiSolverParameters.hpp, 332
- OsiIntegerBranchingObject, 154
 - ~OsiIntegerBranchingObject, 156
 - branch, 156
 - clone, 156
 - down_, 156
 - operator=, 156
 - OsiIntegerBranchingObject, 156
 - OsiIntegerBranchingObject, 156
 - print, 156
 - up_, 157
- OsiLotsize, 157

- ~OsiLotsize, 159
- bound, 160
- bound_, 161
- canDoHeuristics, 161
- canHandleShadowPrices, 161
- clone, 159
- columnNumber, 159
- columnNumber_, 161
- createBranch, 159
- downEstimate, 160
- feasibleRegion, 159
- findRange, 160
- floorCeiling, 160
- infeasibility, 159
- largestGap_, 161
- numberRanges, 160
- numberRanges_, 161
- operator=, 159
- originalLowerBound, 160
- originalUpperBound, 160
- OsiLotsize, 159
- OsiLotsize, 159
- range_, 161
- rangeType, 160
- rangeType_, 161
- resetBounds, 159
- resetSequenceEtc, 160
- setColumnNumber, 159
- upEstimate, 160
- OsiLotsizeBranchingObject, 162
 - ~OsiLotsizeBranchingObject, 163
 - branch, 163
 - clone, 163
 - down_, 163
 - operator=, 163
 - OsiLotsizeBranchingObject, 163
 - OsiLotsizeBranchingObject, 163
 - print, 163
 - up_, 163
- OsiMskSolverInterface, 164
 - ~OsiMskSolverInterface, 172
 - addCol, 180
 - addCols, 181
 - addRow, 181
 - addRows, 181
 - applyColCut, 184
 - applyRowCut, 184
 - assignProblem, 182
 - branchAndBound, 173
 - clone, 184
 - collower_, 186
 - colsol_, 187
 - coltype_, 188
 - coltypesize_, 188
 - colupper_, 187
 - decrementInstanceCounter, 183
 - definedSolution, 184
 - deleteCols, 181
 - deleteRows, 181
 - env_, 185
 - freeAllMemory, 185
 - freeCachedColRim, 185
 - freeCachedData, 185
 - freeCachedMatrix, 185
 - freeCachedResults, 185
 - freeCachedRowRim, 185
 - freeColType, 184
 - getColLower, 175
 - getColSolution, 177
 - getColUpper, 175
 - getCtype, 183
 - getDbIParam, 173
 - getDualRays, 177
 - getEmptyWarmStart, 174
 - getEnvironmentPtr, 183
 - getInfinity, 177
 - getIntParam, 173
 - getIterationCount, 177
 - getLpPtr, 183
 - getMatrixByCol, 176
 - getMatrixByRow, 176
 - getMutableLpPtr, 184
 - getNumCols, 175
 - getNumElements, 175
 - getNumInstances, 184
 - getNumRows, 175
 - getObjCoefficients, 176
 - getObjSense, 176
 - getObjValue, 177
 - getPrimalRays, 178
 - getReducedCost, 177
 - getRescode, 174
 - getRightHandSide, 175
 - getRowActivity, 177
 - getRowLower, 176
 - getRowPrice, 177
 - getRowRange, 176
 - getRowSense, 175
 - getRowUpper, 176
 - getStrParam, 173
 - getWarmStart, 174
 - gutsOfConstructor, 185
 - gutsOfCopy, 185
 - gutsOfDestructor, 185
 - hotStartCStat_, 186
 - hotStartCStatSize_, 186
 - hotStartMaxIteration_, 186
 - hotStartRStat_, 186

- hotStartRStatSize_, 186
- incrementInstanceCounter, 183
- initialSolve, 172
- InitialSolver, 186
- isAbandoned, 173
- isContinuous, 176
- isDualObjectiveLimitReached, 174
- isIterationLimitReached, 174
- isLicenseError, 174
- isPrimalObjectiveLimitReached, 174
- isProvenDualInfeasible, 174
- isProvenOptimal, 173
- isProvenPrimalInfeasible, 173
- keepCachedFlag, 172
- loadProblem, 182, 183
- MSKsolverused_, 186
- markHotStart, 174
- matrixByCol_, 188
- matrixByRow_, 187
- Mskerr, 186
- numInstances_, 185
- obj_, 186
- ObjOffset_, 186
- operator=, 184
- OsiMskSolverInterface, 172
- OsiMskSolverInterfaceUnitTest, 185
- OsiMskSolverInterface, 172
- passInMessageHandler, 183
- probtypemip_, 188
- readMps, 183
- redcost_, 187
- resizeColType, 184
- resolve, 172
- rhs_, 187
- rowact_, 187
- rowlower_, 187
- rowrange_, 187
- rowsense_, 187
- rowsol_, 187
- rowupper_, 187
- setColBounds, 178
- setColLower, 178
- setColSetBounds, 178
- setColSolution, 180
- setColUpper, 178
- setContinuous, 180
- setDbIParam, 173
- setIntParam, 173
- setInteger, 180
- setObjCoeff, 178
- setObjCoeffSet, 178
- setObjSense, 180
- setRowBounds, 179
- setRowLower, 179
- setRowPrice, 180
- setRowSetBounds, 179
- setRowSetTypes, 179
- setRowType, 179
- setRowUpper, 179
- setStrParam, 173
- setWarmStart, 174
- solveFromHotStart, 175
- solverUsed, 184
- switchToLP, 184
- switchToMIP, 184
- task_, 186
- unmarkHotStart, 175
- writeMps, 183
- OsiMskSolverInterface.hpp
 - MSKenv_t, 341
 - MSKtask_t, 341
 - OsiMskSolverInterfaceUnitTest, 341
- OsiMskSolverInterfaceUnitTest
 - OsiMskSolverInterface, 185
 - OsiMskSolverInterface.hpp, 341
- OsiNameVec
 - OsiSolverInterface, 237
- OsiObject, 188
 - ~OsiObject, 190
 - boundBranch, 192
 - canDoHeuristics, 192
 - canHandleShadowPrices, 192
 - canMoveToNearest, 192
 - checkInfeasibility, 191
 - clone, 191
 - columnNumber, 192
 - createBranch, 191
 - downEstimate, 193
 - feasibleRegion, 191
 - infeasibility, 191, 193
 - infeasibility_, 194
 - numberWays, 192
 - numberWays_, 194
 - operator=, 191
 - OsiObject, 190
 - OsiObject, 190
 - preferredWay, 193
 - priority, 192
 - priority_, 194
 - resetBounds, 193
 - resetSequenceEtc, 193
 - setNumberWays, 192
 - setPriority, 192
 - setWhichWay, 192
 - upEstimate, 193
 - updateAfter, 194
 - updateBefore, 193
 - whichWay, 193

- whichWay_, 194
- OsiObject2, 194
 - ~OsiObject2, 195
 - operator=, 195
 - OsiObject2, 195
 - OsiObject2, 195
 - otherInfeasibility_, 196
 - preferredWay, 196
 - preferredWay_, 196
 - setPreferredWay, 195
- OsiPresolve, 196
 - ~OsiPresolve, 198
 - getNumCols, 199
 - getNumRows, 199
 - gutsOfDestroy, 200
 - model, 199
 - ncols_, 201
 - nelems_, 201
 - nonLinearValue, 200
 - nonLinearValue_, 201
 - nrows_, 201
 - numberPasses_, 202
 - originalColumn_, 201
 - originalColumns, 199
 - originalModel, 199
 - originalModel_, 200
 - originalRow_, 201
 - originalRows, 199
 - OsiPresolve, 198
 - OsiPresolve, 198
 - paction_, 201
 - postsolve, 199, 200
 - presolve, 200
 - presolveActions_, 201
 - presolvedModel, 198
 - presolvedModel_, 201
 - setNonLinearValue, 199
 - setOriginalModel, 199
 - setPresolveActions, 200
- OsiPseudoCosts, 202
 - ~OsiPseudoCosts, 203
 - downNumber, 204
 - downNumber_, 205
 - downTotalChange, 204
 - downTotalChange_, 205
 - gutsOfCopy, 203
 - gutsOfDelete, 203
 - initialize, 204
 - numberBeforeTrusted, 203
 - numberBeforeTrusted_, 205
 - numberObjects, 204
 - numberObjects_, 205
 - operator=, 203
 - OsiPseudoCosts, 203
 - OsiPseudoCosts, 203
 - setNumberBeforeTrusted, 203
 - upNumber, 204
 - upNumber_, 205
 - upTotalChange, 204
 - upTotalChange_, 205
 - updateInformation, 204
- OsiRowCut, 205
 - ~OsiRowCut, 208
 - clone, 210
 - consistent, 209
 - infeasible, 209
 - lb, 208
 - lb_, 211
 - mutableRow, 209
 - operator*=, 210
 - operator+=, 210
 - operator-=, 210
 - operator/=: 210
 - operator=, 210
 - operator==, 209
 - OsiRowCut, 208
 - OsiRowCutUnitTest, 211
 - OsiRowCut, 208
 - print, 210
 - range, 209
 - rhs, 209
 - row, 209
 - row_, 211
 - sense, 208
 - setLb, 208
 - setRow, 209
 - setUb, 208
 - sortIncrIndex, 210
 - ub, 208
 - ub_, 211
 - violated, 210
- OsiRowCut.hpp
 - OsiRowCut_inline, 330
- OsiRowCut2, 211
 - ~OsiRowCut2, 212
 - clone, 213
 - operator=, 213
 - OsiRowCut2, 212
 - OsiRowCut2, 212
 - setWhichRow, 212
 - whichRow, 212
 - whichRow_, 213
- OsiRowCut_inline
 - OsiRowCut.hpp, 330
- OsiRowCutDebugger, 213
 - ~OsiRowCutDebugger, 215
 - activate, 216
 - active, 216

- integerVariable_, 217
- invalidCut, 215
- knownSolution_, 217
- knownValue_, 217
- numberColumns, 216
- numberColumns_, 217
- onOptimalPath, 215
- operator=, 216
- optimalSolution, 216
- optimalValue, 216
- OsiRowCutDebugger, 215
- OsiRowCutDebuggerUnitTest, 217
- OsiRowCutDebugger, 215
- printOptimalSolution, 216
- redoSolution, 216
- validateCuts, 215
- OsiRowCutDebuggerUnitTest
 - OsiRowCutDebugger, 217
 - OsiUnitTests.hpp, 338
- OsiRowCutUnitTest
 - OsiRowCut, 211
 - OsiUnitTests.hpp, 338
- OsiSOS, 271
 - ~OsiSOS, 273
 - canDoHeuristics, 275
 - canHandleShadowPrices, 275
 - clone, 273
 - createBranch, 274
 - downEstimate, 274
 - feasibleRegion, 273
 - infeasibility, 273
 - integerValued_, 276
 - members, 274
 - members_, 275
 - mutableMembers, 275
 - mutableWeights, 275
 - numberMembers, 274
 - numberMembers_, 276
 - operator=, 273
 - OsiSOS, 273
 - OsiSOS, 273
 - resetSequenceEtc, 274
 - setIntegerValued, 275
 - setNumberMembers, 275
 - setSosType, 275
 - setType, 274
 - sosType, 274
 - sosType_, 276
 - upEstimate, 274
 - weights, 274
 - weights_, 275
- OsiSOSBranchingObject, 276
 - ~OsiSOSBranchingObject, 277
 - branch, 277
 - clone, 277
 - operator=, 277
 - OsiSOSBranchingObject, 277
 - OsiSOSBranchingObject, 277
 - print, 277
- osiSerial_
 - OsiXprSolverInterface, 319
- OsiSimpleInteger, 217
 - ~OsiSimpleInteger, 219
 - canHandleShadowPrices, 221
 - clone, 219
 - columnNumber, 220
 - columnNumber_, 221
 - createBranch, 220
 - downEstimate, 221
 - feasibleRegion, 219
 - infeasibility, 219
 - operator=, 219
 - originalLower_, 221
 - originalLowerBound, 220
 - originalUpper_, 221
 - originalUpperBound, 220
 - OsiSimpleInteger, 219
 - OsiSimpleInteger, 219
 - resetBounds, 220
 - resetSequenceEtc, 220
 - setColumnNumber, 220
 - setOriginalLowerBound, 220
 - setOriginalUpperBound, 220
 - upEstimate, 220
- OsiSolverBranch, 221
 - ~OsiSolverBranch, 222
 - addBranch, 223
 - applyBounds, 223
 - bound_, 224
 - bounds, 223
 - feasibleOneWay, 223
 - indices_, 223
 - operator=, 223
 - OsiSolverBranch, 222
 - OsiSolverBranch, 222
 - start_, 223
 - starts, 223
 - which, 223
- OsiSolverInterface, 224
 - ~OsiSolverInterface, 237
 - activateRowCutDebugger, 261
 - addCol, 252
 - addCols, 252
 - addObjects, 261
 - addRow, 253
 - addRows, 253, 254
 - appDataEtc_, 267
 - applyColCut, 265

applyCuts, 255
applyRowCut, 265
applyRowCuts, 255
assignProblem, 256
basisIsAvailable, 262
branchAndBound, 237
canDoSimplexInterface, 262
clone, 265
colNames_, 268
columnType, 244
columnType_, 267
convertBoundToSense, 265
convertSenseToBound, 266
copyParameters, 239
dblParam_, 267
defaultHandler, 260
defaultHandler_, 267
deleteBranchingInfo, 255
deleteColNames, 251
deleteCols, 252
deleteObjects, 261
deleteRowNames, 250
deleteRows, 254
dfItRowColName, 250
disableFactorization, 262
disableSimplexInterface, 264
dualPivotResult, 265
enableFactorization, 262
enableSimplexInterface, 264
findIntegers, 260
findIntegersAndSOS, 260
forceFeasible, 261
forceIntoRange, 266
getApplicationData, 259
getAuxiliaryInfo, 260
getBInvACol, 264
getBInvARow, 264
getBInvCol, 264
getBInvRow, 264
getBasics, 264
getBasisStatus, 263
getColLower, 242
getColName, 251
getColNames, 251
getColSolution, 245
getColType, 244
getColUpper, 242
getDbIParam, 238
getDualRays, 245
getEmptyWarmStart, 240
getFractionalIndices, 246
getHintParam, 239
getInfinity, 244
getIntParam, 238
getIntegerTolerance, 239
getIterationCount, 245
getMatrixByCol, 244
getMatrixByRow, 244
getMutableMatrixByCol, 244
getMutableMatrixByRow, 244
getNumCols, 241
getNumElements, 242
getNumIntegers, 242
getNumRows, 241
getObjCoefficients, 243
getObjName, 250
getObjSense, 243
getObjValue, 245
getPointerToWarmStart, 241
getPrimalRays, 246
getReducedCost, 245
getReducedGradient, 263
getRightHandSide, 242
getRowActivity, 245
getRowCutDebugger, 262
getRowCutDebuggerAlways, 262
getRowLower, 243
getRowName, 250
getRowNames, 250
getRowPrice, 245
getRowRange, 242
getRowSense, 242
getRowUpper, 243
getStrParam, 238
getStrictColSolution, 245
getWarmStart, 240
handler_, 266
hintParam_, 268
hintStrength_, 268
initialSolve, 237
intParam_, 267
isAbandoned, 240
isBinary, 243
isContinuous, 243
isDualObjectiveLimitReached, 240
isFreeBinary, 244
isInteger, 243
isIntegerNonBinary, 243
isIterationLimitReached, 240
isPrimalObjectiveLimitReached, 240
isProvenDualInfeasible, 240
isProvenOptimal, 240
isProvenPrimalInfeasible, 240
loadFromCoinModel, 257
loadProblem, 255–257
markHotStart, 241
messageHandler, 260
messages, 260

- messages_, 267
- messagesPointer, 260
- modifiableObject, 261
- newLanguage, 260
- numberIntegers_, 267
- numberObjects, 261
- numberObjects_, 267
- objName_, 268
- object, 261
- object_, 267
- objects, 261
- operator=, 265
- optimalBasisIsAvailable, 263
- OsiNameVec, 237
- OsiSolverInterface, 237
- OsiSolverInterfaceCommonUnitTest, 266
- OsiSolverInterfaceMpsUnitTest, 266
- OsiSolverInterface, 237
- OsiSolverInterface::ApplyCutsReturnCode, 15
- passInMessageHandler, 260
- pivot, 264
- primalPivotResult, 265
- readGMPL, 258
- readLp, 259
- readMps, 257, 258
- reducedCostFix, 249
- replaceMatrix, 254
- replaceMatrixOptional, 254
- reset, 265
- resolve, 237
- restoreBaseModel, 254
- rowCutDebugger_, 266
- rowNames_, 268
- saveBaseModel, 254
- setApplicationData, 259
- setAuxiliaryInfo, 259
- setBasisStatus, 263
- setColBounds, 247
- setColLower, 247
- setColName, 251
- setColNames, 251
- setColSetBounds, 247
- setColSolution, 249
- setColUpper, 247
- setContinuous, 249
- setDbIParam, 238
- setHintParam, 238
- setInitialData, 266
- setIntParam, 237
- setInteger, 249
- setLanguage, 260
- setNumberObjects, 261
- setObjCoeff, 246
- setObjCoeffSet, 246
- setObjName, 250
- setObjSense, 246
- setObjective, 246
- setRowBounds, 248
- setRowColNames, 251
- setRowLower, 248
- setRowName, 250
- setRowNames, 250
- setRowPrice, 249
- setRowSetBounds, 248
- setRowSetTypes, 248
- setRowType, 248
- setRowUpper, 248
- setStrParam, 238
- setWarmStart, 241
- solveFromHotStart, 241
- strParam_, 267
- strictColSolution_, 268
- unmarkHotStart, 241
- writeLp, 258
- writeLpNative, 259
- writeMps, 258
- writeMpsNative, 258
- ws_, 268
- OsiSolverInterface::ApplyCutsReturnCode, 12
 - ~ApplyCutsReturnCode, 14
 - applied_, 16
 - ApplyCutsReturnCode, 14
 - extInconsistent_, 16
 - getNumApplied, 15
 - getNumInconsistent, 14
 - getNumInconsistentWrtIntegerModel, 14
 - getNumIneffective, 15
 - getNumInfeasible, 14
 - incrementApplied, 15
 - incrementExternallyInconsistent, 15
 - incrementIneffective, 15
 - incrementInfeasible, 15
 - incrementInternallyInconsistent, 15
 - ineffective_, 16
 - infeasible_, 16
 - intInconsistent_, 16
 - operator=, 14
 - OsiClpSolverInterface, 15
 - OsiGrbSolverInterface, 15
 - OsiSolverInterface, 15
- OsiSolverInterfaceCommonUnitTest
 - OsiSolverInterface, 266
 - OsiUnitTests.hpp, 337
- OsiSolverInterfaceMpsUnitTest
 - OsiSolverInterface, 266
 - OsiUnitTests.hpp, 337
- OsiSolverParameters.hpp
 - OsiDbIParam, 332

- OsiHintParam, 333
- OsiHintStrength, 333
- OsiIntParam, 332
- OsiStrParam, 332
- OsiSolverResult, 268
 - ~OsiSolverResult, 270
 - basis, 270
 - basis_, 271
 - createResult, 270
 - dualSolution, 270
 - dualSolution_, 271
 - fixed, 270
 - fixed_, 271
 - objectiveValue, 270
 - objectiveValue_, 271
 - operator=, 270
 - OsiSolverResult, 270
 - OsiSolverResult, 270
 - primalSolution, 270
 - primalSolution_, 271
 - restoreResult, 270
- OsiSpxSolverInterface, 277
 - ~OsiSpxSolverInterface, 284
 - addCol, 291
 - addRow, 291
 - applyColCut, 293
 - applyRowCut, 293
 - assignProblem, 292
 - branchAndBound, 284
 - clone, 293
 - colsol_, 295
 - deleteCols, 291
 - deleteRows, 291
 - freeAllMemory, 294
 - freeCachedColRim, 294
 - freeCachedData, 294
 - freeCachedMatrix, 294
 - freeCachedResults, 294
 - freeCachedRowRim, 294
 - getColLower, 287
 - getColSolution, 288
 - getColUpper, 287
 - getDbIParam, 285
 - getDualRays, 289
 - getEmptyWarmStart, 286
 - getInfinity, 288
 - getIntParam, 285
 - getIterationCount, 289
 - getMatrixByCol, 288
 - getMatrixByRow, 288
 - getNumCols, 286
 - getNumElements, 286
 - getNumRows, 286
 - getObjCoefficients, 288
 - getObjSense, 288
 - getObjValue, 289
 - getPrimalRays, 289
 - getReducedCost, 288
 - getRightHandSide, 287
 - getRowActivity, 289
 - getRowLower, 287
 - getRowPrice, 288
 - getRowRange, 287
 - getRowSense, 287
 - getRowUpper, 288
 - getStrParam, 285
 - getTimeLimit, 285
 - getWarmStart, 286
 - hotStartCStat_, 294
 - hotStartCStatSize_, 294
 - hotStartMaxIteration_, 295
 - hotStartRStat_, 294
 - hotStartRStatSize_, 295
 - initialSolve, 284
 - isAbandoned, 285
 - isContinuous, 288
 - isDualObjectiveLimitReached, 285
 - isIterationLimitReached, 286
 - isProvenDualInfeasible, 285
 - isProvenOptimal, 285
 - isProvenPrimalInfeasible, 285
 - isTimeLimitReached, 286
 - keepCachedFlag, 284
 - loadProblem, 292, 293
 - markHotStart, 286
 - matrixByCol_, 296
 - matrixByRow_, 295
 - obj_, 295
 - operator=, 293
 - OsiSpxSolverInterface, 284
 - OsiSpxSolverInterfaceUnitTest, 294
 - OsiSpxSolverInterface, 284
 - readMps, 293
 - redcost_, 295
 - resolve, 284
 - rhs_, 295
 - rowact_, 295
 - rowrange_, 295
 - rowsense_, 295
 - rowsol_, 295
 - setColBounds, 290
 - setColLower, 290
 - setColSolution, 291
 - setColUpper, 290
 - setContinuous, 290
 - setDbIParam, 285
 - setIntParam, 284
 - setInteger, 290

- setObjCoeff, [289](#)
 - setObjSense, [291](#)
 - setRowBounds, [290](#)
 - setRowLower, [290](#)
 - setRowPrice, [291](#)
 - setRowType, [290](#)
 - setRowUpper, [290](#)
 - setTimeLimit, [285](#)
 - setWarmStart, [286](#)
 - solveFromHotStart, [286](#)
 - soplex_, [294](#)
 - spxintvars_, [294](#)
 - unmarkHotStart, [286](#)
 - writeMps, [293](#)
- OsiSpxSolverInterface.hpp
 - OsiSpxSolverInterfaceUnitTest, [342](#)
- OsiSpxSolverInterfaceUnitTest
 - OsiSpxSolverInterface, [294](#)
 - OsiSpxSolverInterface.hpp, [342](#)
- OsiStrParam
 - OsiSolverParameters.hpp, [332](#)
- OsiTwoWayBranchingObject, [296](#)
 - ~OsiTwoWayBranchingObject, [297](#)
 - branch, [297](#)
 - firstBranch, [297](#)
 - firstBranch_, [298](#)
 - operator=, [297](#)
 - OsiTwoWayBranchingObject, [297](#)
 - OsiTwoWayBranchingObject, [297](#)
 - way, [297](#)
- OsiUnitTest, [10](#)
 - compareProblems, [11](#)
 - equivalentVectors, [11](#)
 - failureMessage, [11](#)
 - haltonerror, [12](#)
 - isEquivalent, [11](#)
 - OsiUnitTestAssertSeverityExpected, [12](#)
 - outcomes, [12](#)
 - processParameters, [12](#)
 - testingMessage, [11](#)
 - verbosity, [12](#)
- OsiUnitTest::TestOutcome, [321](#)
 - component, [323](#)
 - expected, [323](#)
 - filename, [323](#)
 - linenumber, [323](#)
 - print, [323](#)
 - severity, [323](#)
 - SeverityLevel, [322](#)
 - SeverityLevelName, [323](#)
 - TestOutcome, [323](#)
 - testcond, [323](#)
 - testname, [323](#)
- OsiUnitTest::TestOutcomes, [324](#)
 - add, [324](#)
 - getCountBySeverity, [325](#)
 - print, [325](#)
- OsiUnitTestAssertSeverityExpected
 - OsiUnitTest, [12](#)
- OsiUnitTests.hpp
 - OsiColCutUnitTest, [338](#)
 - OsiCutsUnitTest, [338](#)
 - OsiRowCutDebuggerUnitTest, [338](#)
 - OsiRowCutUnitTest, [338](#)
 - OsiSolverInterfaceCommonUnitTest, [337](#)
 - OsiSolverInterfaceMpsUnitTest, [337](#)
- OsiVectorColCutPtr
 - OsiCollections.hpp, [328](#)
- OsiVectorCutPtr
 - OsiCollections.hpp, [328](#)
- OsiVectorDouble
 - OsiCollections.hpp, [328](#)
- OsiVectorInt
 - OsiCollections.hpp, [328](#)
- OsiVectorRowCutPtr
 - OsiCollections.hpp, [328](#)
- OsiXprSolverInterface, [298](#)
 - ~OsiXprSolverInterface, [306](#)
 - activateMe, [318](#)
 - addCol, [314](#)
 - addCols, [314](#)
 - addRow, [314](#)
 - addRows, [314](#)
 - applyColCut, [317](#)
 - applyRowCut, [317](#)
 - assignProblem, [315](#), [316](#)
 - branchAndBound, [306](#)
 - clone, [317](#)
 - collower_, [319](#)
 - colprice_, [320](#)
 - colsol_, [320](#)
 - colupper_, [319](#)
 - decrementInstanceCounter, [317](#)
 - deleteCols, [314](#)
 - deleteRows, [315](#)
 - domipstart, [321](#)
 - freeCachedResults, [318](#)
 - freeSolution, [318](#)
 - getColLower, [308](#)
 - getColSolution, [310](#)
 - getColUpper, [308](#)
 - getDbiParam, [306](#)
 - getDualRays, [311](#)
 - getEmptyWarmStart, [307](#)
 - getInfinity, [310](#)
 - getIntParam, [306](#)
 - getIterationCount, [310](#)
 - getLogFilePtr, [317](#)

- getLpPtr, 317
- getMatrixByCol, 310
- getMatrixByRow, 310
- getMipStart, 307
- getNumCols, 308
- getNumElements, 308
- getNumInstances, 317
- getNumIntVars, 318
- getNumRows, 308
- getObjCoefficients, 309
- getObjSense, 309
- getObjValue, 310
- getPrimalRays, 311
- getReducedCost, 310
- getRightHandSide, 309
- getRowActivity, 310
- getRowLower, 309
- getRowPrice, 310
- getRowRange, 309
- getRowSense, 308
- getRowUpper, 309
- getStrParam, 306
- getVarTypes, 318
- getWarmStart, 307
- gutsOfConstructor, 318
- gutsOfCopy, 317
- gutsOfDestructor, 318
- iXprCallCount_, 318
- incrementInstanceCounter, 316
- initialSolve, 306
- isAbandoned, 307
- isContinuous, 310
- isDataLoaded, 318
- isDualObjectiveLimitReached, 307
- isIterationLimitReached, 307
- isPrimalObjectiveLimitReached, 307
- isProvenDualInfeasible, 307
- isProvenOptimal, 307
- isProvenPrimalInfeasible, 307
- ivarind_, 320
- ivartype_, 321
- lastsolvwasmip, 321
- loadProblem, 315, 316
- logFileName_, 318
- logFilePtr_, 318
- markHotStart, 308
- matrixByCol_, 319
- matrixByRow_, 319
- numInstances_, 319
- objcoeffs_, 320
- objsense_, 320
- operator=, 317
- osiSerial_, 319
- OsiXprSolverInterface, 305
- OsiXprSolverInterfaceUnitTest, 318
- OsiXprSolverInterface, 305
- passInMessageHandler, 316
- prob_, 319
- readMps, 316
- resolve, 306
- rhs_, 320
- rowact_, 320
- rowlower_, 319
- rowprice_, 320
- rowrange_, 320
- rowsense_, 320
- rowsol_, 320
- rowupper_, 319
- setColBounds, 311
- setColLower, 311
- setColSetBounds, 312
- setColSolution, 313
- setColUpper, 311
- setContinuous, 313
- setDbiParam, 306
- setIntParam, 306
- setInteger, 313
- setLogFileName, 317
- setMipStart, 307
- setObjCoeff, 311
- setObjSense, 313
- setRowBounds, 312
- setRowLower, 312
- setRowPrice, 314
- setRowSetBounds, 312
- setRowSetTypes, 313
- setRowType, 312
- setRowUpper, 312
- setStrParam, 306
- setWarmStart, 308
- solveFromHotStart, 308
- unmarkHotStart, 308
- vartype_, 321
- version, 317
- writeMps, 316
- xprProbname_, 319
- OsiXprSolverInterface.hpp
 - OsiXprSolverInterfaceUnitTest, 342
 - XPRsprob, 342
- OsiXprSolverInterfaceUnitTest
 - OsiXprSolverInterface, 318
 - OsiXprSolverInterface.hpp, 342
- otherInfeasibility_
 - OsiObject2, 196
- outcomes
 - OsiUnitTest, 12
- owningSolution_
 - OsiBranchingInformation, 34

P

PASSED

- OsiUnitTest::TestOutcome, 322

paction_

- OsiPresolve, 201

passInMessageHandler

- OsiCpxSolverInterface, 77
- OsiMskSolverInterface, 183
- OsiSolverInterface, 260
- OsiXprSolverInterface, 316

pi_

- OsiBranchingInformation, 33

pivot

- OsiSolverInterface, 264

pointer

- OsiCuts::const_iterator, 17

postsolve

- OsiPresolve, 199, 200

preferredWay

- OsiObject, 193
- OsiObject2, 196

preferredWay_

- OsiObject2, 196

presolve

- OsiPresolve, 200

presolveActions_

- OsiPresolve, 201

presolvedModel

- OsiPresolve, 198

presolvedModel_

- OsiPresolve, 201

primalObjectiveLimit_

- OsiGlpkSolverInterface, 117

primalPivotResult

- OsiSolverInterface, 265

primalSolution

- OsiSolverResult, 270

primalSolution_

- OsiSolverResult, 271

primalTolerance_

- OsiBranchingInformation, 32
- OsiGlpkSolverInterface, 117

print

- OsiBranchingObject, 38
- OsiColCut, 56
- OsiCut, 85
- OsiIntegerBranchingObject, 156
- OsiLotsizeBranchingObject, 163
- OsiRowCut, 210
- OsiSOSBranchingObject, 277
- OsiUnitTest::TestOutcome, 323
- OsiUnitTest::TestOutcomes, 325

printBounds

- OsiGlpkSolverInterface, 116

printCuts

- OsiCuts, 91

printOptimalSolution

- OsiRowCutDebugger, 216

priority

- OsiObject, 192

priority_

- OsiObject, 194

prob_

- OsiXprSolverInterface, 319

probName_

- OsiGlpkSolverInterface, 117

probtymip_

- OsiCpxSolverInterface, 82
- OsiGrbSolverInterface, 149
- OsiMskSolverInterface, 188

processParameters

- OsiUnitTest, 12

pseudoCosts

- OsiChooseStrong, 42

pseudoCosts_

- OsiChooseStrong, 43

R

range

- OsiRowCut, 209

range_

- OsiLotsize, 161

rangeType

- OsiLotsize, 160

rangeType_

- OsiLotsize, 161

readGMPL

- OsiSolverInterface, 258

readLp

- OsiSolverInterface, 259

readMps

- OsiCpxSolverInterface, 76
- OsiGlpkSolverInterface, 114
- OsiGrbSolverInterface, 142
- OsiMskSolverInterface, 183
- OsiSolverInterface, 257, 258
- OsiSpxSolverInterface, 293
- OsiXprSolverInterface, 316

redcost_

- OsiCpxSolverInterface, 81
- OsiGlpkSolverInterface, 120
- OsiGrbSolverInterface, 148
- OsiMskSolverInterface, 187
- OsiSpxSolverInterface, 295

redoSolution

- OsiRowCutDebugger, 216

reducedCostFix

- OsiSolverInterface, 249

- reducedCostsAccurate
 - OsiBabSolver, [27](#)
- reference
 - OsiCuts::const_iterator, [17](#)
- replaceMatrix
 - OsiSolverInterface, [254](#)
- replaceMatrixOptional
 - OsiSolverInterface, [254](#)
- reset
 - OsiCpxSolverInterface, [77](#)
 - OsiGlpkSolverInterface, [115](#)
 - OsiGrbSolverInterface, [143](#)
 - OsiSolverInterface, [265](#)
- resetBounds
 - OsiLotsize, [159](#)
 - OsiObject, [193](#)
 - OsiSimpleInteger, [220](#)
- resetResults
 - OsiChooseStrong, [42](#)
- resetSequenceEtc
 - OsiLotsize, [160](#)
 - OsiObject, [193](#)
 - OsiSimpleInteger, [220](#)
 - OsiSOS, [274](#)
- resizeAuxColIndSpace
 - OsiGrbSolverInterface, [145](#)
- resizeAuxColSpace
 - OsiGrbSolverInterface, [145](#)
- resizeColSpace
 - OsiGrbSolverInterface, [145](#)
- resizeColType
 - OsiCpxSolverInterface, [79](#)
 - OsiMskSolverInterface, [184](#)
- resolve
 - OsiCpxSolverInterface, [66](#)
 - OsiGlpkSolverInterface, [103](#)
 - OsiGrbSolverInterface, [131](#)
 - OsiMskSolverInterface, [172](#)
 - OsiSolverInterface, [237](#)
 - OsiSpxSolverInterface, [284](#)
 - OsiXprSolverInterface, [306](#)
- restoreBaseModel
 - OsiSolverInterface, [254](#)
- restoreResult
 - OsiSolverResult, [270](#)
- results_
 - OsiChooseStrong, [43](#)
- rhs
 - OsiRowCut, [209](#)
- rhs_
 - OsiCpxSolverInterface, [81](#)
 - OsiGlpkSolverInterface, [120](#)
 - OsiGrbSolverInterface, [148](#)
 - OsiMskSolverInterface, [187](#)
- OsiSpxSolverInterface, [295](#)
- OsiXprSolverInterface, [320](#)
- row
 - OsiRowCut, [209](#)
- row_
 - OsiBranchingInformation, [34](#)
 - OsiRowCut, [211](#)
- rowActivity_
 - OsiBranchingInformation, [33](#)
- rowCut
 - OsiCuts, [92](#)
- rowCutDebugger_
 - OsiSolverInterface, [266](#)
- rowCutIndex_
 - OsiCuts::const_iterator, [18](#)
 - OsiCuts::iterator, [21](#)
- rowCutPtr
 - OsiCuts, [91](#)
- rowCutPtrAndZap
 - OsiCuts, [92](#)
- rowCutPtrs_
 - OsiCuts, [94](#)
- rowLower_
 - OsiBranchingInformation, [33](#)
- rowNames_
 - OsiSolverInterface, [268](#)
- rowUpper_
 - OsiBranchingInformation, [33](#)
- rowact_
 - OsiCpxSolverInterface, [81](#)
 - OsiGlpkSolverInterface, [120](#)
 - OsiGrbSolverInterface, [148](#)
 - OsiMskSolverInterface, [187](#)
 - OsiSpxSolverInterface, [295](#)
 - OsiXprSolverInterface, [320](#)
- rowlower_
 - OsiCpxSolverInterface, [81](#)
 - OsiGlpkSolverInterface, [120](#)
 - OsiGrbSolverInterface, [148](#)
 - OsiMskSolverInterface, [187](#)
 - OsiXprSolverInterface, [319](#)
- rowprice_
 - OsiXprSolverInterface, [320](#)
- rowrange_
 - OsiCpxSolverInterface, [81](#)
 - OsiGlpkSolverInterface, [120](#)
 - OsiGrbSolverInterface, [148](#)
 - OsiMskSolverInterface, [187](#)
 - OsiSpxSolverInterface, [295](#)
 - OsiXprSolverInterface, [320](#)
- rowsense_
 - OsiCpxSolverInterface, [81](#)
 - OsiGlpkSolverInterface, [120](#)
 - OsiGrbSolverInterface, [148](#)

- OsiMskSolverInterface, [187](#)
- OsiSpxSolverInterface, [295](#)
- OsiXprSolverInterface, [320](#)
- rowsol_
 - OsiCpxSolverInterface, [81](#)
 - OsiGlpkSolverInterface, [120](#)
 - OsiGrbSolverInterface, [148](#)
 - OsiMskSolverInterface, [187](#)
 - OsiSpxSolverInterface, [295](#)
 - OsiXprSolverInterface, [320](#)
- rowupper_
 - OsiCpxSolverInterface, [81](#)
 - OsiGlpkSolverInterface, [120](#)
 - OsiGrbSolverInterface, [148](#)
 - OsiMskSolverInterface, [187](#)
 - OsiXprSolverInterface, [319](#)
- S
- saveBaseModel
 - OsiSolverInterface, [254](#)
- saveSolution
 - OsiChooseVariable, [47](#)
- sense
 - OsiRowCut, [208](#)
- setApplicationData
 - OsiSolverInterface, [259](#)
- setAuxiliaryInfo
 - OsiSolverInterface, [259](#)
- setBasisStatus
 - OsiSolverInterface, [263](#)
- setBeforeLower
 - OsiBabSolver, [28](#)
- setBeforeUpper
 - OsiBabSolver, [28](#)
- setBestObjectIndex
 - OsiChooseVariable, [48](#)
- setBestWhichWay
 - OsiChooseVariable, [48](#)
- setBranchingIndex
 - OsiBranchingObject, [38](#)
- setColBounds
 - OsiCpxSolverInterface, [72](#)
 - OsiGlpkSolverInterface, [109](#)
 - OsiGrbSolverInterface, [137](#)
 - OsiMskSolverInterface, [178](#)
 - OsiSolverInterface, [247](#)
 - OsiSpxSolverInterface, [290](#)
 - OsiXprSolverInterface, [311](#)
- setColLower
 - OsiCpxSolverInterface, [71](#)
 - OsiGlpkSolverInterface, [109](#)
 - OsiGrbSolverInterface, [137](#)
 - OsiMskSolverInterface, [178](#)
 - OsiSolverInterface, [247](#)
- OsiSpxSolverInterface, [290](#)
- OsiXprSolverInterface, [311](#)
- setColName
 - OsiGlpkSolverInterface, [114](#)
 - OsiGrbSolverInterface, [139](#)
 - OsiSolverInterface, [251](#)
- setColNames
 - OsiSolverInterface, [251](#)
- setColSetBounds
 - OsiCpxSolverInterface, [72](#)
 - OsiGlpkSolverInterface, [109](#)
 - OsiGrbSolverInterface, [138](#)
 - OsiMskSolverInterface, [178](#)
 - OsiSolverInterface, [247](#)
 - OsiXprSolverInterface, [312](#)
- setColSolution
 - OsiCpxSolverInterface, [74](#)
 - OsiGlpkSolverInterface, [111](#)
 - OsiGrbSolverInterface, [140](#)
 - OsiMskSolverInterface, [180](#)
 - OsiSolverInterface, [249](#)
 - OsiSpxSolverInterface, [291](#)
 - OsiXprSolverInterface, [313](#)
- setColUpper
 - OsiCpxSolverInterface, [72](#)
 - OsiGlpkSolverInterface, [109](#)
 - OsiGrbSolverInterface, [137](#)
 - OsiMskSolverInterface, [178](#)
 - OsiSolverInterface, [247](#)
 - OsiSpxSolverInterface, [290](#)
 - OsiXprSolverInterface, [311](#)
- setColumnNumber
 - OsiLotsize, [159](#)
 - OsiSimpleInteger, [220](#)
- setContinuous
 - OsiCpxSolverInterface, [73](#)
 - OsiGlpkSolverInterface, [111](#)
 - OsiGrbSolverInterface, [139](#)
 - OsiMskSolverInterface, [180](#)
 - OsiSolverInterface, [249](#)
 - OsiSpxSolverInterface, [290](#)
 - OsiXprSolverInterface, [313](#)
- setDblParam
 - OsiCpxSolverInterface, [66](#)
 - OsiGlpkSolverInterface, [104](#)
 - OsiGrbSolverInterface, [131](#)
 - OsiMskSolverInterface, [173](#)
 - OsiSolverInterface, [238](#)
 - OsiSpxSolverInterface, [285](#)
 - OsiXprSolverInterface, [306](#)
- setDownChange
 - OsiHotInfo, [152](#)
- setDownStatus
 - OsiHotInfo, [153](#)

- setEffectiveness
 - OsiCut, [84](#)
- setEnvironment
 - OsiGrbSolverInterface, [143](#)
- setExtraCharacteristics
 - OsiBabSolver, [27](#)
- setFirstForcedObjectIndex
 - OsiChooseVariable, [48](#)
- setFirstForcedWhichWay
 - OsiChooseVariable, [49](#)
- setGloballyValid
 - OsiCut, [84](#), [85](#)
- setGloballyValidAsInteger
 - OsiCut, [85](#)
- setHintParam
 - OsiGlpkSolverInterface, [104](#)
 - OsiGrbSolverInterface, [131](#)
 - OsiSolverInterface, [238](#)
- setInitialData
 - OsiSolverInterface, [266](#)
- setIntParam
 - OsiCpxSolverInterface, [66](#)
 - OsiGlpkSolverInterface, [103](#)
 - OsiGrbSolverInterface, [131](#)
 - OsiMskSolverInterface, [173](#)
 - OsiSolverInterface, [237](#)
 - OsiSpxSolverInterface, [284](#)
 - OsiXprSolverInterface, [306](#)
- setInteger
 - OsiCpxSolverInterface, [73](#)
 - OsiGlpkSolverInterface, [111](#)
 - OsiGrbSolverInterface, [139](#)
 - OsiMskSolverInterface, [180](#)
 - OsiSolverInterface, [249](#)
 - OsiSpxSolverInterface, [290](#)
 - OsiXprSolverInterface, [313](#)
- setIntegerValued
 - OsiSOS, [275](#)
- setLanguage
 - OsiSolverInterface, [260](#)
- setLb
 - OsiRowCut, [208](#)
- setLbs
 - OsiColCut, [55](#)
- setLogFileName
 - OsiXprSolverInterface, [317](#)
- setMipBound
 - OsiBabSolver, [27](#)
- setMipStart
 - OsiCpxSolverInterface, [67](#)
 - OsiGrbSolverInterface, [132](#)
 - OsiXprSolverInterface, [307](#)
- setNonLinearValue
 - OsiPresolve, [199](#)
- setNotGloballyValid
 - OsiCut, [85](#)
- setNumberBeforeTrusted
 - OsiChooseStrong, [42](#)
 - OsiPseudoCosts, [203](#)
- setNumberBranchesLeft
 - OsiBranchingObject, [37](#)
- setNumberMembers
 - OsiSOS, [275](#)
- setNumberObjects
 - OsiSolverInterface, [261](#)
- setNumberStrong
 - OsiChooseVariable, [49](#)
- setNumberWays
 - OsiObject, [192](#)
- setObjCoeff
 - OsiCpxSolverInterface, [71](#)
 - OsiGlpkSolverInterface, [109](#)
 - OsiGrbSolverInterface, [137](#)
 - OsiMskSolverInterface, [178](#)
 - OsiSolverInterface, [246](#)
 - OsiSpxSolverInterface, [289](#)
 - OsiXprSolverInterface, [311](#)
- setObjCoeffSet
 - OsiCpxSolverInterface, [71](#)
 - OsiGrbSolverInterface, [137](#)
 - OsiMskSolverInterface, [178](#)
 - OsiSolverInterface, [246](#)
- setObjName
 - OsiGlpkSolverInterface, [114](#)
 - OsiSolverInterface, [250](#)
- setObjSense
 - OsiCpxSolverInterface, [74](#)
 - OsiGlpkSolverInterface, [111](#)
 - OsiGrbSolverInterface, [140](#)
 - OsiMskSolverInterface, [180](#)
 - OsiSolverInterface, [246](#)
 - OsiSpxSolverInterface, [291](#)
 - OsiXprSolverInterface, [313](#)
- setObjective
 - OsiSolverInterface, [246](#)
- setOriginalLowerBound
 - OsiSimpleInteger, [220](#)
- setOriginalModel
 - OsiPresolve, [199](#)
- setOriginalObject
 - OsiBranchingObject, [38](#)
- setOriginalUpperBound
 - OsiSimpleInteger, [220](#)
- setPreferredWay
 - OsiObject2, [195](#)
- setPresolveActions
 - OsiPresolve, [200](#)
- setPriority

- OsiObject, [192](#)
- setRow
 - OsiRowCut, [209](#)
- setRowBounds
 - OsiCpxSolverInterface, [72](#)
 - OsiGlpkSolverInterface, [110](#)
 - OsiGrbSolverInterface, [138](#)
 - OsiMskSolverInterface, [179](#)
 - OsiSolverInterface, [248](#)
 - OsiSpxSolverInterface, [290](#)
 - OsiXprSolverInterface, [312](#)
- setRowColNames
 - OsiSolverInterface, [251](#)
- setRowLower
 - OsiCpxSolverInterface, [72](#)
 - OsiGlpkSolverInterface, [110](#)
 - OsiGrbSolverInterface, [138](#)
 - OsiMskSolverInterface, [179](#)
 - OsiSolverInterface, [248](#)
 - OsiSpxSolverInterface, [290](#)
 - OsiXprSolverInterface, [312](#)
- setRowName
 - OsiGlpkSolverInterface, [114](#)
 - OsiGrbSolverInterface, [139](#)
 - OsiSolverInterface, [250](#)
- setRowNames
 - OsiSolverInterface, [250](#)
- setRowPrice
 - OsiCpxSolverInterface, [74](#)
 - OsiGlpkSolverInterface, [111](#)
 - OsiGrbSolverInterface, [140](#)
 - OsiMskSolverInterface, [180](#)
 - OsiSolverInterface, [249](#)
 - OsiSpxSolverInterface, [291](#)
 - OsiXprSolverInterface, [314](#)
- setRowSetBounds
 - OsiCpxSolverInterface, [73](#)
 - OsiGlpkSolverInterface, [110](#)
 - OsiGrbSolverInterface, [138](#)
 - OsiMskSolverInterface, [179](#)
 - OsiSolverInterface, [248](#)
 - OsiXprSolverInterface, [312](#)
- setRowSetTypes
 - OsiCpxSolverInterface, [73](#)
 - OsiGlpkSolverInterface, [110](#)
 - OsiGrbSolverInterface, [139](#)
 - OsiMskSolverInterface, [179](#)
 - OsiSolverInterface, [248](#)
 - OsiXprSolverInterface, [313](#)
- setRowType
 - OsiCpxSolverInterface, [72](#)
 - OsiGlpkSolverInterface, [110](#)
 - OsiGrbSolverInterface, [138](#)
 - OsiMskSolverInterface, [179](#)
- OsiSolverInterface, [248](#)
- OsiSpxSolverInterface, [290](#)
- OsiXprSolverInterface, [312](#)
- setRowUpper
 - OsiCpxSolverInterface, [72](#)
 - OsiGlpkSolverInterface, [110](#)
 - OsiGrbSolverInterface, [138](#)
 - OsiMskSolverInterface, [179](#)
 - OsiSolverInterface, [248](#)
 - OsiSpxSolverInterface, [290](#)
 - OsiXprSolverInterface, [312](#)
- setShadowPriceMode
 - OsiChooseStrong, [42](#)
- setSolution
 - OsiBabSolver, [26](#)
- setSolver
 - OsiBabSolver, [25](#)
 - OsiChooseVariable, [50](#)
- setSolverType
 - OsiBabSolver, [26](#)
- setSosType
 - OsiSOS, [275](#)
- setStatus
 - OsiChooseVariable, [50](#)
- setStrParam
 - OsiCpxSolverInterface, [66](#)
 - OsiGlpkSolverInterface, [104](#)
 - OsiGrbSolverInterface, [131](#)
 - OsiMskSolverInterface, [173](#)
 - OsiSolverInterface, [238](#)
 - OsiXprSolverInterface, [306](#)
- setTimeLimit
 - OsiSpxSolverInterface, [285](#)
- setTrustStrongForBound
 - OsiChooseVariable, [50](#)
- setTrustStrongForSolution
 - OsiChooseVariable, [50](#)
- setType
 - OsiSOS, [274](#)
- setUb
 - OsiRowCut, [208](#)
- setUbs
 - OsiColCut, [55](#)
- setUpChange
 - OsiHotInfo, [152](#)
- setUpStatus
 - OsiHotInfo, [153](#)
- setWarmStart
 - OsiCpxSolverInterface, [68](#)
 - OsiGlpkSolverInterface, [105](#)
 - OsiGrbSolverInterface, [133](#)
 - OsiMskSolverInterface, [174](#)
 - OsiSolverInterface, [241](#)
 - OsiSpxSolverInterface, [286](#)

- OsiXprSolverInterface, 308
- setWhichRow
 - OsiRowCut2, 212
- setWhichWay
 - OsiObject, 192
- setupList
 - OsiChooseStrong, 41
 - OsiChooseVariable, 47
- severity
 - OsiUnitTest::TestOutcome, 323
- SeverityLevel
 - OsiUnitTest::TestOutcome, 322
- SeverityLevelName
 - OsiUnitTest::TestOutcome, 323
- shadowPriceMode
 - OsiChooseStrong, 42
- shadowPriceMode_
 - OsiChooseStrong, 42
- sizeColCuts
 - OsiCuts, 91
- sizeCuts
 - OsiCuts, 91
- sizeRowCuts
 - OsiCuts, 91
- sizeSolution_
 - OsiBabSolver, 29
- solution
 - OsiBabSolver, 25
- solution_
 - OsiBranchingInformation, 33
- solutionAddsCuts
 - OsiBabSolver, 26
- solveFromHotStart
 - OsiCpxSolverInterface, 68
 - OsiGlpkSolverInterface, 106
 - OsiGrbSolverInterface, 134
 - OsiMskSolverInterface, 175
 - OsiSolverInterface, 241
 - OsiSpxSolverInterface, 286
 - OsiXprSolverInterface, 308
- solver_
 - OsiBabSolver, 28
 - OsiBranchingInformation, 32
 - OsiChooseVariable, 51
- solverAccurate
 - OsiBabSolver, 26
- solverType
 - OsiBabSolver, 26
- solverType_
 - OsiBabSolver, 28
- solverUsed
 - OsiMskSolverInterface, 184
- soplex, 12
- soplex_
 - OsiSpxSolverInterface, 294
- sort
 - OsiCuts, 93
- sortIncrIndex
 - OsiRowCut, 210
- sosType
 - OsiSOS, 274
- sosType_
 - OsiSOS, 276
- spxintvars_
 - OsiSpxSolverInterface, 294
- start_
 - OsiSolverBranch, 223
- starts
 - OsiSolverBranch, 223
- stateOfSearch_
 - OsiBranchingInformation, 32
- status
 - OsiChooseVariable, 50
 - OsiHotInfo, 153
- status_
 - OsiChooseVariable, 51
- statuses_
 - OsiHotInfo, 154
- strParam_
 - OsiSolverInterface, 267
- strictColSolution_
 - OsiSolverInterface, 268
- switchToLP
 - OsiCpxSolverInterface, 78
 - OsiGrbSolverInterface, 144
 - OsiMskSolverInterface, 184
- switchToMIP
 - OsiCpxSolverInterface, 78
 - OsiGrbSolverInterface, 144
 - OsiMskSolverInterface, 184
- T
- task_
 - OsiMskSolverInterface, 186
- TestOutcome
 - OsiUnitTest::TestOutcome, 323
- testcond
 - OsiUnitTest::TestOutcome, 323
- testingMessage
 - OsiUnitTest, 11
- testname
 - OsiUnitTest::TestOutcome, 323
- timeRemaining_
 - OsiBranchingInformation, 32
- trustStrongForBound
 - OsiChooseVariable, 49
- trustStrongForBound_
 - OsiChooseVariable, 52

- trustStrongForSolution
 - OsiChooseVariable, [50](#)
- trustStrongForSolution_
 - OsiChooseVariable, [52](#)
- tryCuts
 - OsiBabSolver, [27](#)
- U
- ub
 - OsiRowCut, [208](#)
- ub_
 - OsiRowCut, [211](#)
- ubs
 - OsiColCut, [55](#)
- ubs_
 - OsiColCut, [57](#)
- unmarkHotStart
 - OsiCpxSolverInterface, [68](#)
 - OsiGlpkSolverInterface, [106](#)
 - OsiGrbSolverInterface, [134](#)
 - OsiMskSolverInterface, [175](#)
 - OsiSolverInterface, [241](#)
 - OsiSpxSolverInterface, [286](#)
 - OsiXprSolverInterface, [308](#)
- up_
 - OsiIntegerBranchingObject, [157](#)
 - OsiLotsizeBranchingObject, [163](#)
- upChange
 - OsiChooseVariable, [48](#)
 - OsiHotInfo, [152](#)
- upChange_
 - OsiChooseVariable, [50](#)
- upEstimate
 - OsiLotsize, [160](#)
 - OsiObject, [193](#)
 - OsiSimpleInteger, [220](#)
 - OsiSOS, [274](#)
- upIterationCount
 - OsiHotInfo, [153](#)
- upNumber
 - OsiPseudoCosts, [204](#)
- upNumber_
 - OsiPseudoCosts, [205](#)
- upStatus
 - OsiHotInfo, [153](#)
- upTotalChange
 - OsiPseudoCosts, [204](#)
- upTotalChange_
 - OsiPseudoCosts, [205](#)
- updateAfter
 - OsiObject, [194](#)
- updateBefore
 - OsiObject, [193](#)
- updateInformation
 - OsiChooseVariable, [47](#)
 - OsiHotInfo, [152](#)
 - OsiPseudoCosts, [204](#)
- upper_
 - OsiBranchingInformation, [33](#)
- useful_
 - OsiChooseVariable, [51](#)
- usefulRegion_
 - OsiBranchingInformation, [34](#)
- V
- validateCuts
 - OsiRowCutDebugger, [215](#)
- value
 - OsiBranchingObject, [38](#)
- value_
 - OsiBranchingObject, [39](#)
- value_type
 - OsiCuts::const_iterator, [17](#)
- vartype_
 - OsiXprSolverInterface, [321](#)
- verbosity
 - OsiUnitTest, [12](#)
- version
 - OsiXprSolverInterface, [317](#)
- violated
 - OsiColCut, [56](#)
 - OsiCut, [86](#)
 - OsiRowCut, [210](#)
- W
- WARNING
 - OsiUnitTest::TestOutcome, [322](#)
- warmStart
 - OsiBabSolver, [27](#)
- way
 - OsiTwoWayBranchingObject, [297](#)
- weights
 - OsiSOS, [274](#)
- weights_
 - OsiSOS, [275](#)
- which
 - OsiSolverBranch, [223](#)
- whichObject
 - OsiHotInfo, [154](#)
- whichObject_
 - OsiHotInfo, [154](#)
- whichRow
 - OsiRowCut2, [212](#)
- whichRow_
 - OsiRowCut2, [213](#)
- whichWay
 - OsiObject, [193](#)
- whichWay_
 - OsiObject, [194](#)

- writeLp
 - OsiSolverInterface, [258](#)
- writeLpNative
 - OsiSolverInterface, [259](#)
- writeMps
 - OsiCpxSolverInterface, [76](#)
 - OsiGlpkSolverInterface, [114](#)
 - OsiGrbSolverInterface, [142](#)
 - OsiMskSolverInterface, [183](#)
 - OsiSolverInterface, [258](#)
 - OsiSpxSolverInterface, [293](#)
 - OsiXprSolverInterface, [316](#)
- writeMpsNative
 - OsiSolverInterface, [258](#)
- ws_
 - OsiSolverInterface, [268](#)
- X
- XPRSProb
 - OsiXprSolverInterface.hpp, [342](#)
- xprProbname_
 - OsiXprSolverInterface, [319](#)