

Vol  
trunk

Generated by Doxygen 1.8.5

Mon Oct 21 2013 19:05:12

## Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>4</b>
2.1	Class List . . . . .	5
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>5</b>
4.1	OsiVolSolverInterface::OsiVolMatrixOneMinusOne_ Class Reference . . . . .	5
4.1.1	Detailed Description . . . . .	6
4.1.2	Constructor & Destructor Documentation . . . . .	6
4.1.3	Member Function Documentation . . . . .	6
4.1.4	Member Data Documentation . . . . .	6
4.2	OsiVolSolverInterface Class Reference . . . . .	7
4.2.1	Detailed Description . . . . .	14
4.2.2	Constructor & Destructor Documentation . . . . .	14
4.2.3	Member Function Documentation . . . . .	15
4.2.4	Friends And Related Function Documentation . . . . .	26
4.2.5	Member Data Documentation . . . . .	26
4.3	VOL_alpha_factor Class Reference . . . . .	29
4.3.1	Detailed Description . . . . .	29
4.3.2	Constructor & Destructor Documentation . . . . .	29
4.3.3	Member Function Documentation . . . . .	29
4.3.4	Member Data Documentation . . . . .	29
4.4	VOL_dual Class Reference . . . . .	30
4.4.1	Detailed Description . . . . .	30
4.4.2	Constructor & Destructor Documentation . . . . .	30
4.4.3	Member Function Documentation . . . . .	30
4.4.4	Member Data Documentation . . . . .	31
4.5	VOL_dvector Class Reference . . . . .	31
4.5.1	Detailed Description . . . . .	32
4.5.2	Constructor & Destructor Documentation . . . . .	32
4.5.3	Member Function Documentation . . . . .	33
4.5.4	Member Data Documentation . . . . .	33
4.6	VOL_indc Class Reference . . . . .	34

4.6.1	Detailed Description	34
4.6.2	Constructor & Destructor Documentation	34
4.6.3	Member Function Documentation	35
4.6.4	Member Data Documentation	35
4.7	VOL_ivector Class Reference	35
4.7.1	Detailed Description	36
4.7.2	Constructor & Destructor Documentation	36
4.7.3	Member Function Documentation	37
4.7.4	Member Data Documentation	37
4.8	VOL_parms Struct Reference	38
4.8.1	Detailed Description	39
4.8.2	Member Data Documentation	39
4.9	VOL_primal Class Reference	41
4.9.1	Detailed Description	42
4.9.2	Constructor & Destructor Documentation	42
4.9.3	Member Function Documentation	42
4.9.4	Member Data Documentation	42
4.10	VOL_problem Class Reference	42
4.10.1	Detailed Description	44
4.10.2	Constructor & Destructor Documentation	44
4.10.3	Member Function Documentation	45
4.10.4	Member Data Documentation	46
4.11	VOL_swing Class Reference	47
4.11.1	Detailed Description	48
4.11.2	Member Enumeration Documentation	48
4.11.3	Constructor & Destructor Documentation	48
4.11.4	Member Function Documentation	48
4.11.5	Member Data Documentation	49
4.12	VOL_user_hooks Class Reference	49
4.12.1	Detailed Description	50
4.12.2	Constructor & Destructor Documentation	50
4.12.3	Member Function Documentation	50
4.13	VOL_vh Class Reference	51
4.13.1	Detailed Description	51
4.13.2	Constructor & Destructor Documentation	52
4.13.3	Member Function Documentation	52
4.13.4	Member Data Documentation	52

<b>5</b>	<b>File Documentation</b>	<b>52</b>
5.1	/home/ted/COIN/trunk/Vol/src/OsiVol/OsiVolSolverInterface.hpp File Reference . . . . .	52
5.1.1	Function Documentation . . . . .	53
5.1.2	Variable Documentation . . . . .	53
5.2	/home/ted/COIN/trunk/Vol/src/VolVolume.hpp File Reference . . . . .	53
5.2.1	Macro Definition Documentation . . . . .	54
5.2.2	Function Documentation . . . . .	54

<b>Index</b>	<b>55</b>
--------------	-----------

## 1 Hierarchical Index

### 1.1 Class Hierarchy

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

```

std::allocator< T >
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::list< T >::const_iterator
std::forward_list< T >::const_iterator
std::map< K, T >::const_iterator
std::unordered_map< K, T >::const_iterator
std::basic_string< Char >::const_iterator
std::unordered_multimap< K, T >::const_iterator
std::set< K >::const_iterator
std::string::const_iterator
std::unordered_set< K >::const_iterator
std::wstring::const_iterator
std::multiset< K >::const_iterator
std::unordered_multiset< K >::const_iterator
std::vector< T >::const_iterator
std::multimap< K, 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 >
- std::ios\_base
  - basic\_ios< char >
  - basic\_ios< wchar\_t >
  - std::basic\_ios
    - basic\_istream< char >
    - basic\_istream< wchar\_t >
    - basic\_ostream< char >
    - basic\_ostream< wchar\_t >
    - std::basic\_istream
      - basic\_ifstream< char >
      - basic\_ifstream< wchar\_t >
      - basic\_iostream< char >
      - basic\_iostream< wchar\_t >
      - basic\_istreamstream< char >
      - basic\_istreamstream< wchar\_t >
    - std::basic\_ifstream
      - std::ifstream
      - std::wifstream
    - std::basic\_iostream
      - basic\_fstream< char >
      - basic\_fstream< wchar\_t >
      - basic\_stringstream< char >
      - basic\_stringstream< wchar\_t >
    - std::basic\_fstream
      - std::fstream
      - std::wfstream
    - std::basic\_stringstream

```

        std::stringstream
        std::wstringstream
    std::basic_istream
        std::istream
        std::wistream
    std::basic_ostream
        basic_iostream< char >
        basic_iostream< wchar_t >
        basic_ofstream< char >
        basic_ofstream< wchar_t >
        basic_ostringstream< char >
        basic_ostringstream< wchar_t >
    std::basic_iostream
    std::basic_ofstream
        std::ofstream
        std::wofstream
    std::basic_ostringstream
        std::ostringstream
        std::wostringstream
    std::ostream
    std::wostream
    std::ios
    std::wios
    std::forward_list< T >::iterator
    std::list< T >::iterator
    std::map< K, T >::iterator
    std::unordered_map< K, T >::iterator
    std::multimap< K, T >::iterator
    std::unordered_multimap< K, T >::iterator
    std::set< K >::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::basic_string< Char >::iterator
    std::list< T >
    std::map< K, T >
    std::multimap< K, T >
    std::multiset< K >
    OsiSolverInterface

```

### OsiVolSolverInterface

7

#### OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_

5

```

    std::priority_queue< T >
    std::queue< T >
    std::list< T >::reverse_iterator
    std::wstring::reverse_iterator
    std::forward_list< T >::reverse_iterator

```

std::vector< T >::reverse\_iterator  
 std::unordered\_multiset< K >::reverse\_iterator  
 std::multimap< K, T >::reverse\_iterator  
 std::unordered\_map< K, T >::reverse\_iterator  
 std::basic\_string< Char >::reverse\_iterator  
 std::unordered\_multimap< K, T >::reverse\_iterator  
 std::multiset< K >::reverse\_iterator  
 std::set< K >::reverse\_iterator  
 std::string::reverse\_iterator  
 std::unordered\_set< K >::reverse\_iterator  
 std::map< K, T >::reverse\_iterator  
 std::deque< T >::reverse\_iterator  
 std::set< K >  
 std::smart\_ptr< T >  
 std::stack< T >  
 std::system\_error  
 std::thread  
 std::unique\_ptr< T >  
 std::unordered\_map< K, T >  
 std::unordered\_multimap< K, T >  
 std::unordered\_multiset< K >  
 std::unordered\_set< K >  
 std::valarray< T >  
 std::vector< T >

<b>VOL_alpha_factor</b>	<b>29</b>
<b>VOL_dual</b>	<b>30</b>
<b>VOL_dvector</b>	<b>31</b>
<b>VOL_indc</b>	<b>34</b>
<b>VOL_ivector</b>	<b>35</b>
<b>VOL_parms</b>	<b>38</b>
<b>VOL_primal</b>	<b>41</b>
<b>VOL_problem</b>	<b>42</b>
<b>VOL_swing</b>	<b>47</b>
<b>VOL_user_hooks</b>	<b>49</b>
<b>OsiVolSolverInterface</b>	<b>7</b>
<b>VOL_vh</b>	<b>51</b>
std::weak_ptr< T >	
K	
T	

## 2 Class Index

## 2.1 Class List

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

<a href="#">OsiVolSolverInterface::OsiVolMatrixOneMinusOne_</a>	5
<a href="#">OsiVolSolverInterface</a> Vol(ume) Solver Interface	7
<a href="#">VOL_alpha_factor</a>	29
<a href="#">VOL_dual</a>	30
<a href="#">VOL_dvector</a> Vector of doubles	31
<a href="#">VOL_indc</a>	34
<a href="#">VOL_ivector</a> Vector of ints	35
<a href="#">VOL_parms</a> This class contains the parameters controlling the Volume Algorithm	38
<a href="#">VOL_primal</a>	41
<a href="#">VOL_problem</a> This class holds every data for the Volume Algorithm and its <code>solve</code> method must be invoked to solve the problem	42
<a href="#">VOL_swing</a>	47
<a href="#">VOL_user_hooks</a> The user hooks should be overridden by the user to provide the problem specific routines for the volume algorithm	49
<a href="#">VOL_vh</a>	51

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">/home/ted/COIN/trunk/Vol/src/VolVolume.hpp</a>	53
<a href="#">/home/ted/COIN/trunk/Vol/src/OsiVol/OsiVolSolverInterface.hpp</a>	52

## 4 Class Documentation

### 4.1 OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_ Class Reference



## Public Member Functions

- [OsiVolMatrixOneMinusOne\\_](#) (const CoinPackedMatrix &m)
- [~OsiVolMatrixOneMinusOne\\_](#) ()
- void [timesMajor](#) (const double \*x, double \*y) const

## Private Attributes

- int [majorDim\\_](#)
- int [minorDim\\_](#)
- int [plusSize\\_](#)
- int \* [plusInd\\_](#)
- int \* [plusStart\\_](#)
- int \* [plusLength\\_](#)
- int [minusSize\\_](#)
- int \* [minusInd\\_](#)
- int \* [minusStart\\_](#)
- int \* [minusLength\\_](#)

### 4.1.1 Detailed Description

Definition at line 32 of file OsiVolSolverInterface.hpp.

### 4.1.2 Constructor & Destructor Documentation

4.1.2.1 `OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::OsiVolMatrixOneMinusOne_ ( const CoinPackedMatrix & m )`

4.1.2.2 `OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::~~OsiVolMatrixOneMinusOne_ ( )`

### 4.1.3 Member Function Documentation

4.1.3.1 `void OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::timesMajor ( const double * x, double * y ) const`

### 4.1.4 Member Data Documentation

4.1.4.1 `int OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::majorDim_ [private]`

Definition at line 33 of file OsiVolSolverInterface.hpp.

4.1.4.2 `int OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::minorDim_ [private]`

Definition at line 34 of file OsiVolSolverInterface.hpp.

4.1.4.3 `int OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::plusSize_ [private]`

Definition at line 36 of file OsiVolSolverInterface.hpp.

4.1.4.4 `int* OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::plusInd_ [private]`

Definition at line 37 of file OsiVolSolverInterface.hpp.

4.1.4.5 `int* OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::plusStart_` [private]

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

4.1.4.6 `int* OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::plusLength_` [private]

Definition at line 39 of file `OsiVolSolverInterface.hpp`.

4.1.4.7 `int OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::minusSize_` [private]

Definition at line 41 of file `OsiVolSolverInterface.hpp`.

4.1.4.8 `int* OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::minusInd_` [private]

Definition at line 42 of file `OsiVolSolverInterface.hpp`.

4.1.4.9 `int* OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::minusStart_` [private]

Definition at line 43 of file `OsiVolSolverInterface.hpp`.

4.1.4.10 `int* OsiVolSolverInterface::OsiVolMatrixOneMinusOne_::minusLength_` [private]

Definition at line 44 of file `OsiVolSolverInterface.hpp`.

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

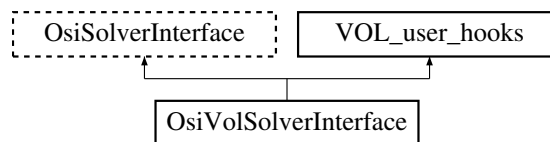
- </home/ted/COIN/trunk/Vol/src/OsiVol/OsiVolSolverInterface.hpp>

## 4.2 OsiVolSolverInterface Class Reference

Vol(ume) Solver Interface.

```
#include <OsiVolSolverInterface.hpp>
```

Inheritance diagram for `OsiVolSolverInterface`:



### Classes

- class [OsiVolMatrixOneMinusOne\\_](#)

### 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)
- bool [setDbParam](#) (OsiDbParam key, double value)
- bool [setStrParam](#) (OsiStrParam key, const std::string &value)
- bool [getIntParam](#) (OsiIntParam key, int &value) const
- bool [getDbParam](#) (OsiDbParam key, double &value) const
- bool [getStrParam](#) (OsiStrParam key, std::string &value) 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

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

**Hotstart related methods (primarily used in strong branching). <br>**

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

**NOTE:** between hotstarted optimizations only bound changes are allowed.

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

**Methods related to querying the input data**

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

**Methods related to querying the solution**

- virtual const double \* [getColSolution](#) () const  
Get pointer to array[getNumCols()] of primal solution vector.
- virtual const double \* [getRowPrice](#) () const  
Get pointer to array[getNumRows()] of dual prices.
- virtual const double \* [getReducedCost](#) () const

- *Get a pointer to array[getNumCols()] of reduced costs.*
- virtual const double \* [getRowActivity](#) () const  
*Get pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector).*
- virtual double [getObjValue](#) () const  
*Get objective function value.*
- virtual int [getIterationCount](#) () const  
*Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver).*
- virtual std::vector< double \* > [getDualRays](#) (int maxNumRays, bool fullRay=false) const  
*Get as many dual rays as the solver can provide.*
- virtual std::vector< double \* > [getPrimalRays](#) (int maxNumRays) const  
*Get as many primal rays as the solver can provide.*

### Changing bounds on variables and constraints

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

### Integrality related changing methods

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

**Methods to expand a problem.<br>**

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

- virtual void [addCol](#) (const CoinPackedVectorBase &vec, const double collb, const double colub, const double obj)
- virtual void [addCols](#) (const int numcols, const CoinPackedVectorBase \*const \*cols, const double \*collb, const double \*colub, const double \*obj)
- virtual void [deleteCols](#) (const int num, const int \*colIndices)
- virtual void [addRow](#) (const CoinPackedVectorBase &vec, const double rowlb, const double rowub)
- virtual void [addRow](#) (const CoinPackedVectorBase &vec, const char rowsen, const double rowrhs, const double rowrng)
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase \*const \*rows, const double \*rowlb, const double \*rowub)
- virtual void [addRows](#) (const int numRows, const CoinPackedVectorBase \*const \*rows, const char \*rowsen, const double \*rowrhs, const double \*rowrng)
- virtual void [deleteRows](#) (const int num, const int \*rowIndices)

**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.*

**OSL specific public interfaces**

- [VOL\\_problem](#) \* [volprob](#) ()  
*Get pointer to Vol model.*

**Constructors and destructors**

- [OsiVolSolverInterface](#) ()

- *Default Constructor.*
- virtual `OsiSolverInterface * clone` (bool copyData=true) const  
*Clone.*
- `OsiVolSolverInterface` (const `OsiVolSolverInterface &`)  
*Copy constructor.*
- `OsiVolSolverInterface & operator=` (const `OsiVolSolverInterface &rhs`)  
*Assignment operator.*
- virtual `~OsiVolSolverInterface` ()  
*Destructor.*

## Protected Member Functions

### Helper methods for problem input

- void `initFromRlbRub` (const int rownum, const double \*rowlb, const double \*rowub)
- void `initFromRhsSenseRange` (const int rownum, const char \*rowSEN, const double \*rowrhs, const double \*rowrng)
- void `initFromClibCubObj` (const int colnum, const double \*collb, const double \*colub, const double \*obj)

### Protected methods

- virtual void `applyRowCut` (const `OsiRowCut &rc`)  
*Apply a row cut (append to constraint matrix).*
- virtual void `applyColCut` (const `OsiColCut &cc`)  
*Apply a column cut (adjust one or more bounds).*

## Private Member Functions

### Methods of `<code>VOL_user_hooks</code>`

- virtual int `compute_rc` (const `VOL_dvector &u`, `VOL_dvector &rc`)  
*compute reduced costs*
- virtual int `solve_subproblem` (const `VOL_dvector &dual`, const `VOL_dvector &rc`, double &lcost, `VOL_dvector &x`, `VOL_dvector &v`, double &pcost)  
*Solve the subproblem for the subgradient step.*
- virtual int `heuristics` (const `VOL_problem &`, const `VOL_dvector &`, double &heur\_val)  
*Starting from the primal vector x, run a heuristic to produce an integer solution.*

### Private helper methods

- void `updateRowMatrix_` () const  
*Update the row ordered matrix from the column ordered one.*
- void `updateColMatrix_` () const  
*Update the column ordered matrix from the row ordered one.*
- void `checkData_` () const  
*Test whether the Volume Algorithm can be applied to the given problem.*
- void `compute_rc_` (const double \*u, double \*rc) const  
*Compute the reduced costs ( $\pi C$ ) with respect to the dual values given in u.*
- void `gutsOfDestructor_` ()  
*A method deleting every member data.*
- void `rowRimAllocator_` ()  
*A method allocating sufficient space for the rim vectors corresponding to the rows.*
- void `colRimAllocator_` ()

- *A method allocating sufficient space for the rim vectors corresponding to the columns.*
- void [rowRimResize\\_](#) (const int newSize)  
*Reallocate the rim arrays corresponding to the rows.*
- void [colRimResize\\_](#) (const int newSize)  
*Reallocate the rim arrays corresponding to the columns.*
- void [convertBoundsToSenses\\_](#) ()  
*For each row convert LB/UB style row constraints to sense/rhs style.*
- void [convertSensesToBounds\\_](#) ()  
*For each row convert sense/rhs style row constraints to LB/UB style.*
- bool [test\\_zero\\_one\\_minusone\\_](#) (const CoinPackedMatrix &m) const  
*test whether the given matrix is 0/1/-1 entries only.*

#### Private Attributes

- double [objsense\\_](#)  
*Sense of objective (1 for min; -1 for max)*
- double \* [rowpriceHotStart\\_](#)  
*An array to store the hotstart information between solveHotStart() calls.*
- int [maxNumrows\\_](#)  
*allocated size of the row related rim vectors*
- int [maxNumcols\\_](#)  
*allocated size of the column related rim vectors*
- [VOL\\_problem](#) [volprob\\_](#)  
*The volume solver.*

#### The problem matrix in row and column ordered forms <br>

*Note that at least one of the matrices is always current.*

- bool [rowMatrixCurrent\\_](#)  
*A flag indicating whether the row ordered matrix is up-to-date.*
- CoinPackedMatrix [rowMatrix\\_](#)  
*The problem matrix in a row ordered form.*
- bool [colMatrixCurrent\\_](#)  
*A flag indicating whether the column ordered matrix is up-to-date.*
- CoinPackedMatrix [colMatrix\\_](#)  
*The problem matrix in a column ordered form.*

#### Data members used when 0/1/-1 matrix is detected

- bool [isZeroOneMinusOne\\_](#)  
*An indicator whether the matrix is 0/1/-1.*
- [OsiVolMatrixOneMinusOne\\_](#) \* [rowMatrixOneMinusOne\\_](#)  
*The row ordered matrix without the elements.*
- [OsiVolMatrixOneMinusOne\\_](#) \* [colMatrixOneMinusOne\\_](#)  
*The column ordered matrix without the elements.*

#### The rim vectors

- double \* [colupper\\_](#)  
*Pointer to dense vector of structural variable upper bounds.*
- double \* [collower\\_](#)  
*Pointer to dense vector of structural variable lower bounds.*



- `bool * continuous_`  
*Pointer to dense vector of bool to indicate if column is continuous.*
- `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 * objcoffs_`  
*Pointer to dense vector of objective coefficients.*

### The solution

- `double * colsol_`  
*Pointer to dense vector of primal structural variable values.*
- `double * rowprice_`  
*Pointer to dense vector of dual row variable values.*
- `double * rc_`  
*Pointer to dense vector of reduced costs.*
- `double * lhs_`  
*Pointer to dense vector of left hand sides (row activity levels)*
- `double lagrangeanCost_`  
*The Lagrangean cost, a lower bound on the objective value.*

### Friends

- `void OsiVolSolverInterfaceUnitTest` (const std::string &mpsDir, const std::string &netlibDir)  
*A function that tests the methods in the [OsiVolSolverInterface](#) class.*

## 4.2.1 Detailed Description

Vol(ume) Solver Interface.

Instantiation of [OsiVolSolverInterface](#) for the Volume Algorithm

Definition at line 27 of file `OsiVolSolverInterface.hpp`.

## 4.2.2 Constructor & Destructor Documentation

### 4.2.2.1 `OsiVolSolverInterface::OsiVolSolverInterface ( )`

Default Constructor.

### 4.2.2.2 `OsiVolSolverInterface::OsiVolSolverInterface ( const OsiVolSolverInterface & )`

Copy constructor.

### 4.2.2.3 `virtual OsiVolSolverInterface::~~OsiVolSolverInterface ( ) [virtual]`

Destructor.

## 4.2.3 Member Function Documentation

## 4.2.3.1 virtual void OsiVolSolverInterface::initialSolve ( ) [virtual]

Solve initial LP relaxation.

## 4.2.3.2 virtual void OsiVolSolverInterface::resolve ( ) [virtual]

Resolve an LP relaxation after problem modification.

## 4.2.3.3 virtual void OsiVolSolverInterface::branchAndBound ( ) [inline],[virtual]

Invoke solver's built-in enumeration algorithm.

Definition at line 63 of file OsiVolSolverInterface.hpp.

4.2.3.4 bool OsiVolSolverInterface::setIntParam ( OsiIntParam *key*, int *value* )4.2.3.5 bool OsiVolSolverInterface::setDbiParam ( OsiDbiParam *key*, double *value* )4.2.3.6 bool OsiVolSolverInterface::setStrParam ( OsiStrParam *key*, const std::string & *value* )4.2.3.7 bool OsiVolSolverInterface::getIntParam ( OsiIntParam *key*, int & *value* ) const4.2.3.8 bool OsiVolSolverInterface::getDbiParam ( OsiDbiParam *key*, double & *value* ) const4.2.3.9 bool OsiVolSolverInterface::getStrParam ( OsiStrParam *key*, std::string & *value* ) const

## 4.2.3.10 virtual bool OsiVolSolverInterface::isAbandoned ( ) const [virtual]

Are there a numerical difficulties?

## 4.2.3.11 virtual bool OsiVolSolverInterface::isProvenOptimal ( ) const [virtual]

Is optimality proven?

## 4.2.3.12 virtual bool OsiVolSolverInterface::isProvenPrimalInfeasible ( ) const [virtual]

Is primal infeasibility proven?

## 4.2.3.13 virtual bool OsiVolSolverInterface::isProvenDualInfeasible ( ) const [virtual]

Is dual infeasibility proven?

## 4.2.3.14 virtual bool OsiVolSolverInterface::isPrimalObjectiveLimitReached ( ) const [virtual]

Is the given primal objective limit reached?

## 4.2.3.15 virtual bool OsiVolSolverInterface::isDualObjectiveLimitReached ( ) const [virtual]

Is the given dual objective limit reached?

## 4.2.3.16 virtual bool OsiVolSolverInterface::isIterationLimitReached ( ) const [virtual]

Iteration limit reached?

4.2.3.17 `virtual CoinWarmStart* OsiVolSolverInterface::getEmptyWarmStart ( ) const [virtual]`

Get an empty warm start object.

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

4.2.3.18 `virtual CoinWarmStart* OsiVolSolverInterface::getWarmStart ( ) const [virtual]`

Get warmstarting information.

4.2.3.19 `virtual bool OsiVolSolverInterface::setWarmStart ( const CoinWarmStart * warmstart ) [virtual]`

Set warmstarting information.

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

4.2.3.20 `virtual void OsiVolSolverInterface::markHotStart ( ) [virtual]`

Create a hotstart point of the optimization process.

4.2.3.21 `virtual void OsiVolSolverInterface::solveFromHotStart ( ) [virtual]`

Optimize starting from the hotstart.

4.2.3.22 `virtual void OsiVolSolverInterface::unmarkHotStart ( ) [virtual]`

Delete the snapshot.

4.2.3.23 `virtual int OsiVolSolverInterface::getNumCols ( ) const [inline],[virtual]`

Get number of columns.

Definition at line 167 of file `OsiVolSolverInterface.hpp`.

4.2.3.24 `virtual int OsiVolSolverInterface::getNumRows ( ) const [inline],[virtual]`

Get number of rows.

Definition at line 172 of file `OsiVolSolverInterface.hpp`.

4.2.3.25 `virtual int OsiVolSolverInterface::getNumElements ( ) const [inline],[virtual]`

Get number of nonzero elements.

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

4.2.3.26 `virtual const double* OsiVolSolverInterface::getColLower ( ) const [inline],[virtual]`

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

Definition at line 182 of file `OsiVolSolverInterface.hpp`.

4.2.3.27 `virtual const double* OsiVolSolverInterface::getColUpper ( ) const [inline],[virtual]`

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

Definition at line 185 of file `OsiVolSolverInterface.hpp`.

**4.2.3.28** `virtual const char* OsiVolSolverInterface::getRowSense ( ) const [inline],[virtual]`

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

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

Definition at line 196 of file OsiVolSolverInterface.hpp.

**4.2.3.29** `virtual const double* OsiVolSolverInterface::getRightHandSide ( ) const [inline],[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

Definition at line 206 of file OsiVolSolverInterface.hpp.

**4.2.3.30** `virtual const double* OsiVolSolverInterface::getRowRange ( ) const [inline],[virtual]`

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

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

Definition at line 216 of file OsiVolSolverInterface.hpp.

**4.2.3.31** `virtual const double* OsiVolSolverInterface::getRowLower ( ) const [inline],[virtual]`

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

Definition at line 219 of file OsiVolSolverInterface.hpp.

**4.2.3.32** `virtual const double* OsiVolSolverInterface::getRowUpper ( ) const [inline],[virtual]`

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

Definition at line 222 of file OsiVolSolverInterface.hpp.

**4.2.3.33** `virtual const double* OsiVolSolverInterface::getObjCoefficients ( ) const [inline],[virtual]`

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

Definition at line 225 of file OsiVolSolverInterface.hpp.

**4.2.3.34** `virtual double OsiVolSolverInterface::getObjSense ( ) const [inline],[virtual]`

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

Definition at line 228 of file OsiVolSolverInterface.hpp.

**4.2.3.35** `virtual bool OsiVolSolverInterface::isContinuous ( int colNumber ) const [virtual]`

Return true if column is continuous.

**4.2.3.36** `virtual const CoinPackedMatrix* OsiVolSolverInterface::getMatrixByRow ( ) const [virtual]`

Get pointer to row-wise copy of matrix.

**4.2.3.37** `virtual const CoinPackedMatrix* OsiVolSolverInterface::getMatrixByCol ( ) const [virtual]`

Get pointer to column-wise copy of matrix.

**4.2.3.38** `virtual double OsiVolSolverInterface::getInfinity ( ) const [inline],[virtual]`

Get solver's value for infinity.

Definition at line 257 of file OsiVolSolverInterface.hpp.

**4.2.3.39** `virtual const double* OsiVolSolverInterface::getColSolution ( ) const [inline],[virtual]`

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

Definition at line 263 of file OsiVolSolverInterface.hpp.

**4.2.3.40** `virtual const double* OsiVolSolverInterface::getRowPrice ( ) const [inline],[virtual]`

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

Definition at line 266 of file OsiVolSolverInterface.hpp.

**4.2.3.41** `virtual const double* OsiVolSolverInterface::getReducedCost ( ) const [inline],[virtual]`

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

Definition at line 269 of file OsiVolSolverInterface.hpp.

**4.2.3.42** `virtual const double* OsiVolSolverInterface::getRowActivity ( ) const [inline],[virtual]`

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

Definition at line 273 of file OsiVolSolverInterface.hpp.

**4.2.3.43** `virtual double OsiVolSolverInterface::getObjValue ( ) const [inline],[virtual]`

Get objective function value.

Definition at line 276 of file OsiVolSolverInterface.hpp.

**4.2.3.44** `virtual int OsiVolSolverInterface::getIterationCount ( ) const [inline],[virtual]`

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

Definition at line 287 of file OsiVolSolverInterface.hpp.

4.2.3.45 `virtual std::vector<double*> OsiVolSolverInterface::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[]`.

4.2.3.46 `virtual std::vector<double*> OsiVolSolverInterface::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[]`.

4.2.3.47 `virtual void OsiVolSolverInterface::setObjCoeff ( int elementIndex, double elementValue )` `[inline],[virtual]`

Set an objective function coefficient.

Definition at line 338 of file `OsiVolSolverInterface.hpp`.

4.2.3.48 `virtual void OsiVolSolverInterface::setColLower ( int elementIndex, double elementValue )` `[inline],[virtual]`

Set a single column lower bound

Use `-DBL_MAX` for -infinity.

Definition at line 345 of file `OsiVolSolverInterface.hpp`.

4.2.3.49 `virtual void OsiVolSolverInterface::setColUpper ( int elementIndex, double elementValue )` `[inline],[virtual]`

Set a single column upper bound

Use `DBL_MAX` for infinity.

Definition at line 352 of file `OsiVolSolverInterface.hpp`.

4.2.3.50 `virtual void OsiVolSolverInterface::setColBounds ( int elementIndex, double lower, double upper )` `[inline],[virtual]`

Set a single column lower and upper bound.

Definition at line 357 of file `OsiVolSolverInterface.hpp`.

**4.2.3.51** `virtual void OsiVolSolverInterface::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

**4.2.3.52** `virtual void OsiVolSolverInterface::setRowLower ( int elementIndex, double elementValue ) [inline],[virtual]`

Set a single row lower bound

Use -DBL\_MAX for -infinity.

Definition at line 377 of file OsiVolSolverInterface.hpp.

**4.2.3.53** `virtual void OsiVolSolverInterface::setRowUpper ( int elementIndex, double elementValue ) [inline],[virtual]`

Set a single row upper bound

Use DBL\_MAX for infinity.

Definition at line 386 of file OsiVolSolverInterface.hpp.

**4.2.3.54** `virtual void OsiVolSolverInterface::setRowBounds ( int elementIndex, double lower, double upper ) [inline],[virtual]`

Set a single row lower and upper bound.

Definition at line 394 of file OsiVolSolverInterface.hpp.

**4.2.3.55** `virtual void OsiVolSolverInterface::setRowType ( int index, char sense, double rightHandSide, double range ) [inline],[virtual]`

Set the type of a single row

Definition at line 404 of file OsiVolSolverInterface.hpp.

**4.2.3.56** `virtual void OsiVolSolverInterface::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

**4.2.3.57** `virtual void OsiVolSolverInterface::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

4.2.3.58 `virtual void OsiVolSolverInterface::setContinuous ( int index ) [virtual]`

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

4.2.3.59 `virtual void OsiVolSolverInterface::setInteger ( int index ) [virtual]`

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

4.2.3.60 `virtual void OsiVolSolverInterface::setContinuous ( const int * indices, int len ) [virtual]`

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

4.2.3.61 `virtual void OsiVolSolverInterface::setInteger ( const int * indices, int len ) [virtual]`

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

4.2.3.62 `virtual void OsiVolSolverInterface::setObjSense ( double s ) [inline],[virtual]`

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

Definition at line 459 of file OsiVolSolverInterface.hpp.

4.2.3.63 `virtual void OsiVolSolverInterface::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.

4.2.3.64 `virtual void OsiVolSolverInterface::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.

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

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

4.2.3.67 `virtual void OsiVolSolverInterface::deleteCols ( const int num, const int * colIndices ) [virtual]`

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

- 4.2.3.69 `virtual void OsiVolSolverInterface::addRow ( const CoinPackedVectorBase & vec, const char rowSEN, const double rowRHS, const double rowrng )` [virtual]
- 4.2.3.70 `virtual void OsiVolSolverInterface::addRows ( const int numrows, const CoinPackedVectorBase *const * rows, const double * rowlb, const double * rowub )` [virtual]
- 4.2.3.71 `virtual void OsiVolSolverInterface::addRows ( const int numrows, const CoinPackedVectorBase *const * rows, const char * rowSEN, const double * rowRHS, const double * rowrng )` [virtual]
- 4.2.3.72 `virtual void OsiVolSolverInterface::deleteRows ( const int num, const int * rowIndices )` [virtual]
- 4.2.3.73 `void OsiVolSolverInterface::initFromRlbRub ( const int rownum, const double * rowlb, const double * rowub )` [protected]
- 4.2.3.74 `void OsiVolSolverInterface::initFromRhsSenseRange ( const int rownum, const char * rowSEN, const double * rowRHS, const double * rowrng )` [protected]
- 4.2.3.75 `void OsiVolSolverInterface::initFromClibCubObj ( const int colnum, const double * collb, const double * colub, const double * obj )` [protected]
- 4.2.3.76 `virtual void OsiVolSolverInterface::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

- 4.2.3.77 `virtual void OsiVolSolverInterface::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.

- 4.2.3.78 `virtual void OsiVolSolverInterface::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

**4.2.3.79** `virtual void OsiVolSolverInterface::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.

**4.2.3.80** `virtual void OsiVolSolverInterface::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).

**4.2.3.81** `virtual void OsiVolSolverInterface::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).

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

Read an mps file from the given filename.

**4.2.3.83** `virtual void OsiVolSolverInterface::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

**4.2.3.84** `VOL_problem* OsiVolSolverInterface::volprob ( ) [inline]`

Get pointer to Vol model.

Definition at line 681 of file `OsiVolSolverInterface.hpp`.

**4.2.3.85** `virtual OsiSolverInterface* OsiVolSolverInterface::clone ( bool copyData = true ) const [virtual]`

Clone.

**4.2.3.86** `OsiVolSolverInterface& OsiVolSolverInterface::operator= ( const OsiVolSolverInterface & rhs )`

Assignment operator.

**4.2.3.87** `virtual void OsiVolSolverInterface::applyRowCut ( const OsiRowCut & rc ) [protected],[virtual]`

Apply a row cut (append to constraint matrix).

4.2.3.88 `virtual void OsiVolSolverInterface::applyColCut ( const OsiColCut & cc ) [protected],[virtual]`

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

4.2.3.89 `virtual int OsiVolSolverInterface::compute_rc ( const VOL_dvector & u, VOL_dvector & rc ) [private],[virtual]`

compute reduced costs

Implements [VOL\\_user\\_hooks](#).

4.2.3.90 `virtual int OsiVolSolverInterface::solve_subproblem ( const VOL_dvector & dual, const VOL_dvector & rc, double & lcost, VOL_dvector & x, VOL_dvector & v, double & pcost ) [private],[virtual]`

Solve the subproblem for the subgradient step.

Implements [VOL\\_user\\_hooks](#).

4.2.3.91 `virtual int OsiVolSolverInterface::heuristics ( const VOL_problem &, const VOL_dvector &, double & heur_val ) [inline],[private],[virtual]`

Starting from the primal vector x, run a heuristic to produce an integer solution.

This is not done in LP solving.

Implements [VOL\\_user\\_hooks](#).

Definition at line 730 of file OsiVolSolverInterface.hpp.

4.2.3.92 `void OsiVolSolverInterface::updateRowMatrix_ ( ) const [private]`

Update the row ordered matrix from the column ordered one.

4.2.3.93 `void OsiVolSolverInterface::updateColMatrix_ ( ) const [private]`

Update the column ordered matrix from the row ordered one.

4.2.3.94 `void OsiVolSolverInterface::checkData_ ( ) const [private]`

Test whether the Volume Algorithm can be applied to the given problem.

4.2.3.95 `void OsiVolSolverInterface::compute_rc_ ( const double * u, double * rc ) const [private]`

Compute the reduced costs (rc) with respect to the dual values given in u.

4.2.3.96 `void OsiVolSolverInterface::gutsOfDestructor_ ( ) [private]`

A method deleting every member data.

4.2.3.97 `void OsiVolSolverInterface::rowRimAllocator_ ( ) [private]`

A method allocating sufficient space for the rim vectors corresponding to the rows.

4.2.3.98 `void OsiVolSolverInterface::colRimAllocator_ ( ) [private]`

A method allocating sufficient space for the rim vectors corresponding to the columns.

4.2.3.99 `void OsiVolSolverInterface::rowRimResize_ ( const int newSize ) [private]`

Reallocate the rim arrays corresponding to the rows.

4.2.3.100 void OsiVolSolverInterface::colRimResize\_( const int *newSize* ) [private]

Reallocate the rim arrays corresponding to the columns.

4.2.3.101 void OsiVolSolverInterface::convertBoundsToSenses\_( ) [private]

For each row convert LB/UB style row constraints to sense/rhs style.

4.2.3.102 void OsiVolSolverInterface::convertSensesToBounds\_( ) [private]

For each row convert sense/rhs style row constraints to LB/UB style.

4.2.3.103 bool OsiVolSolverInterface::test\_zero\_one\_minusone\_( const CoinPackedMatrix & *m* ) const [private]

test whether the given matrix is 0/1/-1 entries only.

#### 4.2.4 Friends And Related Function Documentation

4.2.4.1 void OsiVolSolverInterfaceUnitTest( const std::string & *mpsDir*, const std::string & *netlibDir* ) [friend]

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

#### 4.2.5 Member Data Documentation

4.2.5.1 bool OsiVolSolverInterface::rowMatrixCurrent\_ [mutable],[private]

A flag indicating whether the row ordered matrix is up-to-date.

Definition at line 786 of file OsiVolSolverInterface.hpp.

4.2.5.2 CoinPackedMatrix OsiVolSolverInterface::rowMatrix\_ [mutable],[private]

The problem matrix in a row ordered form.

Definition at line 788 of file OsiVolSolverInterface.hpp.

4.2.5.3 bool OsiVolSolverInterface::colMatrixCurrent\_ [mutable],[private]

A flag indicating whether the column ordered matrix is up-to-date.

Definition at line 790 of file OsiVolSolverInterface.hpp.

4.2.5.4 CoinPackedMatrix OsiVolSolverInterface::colMatrix\_ [mutable],[private]

The problem matrix in a column ordered form.

Definition at line 792 of file OsiVolSolverInterface.hpp.

4.2.5.5 bool OsiVolSolverInterface::isZeroOneMinusOne\_ [private]

An indicator whether the matrix is 0/1/-1.

Definition at line 799 of file OsiVolSolverInterface.hpp.

4.2.5.6 OsiVolMatrixOneMinusOne\_\* OsiVolSolverInterface::rowMatrixOneMinusOne\_ [private]

The row ordered matrix without the elements.

Definition at line 801 of file OsiVolSolverInterface.hpp.

#### 4.2.5.7 OsiVolMatrixOneMinusOne\_\* OsiVolSolverInterface::colMatrixOneMinusOne\_ [private]

The column ordered matrix without the elements.

Definition at line 803 of file OsiVolSolverInterface.hpp.

#### 4.2.5.8 double\* OsiVolSolverInterface::colupper\_ [private]

Pointer to dense vector of structural variable upper bounds.

Definition at line 810 of file OsiVolSolverInterface.hpp.

#### 4.2.5.9 double\* OsiVolSolverInterface::collower\_ [private]

Pointer to dense vector of structural variable lower bounds.

Definition at line 812 of file OsiVolSolverInterface.hpp.

#### 4.2.5.10 bool\* OsiVolSolverInterface::continuous\_ [private]

Pointer to dense vector of bool to indicate if column is continuous.

Definition at line 814 of file OsiVolSolverInterface.hpp.

#### 4.2.5.11 double\* OsiVolSolverInterface::rowupper\_ [private]

Pointer to dense vector of slack variable upper bounds.

Definition at line 816 of file OsiVolSolverInterface.hpp.

#### 4.2.5.12 double\* OsiVolSolverInterface::rowlower\_ [private]

Pointer to dense vector of slack variable lower bounds.

Definition at line 818 of file OsiVolSolverInterface.hpp.

#### 4.2.5.13 char\* OsiVolSolverInterface::rowsense\_ [private]

Pointer to dense vector of row sense indicators.

Definition at line 820 of file OsiVolSolverInterface.hpp.

#### 4.2.5.14 double\* OsiVolSolverInterface::rhs\_ [private]

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

Definition at line 822 of file OsiVolSolverInterface.hpp.

#### 4.2.5.15 double\* OsiVolSolverInterface::rowrange\_ [private]

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

Definition at line 825 of file OsiVolSolverInterface.hpp.

#### 4.2.5.16 double\* OsiVolSolverInterface::objcoeffs\_ [private]

Pointer to dense vector of objective coefficients.

Definition at line 827 of file OsiVolSolverInterface.hpp.

**4.2.5.17 double OsiVolSolverInterface::objsense\_ [private]**

Sense of objective (1 for min; -1 for max)

Definition at line 832 of file OsiVolSolverInterface.hpp.

**4.2.5.18 double\* OsiVolSolverInterface::colsol\_ [private]**

Pointer to dense vector of primal structural variable values.

Definition at line 838 of file OsiVolSolverInterface.hpp.

**4.2.5.19 double\* OsiVolSolverInterface::rowprice\_ [private]**

Pointer to dense vector of dual row variable values.

Definition at line 840 of file OsiVolSolverInterface.hpp.

**4.2.5.20 double\* OsiVolSolverInterface::rc\_ [private]**

Pointer to dense vector of reduced costs.

Definition at line 842 of file OsiVolSolverInterface.hpp.

**4.2.5.21 double\* OsiVolSolverInterface::lhs\_ [private]**

Pointer to dense vector of left hand sides (row activity levels)

Definition at line 844 of file OsiVolSolverInterface.hpp.

**4.2.5.22 double OsiVolSolverInterface::lagrangeanCost\_ [private]**

The Lagrangean cost, a lower bound on the objective value.

Definition at line 846 of file OsiVolSolverInterface.hpp.

**4.2.5.23 double\* OsiVolSolverInterface::rowpriceHotStart\_ [private]**

An array to store the hotstart information between solveHotStart() calls.

Definition at line 852 of file OsiVolSolverInterface.hpp.

**4.2.5.24 int OsiVolSolverInterface::maxNumrows\_ [private]**

allocated size of the row related rim vectors

Definition at line 855 of file OsiVolSolverInterface.hpp.

**4.2.5.25 int OsiVolSolverInterface::maxNumcols\_ [private]**

allocated size of the column related rim vectors

Definition at line 857 of file OsiVolSolverInterface.hpp.

**4.2.5.26 VOL\_problem OsiVolSolverInterface::volprob\_ [private]**

The volume solver.

Definition at line 860 of file OsiVolSolverInterface.hpp.

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

- /home/ted/COIN/trunk/Vol/src/OsiVol/[OsiVolSolverInterface.hpp](#)

## 4.3 VOL\_alpha\_factor Class Reference

```
#include <VolVolume.hpp>
```

### Public Member Functions

- [VOL\\_alpha\\_factor](#) ()
- [~VOL\\_alpha\\_factor](#) ()
- double [factor](#) (const [VOL\\_parms](#) &parm, const double lcost, const double alpha)

### Public Attributes

- double [lastvalue](#)

### Private Member Functions

- [VOL\\_alpha\\_factor](#) (const [VOL\\_alpha\\_factor](#) &)
- [VOL\\_alpha\\_factor](#) & [operator=](#) (const [VOL\\_alpha\\_factor](#) &)

#### 4.3.1 Detailed Description

Definition at line 488 of file VolVolume.hpp.

#### 4.3.2 Constructor & Destructor Documentation

4.3.2.1 [VOL\\_alpha\\_factor::VOL\\_alpha\\_factor \( const VOL\\_alpha\\_factor & \)](#) [private]

4.3.2.2 [VOL\\_alpha\\_factor::VOL\\_alpha\\_factor \( \)](#) [inline]

Definition at line 495 of file VolVolume.hpp.

4.3.2.3 [VOL\\_alpha\\_factor::~~VOL\\_alpha\\_factor \( \)](#) [inline]

Definition at line 496 of file VolVolume.hpp.

#### 4.3.3 Member Function Documentation

4.3.3.1 [VOL\\_alpha\\_factor& VOL\\_alpha\\_factor::operator= \( const VOL\\_alpha\\_factor & \)](#) [private]

4.3.3.2 [double VOL\\_alpha\\_factor::factor \( const VOL\\_parms &parm, const double lcost, const double alpha \)](#) [inline]

Definition at line 498 of file VolVolume.hpp.

#### 4.3.4 Member Data Documentation

4.3.4.1 [double VOL\\_alpha\\_factor::lastvalue](#)

Definition at line 493 of file VolVolume.hpp.

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



- [/home/ted/COIN/trunk/Vol/src/VolVolume.hpp](#)

## 4.4 VOL\_dual Class Reference

```
#include <VolVolume.hpp>
```

### Public Member Functions

- [VOL\\_dual](#) (const int dsize)
- [VOL\\_dual](#) (const [VOL\\_dual](#) &dual)
- [~VOL\\_dual](#) ()
- [VOL\\_dual & operator=](#) (const [VOL\\_dual](#) &p)
- void [step](#) (const double target, const double lambda, const [VOL\\_dvector](#) &dual\_lb, const [VOL\\_dvector](#) &dual\_ub, const [VOL\\_dvector](#) &v)
- double [ascent](#) (const [VOL\\_dvector](#) &v, const [VOL\\_dvector](#) &last\_u) const
- void [compute\\_xrc](#) (const [VOL\\_dvector](#) &pstarx, const [VOL\\_dvector](#) &primalx, const [VOL\\_dvector](#) &rc)

### Public Attributes

- double [lcost](#)
- double [xrc](#)
- [VOL\\_dvector](#) [u](#)

#### 4.4.1 Detailed Description

Definition at line 353 of file VolVolume.hpp.

#### 4.4.2 Constructor & Destructor Documentation

##### 4.4.2.1 [VOL\\_dual::VOL\\_dual \( const int \*dsize\* \)](#) `[inline]`

Definition at line 363 of file VolVolume.hpp.

##### 4.4.2.2 [VOL\\_dual::VOL\\_dual \( const \[VOL\\\_dual\]\(#\) & \*dual\* \)](#) `[inline]`

Definition at line 364 of file VolVolume.hpp.

##### 4.4.2.3 [VOL\\_dual::~~VOL\\_dual \( \)](#) `[inline]`

Definition at line 366 of file VolVolume.hpp.

#### 4.4.3 Member Function Documentation

##### 4.4.3.1 [VOL\\_dual& VOL\\_dual::operator= \( const \[VOL\\\_dual\]\(#\) & \*p\* \)](#) `[inline]`

Definition at line 367 of file VolVolume.hpp.

4.4.3.2 void VOL\_dual::step ( const double *target*, const double *lambda*, const VOL\_dvector & *dual\_lb*, const VOL\_dvector & *dual\_ub*, const VOL\_dvector & *v* )

4.4.3.3 double VOL\_dual::ascent ( const VOL\_dvector & *v*, const VOL\_dvector & *last\_u* ) const

4.4.3.4 void VOL\_dual::compute\_xrc ( const VOL\_dvector & *pstarx*, const VOL\_dvector & *primalx*, const VOL\_dvector & *rc* )

#### 4.4.4 Member Data Documentation

4.4.4.1 double VOL\_dual::lcost

Definition at line 356 of file VolVolume.hpp.

4.4.4.2 double VOL\_dual::xrc

Definition at line 358 of file VolVolume.hpp.

4.4.4.3 VOL\_dvector VOL\_dual::u

Definition at line 361 of file VolVolume.hpp.

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

- /home/ted/COIN/trunk/Vol/src/VolVolume.hpp

## 4.5 VOL\_dvector Class Reference

vector of doubles.

```
#include <VolVolume.hpp>
```

### Public Member Functions

- [VOL\\_dvector](#) (const int s)  
*Construct a vector of size s.*
- [VOL\\_dvector](#) ()  
*Default constructor creates a vector of size 0.*
- [VOL\\_dvector](#) (const [VOL\\_dvector](#) &x)  
*Copy constructor makes a replica of x.*
- [~VOL\\_dvector](#) ()  
*The destructor deletes the data array.*
- int [size](#) () const  
*Return the size of the vector.*
- double & [operator\[\]](#) (const int i)  
*Return a reference to the i-th entry.*
- double [operator\[\]](#) (const int i) const  
*Return the i-th entry.*
- void [clear](#) ()  
*Delete the content of the vector and replace it with a vector of length 0.*
- void [cc](#) (const double gamma, const [VOL\\_dvector](#) &w)  
*Convex combination.*

- void `allocate` (const int s)  
*delete the current vector and allocate space for a vector of size s.*
- void `swap` (VOL\_dvector &w)  
*swaps the vector with w.*
- VOL\_dvector & `operator=` (const VOL\_dvector &w)  
*Copy w into the vector.*
- VOL\_dvector & `operator=` (const double w)  
*Replace every entry in the vector with w.*

#### Public Attributes

- double \* `v`  
*The array holding the vector.*
- int `sz`  
*The size of the vector.*

#### 4.5.1 Detailed Description

vector of doubles.

It is used for most vector operations.

Note: If `VOL_DEBUG` is `#defined` to be 1 then each time an entry is accessed in the vector the index of the entry is tested for nonnegativity and for being less than the size of the vector. It's good to turn this on while debugging, but in final runs it should be turned off (because of the performance hit).

Definition at line 147 of file `VolVolume.hpp`.

#### 4.5.2 Constructor & Destructor Documentation

##### 4.5.2.1 VOL\_dvector::VOL\_dvector ( const int s ) [inline]

Construct a vector of size s.

The content of the vector is undefined.

Definition at line 156 of file `VolVolume.hpp`.

##### 4.5.2.2 VOL\_dvector::VOL\_dvector ( ) [inline]

Default constructor creates a vector of size 0.

Definition at line 161 of file `VolVolume.hpp`.

##### 4.5.2.3 VOL\_dvector::VOL\_dvector ( const VOL\_dvector & x ) [inline]

Copy constructor makes a replica of x.

Definition at line 163 of file `VolVolume.hpp`.

##### 4.5.2.4 VOL\_dvector::~~VOL\_dvector ( ) [inline]

The destructor deletes the data array.

Definition at line 171 of file `VolVolume.hpp`.

### 4.5.3 Member Function Documentation

#### 4.5.3.1 int VOL\_dvector::size ( ) const [inline]

Return the size of the vector.

Definition at line 174 of file VolVolume.hpp.

#### 4.5.3.2 double& VOL\_dvector::operator[] ( const int i ) [inline]

Return a reference to the *i*-th entry.

Definition at line 177 of file VolVolume.hpp.

#### 4.5.3.3 double VOL\_dvector::operator[] ( const int i ) const [inline]

Return the *i*-th entry.

Definition at line 183 of file VolVolume.hpp.

#### 4.5.3.4 void VOL\_dvector::clear ( ) [inline]

Delete the content of the vector and replace it with a vector of length 0.

Definition at line 190 of file VolVolume.hpp.

#### 4.5.3.5 void VOL\_dvector::cc ( const double *gamma*, const VOL\_dvector & *w* ) [inline]

Convex combination.

Replace the current vector *v* with  $v = (1-\text{gamma}) v + \text{gamma } w$ .

Definition at line 197 of file VolVolume.hpp.

#### 4.5.3.6 void VOL\_dvector::allocate ( const int *s* ) [inline]

delete the current vector and allocate space for a vector of size *s*.

Definition at line 213 of file VolVolume.hpp.

#### 4.5.3.7 void VOL\_dvector::swap ( VOL\_dvector & *w* ) [inline]

swaps the vector with *w*.

Definition at line 220 of file VolVolume.hpp.

#### 4.5.3.8 VOL\_dvector& VOL\_dvector::operator= ( const VOL\_dvector & *w* )

Copy *w* into the vector.

#### 4.5.3.9 VOL\_dvector& VOL\_dvector::operator= ( const double *w* )

Replace every entry in the vector with *w*.

### 4.5.4 Member Data Documentation

#### 4.5.4.1 double\* VOL\_dvector::v

The array holding the vector.

Definition at line 150 of file VolVolume.hpp.

#### 4.5.4.2 int VOL\_dvector::sz

The size of the vector.

Definition at line 152 of file VolVolume.hpp.

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

- /home/ted/COIN/trunk/Vol/src/VolVolume.hpp

## 4.6 VOL\_indc Class Reference

```
#include <VolVolume.hpp>
```

### Public Member Functions

- [VOL\\_indc](#) (const [VOL\\_dvector](#) &dual\_lb, const [VOL\\_dvector](#) &dual\_ub, const [VOL\\_primal](#) &primal, const [VOL\\_-primal](#) &pstar, const [VOL\\_dual](#) &dual)
- [~VOL\\_indc](#) ()

### Public Attributes

- double [v2](#)
- double [vu](#)
- double [vabs](#)
- double [asc](#)

### Private Member Functions

- [VOL\\_indc](#) (const [VOL\\_indc](#) &)
- [VOL\\_indc](#) & [operator=](#) (const [VOL\\_indc](#) &)

#### 4.6.1 Detailed Description

Definition at line 537 of file VolVolume.hpp.

#### 4.6.2 Constructor & Destructor Documentation

4.6.2.1 [VOL\\_indc::VOL\\_indc](#) ( const [VOL\\_indc](#) & ) [private]

4.6.2.2 [VOL\\_indc::VOL\\_indc](#) ( const [VOL\\_dvector](#) & *dual\_lb*, const [VOL\\_dvector](#) & *dual\_ub*, const [VOL\\_primal](#) & *primal*, const [VOL\\_primal](#) & *pstar*, const [VOL\\_dual](#) & *dual* )

4.6.2.3 [VOL\\_indc::~VOL\\_indc](#) ( ) [inline]

Definition at line 551 of file VolVolume.hpp.

## 4.6.3 Member Function Documentation

## 4.6.3.1 VOL\_indc&amp; VOL\_indc::operator= ( const VOL\_indc &amp; ) [private]

## 4.6.4 Member Data Documentation

## 4.6.4.1 double VOL\_indc::v2

Definition at line 542 of file VolVolume.hpp.

## 4.6.4.2 double VOL\_indc::vu

Definition at line 543 of file VolVolume.hpp.

## 4.6.4.3 double VOL\_indc::vabs

Definition at line 544 of file VolVolume.hpp.

## 4.6.4.4 double VOL\_indc::asc

Definition at line 545 of file VolVolume.hpp.

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

- /home/ted/COIN/trunk/Vol/src/VolVolume.hpp

## 4.7 VOL\_ivector Class Reference

vector of ints.

```
#include <VolVolume.hpp>
```

## Public Member Functions

- [VOL\\_ivector](#) (const int s)  
*Construct a vector of size s.*
- [VOL\\_ivector](#) ()  
*Default constructor creates a vector of size 0.*
- [VOL\\_ivector](#) (const [VOL\\_ivector](#) &x)  
*Copy constructor makes a replica of x.*
- [~VOL\\_ivector](#) ()  
*The destructor deletes the data array.*
- int [size](#) () const  
*Return the size of the vector.*
- int & [operator\[\]](#) (const int i)  
*Return a reference to the i-th entry.*
- int [operator\[\]](#) (const int i) const  
*Return the i-th entry.*
- void [clear](#) ()  
*Delete the content of the vector and replace it with a vector of length 0.*
- void [allocate](#) (const int s)  
*delete the current vector and allocate space for a vector of size s.*

- void `swap` (`VOL_ivec` &`w`)  
*swaps the vector with `w`.*
- `VOL_ivec` & `operator=` (const `VOL_ivec` &`v`)  
*Copy `w` into the vector.*
- `VOL_ivec` & `operator=` (const int `w`)  
*Replace every entry in the vector with `w`.*

#### Public Attributes

- int \* `v`  
*The array holding the vector.*
- int `sz`  
*The size of the vector.*

#### 4.7.1 Detailed Description

vector of ints.

It's used to store indices, it has similar functions as `VOL_dvector`.

Note: If `VOL_DEBUG` is `#defined` to be 1 then each time an entry is accessed in the vector the index of the entry is tested for nonnegativity and for being less than the size of the vector. It's good to turn this on while debugging, but in final runs it should be turned off (because of the performance hit).

Definition at line 241 of file `VolVolume.hpp`.

#### 4.7.2 Constructor & Destructor Documentation

##### 4.7.2.1 `VOL_ivec::VOL_ivec ( const int s ) [inline]`

Construct a vector of size `s`.

The content of the vector is undefined.

Definition at line 249 of file `VolVolume.hpp`.

##### 4.7.2.2 `VOL_ivec::VOL_ivec ( ) [inline]`

Default constructor creates a vector of size 0.

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

##### 4.7.2.3 `VOL_ivec::VOL_ivec ( const VOL_ivec & x ) [inline]`

Copy constructor makes a replica of `x`.

Definition at line 256 of file `VolVolume.hpp`.

##### 4.7.2.4 `VOL_ivec::~~VOL_ivec ( ) [inline]`

The destructor deletes the data array.

Definition at line 264 of file `VolVolume.hpp`.

### 4.7.3 Member Function Documentation

#### 4.7.3.1 int VOL\_ivector::size ( ) const [inline]

Return the size of the vector.

Definition at line 269 of file VolVolume.hpp.

#### 4.7.3.2 int& VOL\_ivector::operator[] ( const int i ) [inline]

Return a reference to the *i*-th entry.

Definition at line 271 of file VolVolume.hpp.

#### 4.7.3.3 int VOL\_ivector::operator[] ( const int i ) const [inline]

Return the *i*-th entry.

Definition at line 277 of file VolVolume.hpp.

#### 4.7.3.4 void VOL\_ivector::clear ( ) [inline]

Delete the content of the vector and replace it with a vector of length 0.

Definition at line 284 of file VolVolume.hpp.

#### 4.7.3.5 void VOL\_ivector::allocate ( const int s ) [inline]

delete the current vector and allocate space for a vector of size *s*.

Definition at line 292 of file VolVolume.hpp.

#### 4.7.3.6 void VOL\_ivector::swap ( VOL\_ivector & w ) [inline]

swaps the vector with *w*.

Definition at line 299 of file VolVolume.hpp.

#### 4.7.3.7 VOL\_ivector& VOL\_ivector::operator= ( const VOL\_ivector & v )

Copy *w* into the vector.

#### 4.7.3.8 VOL\_ivector& VOL\_ivector::operator= ( const int w )

Replace every entry in the vector with *w*.

### 4.7.4 Member Data Documentation

#### 4.7.4.1 int\* VOL\_ivector::v

The array holding the vector.

Definition at line 244 of file VolVolume.hpp.

#### 4.7.4.2 int VOL\_ivector::sz

The size of the vector.

Definition at line 246 of file VolVolume.hpp.



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

- </home/ted/COIN/trunk/Vol/src/VolVolume.hpp>

## 4.8 VOL\_parms Struct Reference

This class contains the parameters controlling the Volume Algorithm.

```
#include <VolVolume.hpp>
```

### Public Attributes

- double [lambdainit](#)  
*initial value of lambda*
- double [alphainit](#)  
*initial value of alpha*
- double [alphamin](#)  
*minimum value for alpha*
- double [alphafactor](#)  
*when little progress is being done, we multiply alpha by alphafactor*
- double [ubinit](#)  
*initial upper bound of the value of an integer solution*
- double [primal\\_abs\\_precision](#)  
*accept if max abs viol is less than this*
- double [gap\\_abs\\_precision](#)  
*accept if abs gap is less than this*
- double [gap\\_rel\\_precision](#)  
*accept if rel gap is less than this*
- double [granularity](#)  
*terminate if best\_ub - lcost < granularity*
- double [minimum\\_rel\\_ascent](#)  
*terminate if the relative increase in lcost through ascent\_check\_invl steps is less than this*
- int [ascent\\_first\\_check](#)  
*when to check for sufficient relative ascent the first time*
- int [ascent\\_check\\_invl](#)  
*through how many iterations does the relative ascent have to reach a minimum*
- int [maxsgriters](#)  
*maximum number of iterations*
- int [printflag](#)  
*controls the level of printing.*
- int [printinvl](#)  
*controls how often do we print*
- int [heurinvl](#)  
*controls how often we run the primal heuristic*
- int [greentestinvl](#)  
*how many consecutive green iterations are allowed before changing lambda*
- int [yellowtestinvl](#)

- how many consecutive yellow iterations are allowed before changing lambda*
  - int `redtestinvl`
    - how many consecutive red iterations are allowed before changing lambda*
  - int `alphaint`
    - number of iterations before we check if alpha should be decreased*
  - char \* `temp_dualfile`
    - name of file for saving dual solution*

#### 4.8.1 Detailed Description

This class contains the parameters controlling the Volume Algorithm.

Definition at line 70 of file VolVolume.hpp.

#### 4.8.2 Member Data Documentation

##### 4.8.2.1 double VOL\_parms::lambdainit

initial value of lambda

Definition at line 72 of file VolVolume.hpp.

##### 4.8.2.2 double VOL\_parms::alphainit

initial value of alpha

Definition at line 74 of file VolVolume.hpp.

##### 4.8.2.3 double VOL\_parms::alphamin

minimum value for alpha

Definition at line 76 of file VolVolume.hpp.

##### 4.8.2.4 double VOL\_parms::alphafactor

when little progress is being done, we multiply alpha by alphafactor

Definition at line 78 of file VolVolume.hpp.

##### 4.8.2.5 double VOL\_parms::ubinit

initial upper bound of the value of an integer solution

Definition at line 81 of file VolVolume.hpp.

##### 4.8.2.6 double VOL\_parms::primal\_abs\_precision

accept if max abs viol is less than this

Definition at line 84 of file VolVolume.hpp.

##### 4.8.2.7 double VOL\_parms::gap\_abs\_precision

accept if abs gap is less than this

Definition at line 86 of file VolVolume.hpp.

#### 4.8.2.8 double VOL\_parms::gap\_rel\_precision

accept if rel gap is less than this

Definition at line 88 of file VolVolume.hpp.

#### 4.8.2.9 double VOL\_parms::granularity

terminate if  $\text{best\_ub} - \text{lcost} < \text{granularity}$

Definition at line 90 of file VolVolume.hpp.

#### 4.8.2.10 double VOL\_parms::minimum\_rel\_ascent

terminate if the relative increase in lcost through `ascent_check_invl` steps is less than this

Definition at line 94 of file VolVolume.hpp.

#### 4.8.2.11 int VOL\_parms::ascent\_first\_check

when to check for sufficient relative ascent the first time

Definition at line 96 of file VolVolume.hpp.

#### 4.8.2.12 int VOL\_parms::ascent\_check\_invl

through how many iterations does the relative ascent have to reach a minimum

Definition at line 99 of file VolVolume.hpp.

#### 4.8.2.13 int VOL\_parms::maxsgriters

maximum number of iterations

Definition at line 102 of file VolVolume.hpp.

#### 4.8.2.14 int VOL\_parms::printflag

controls the level of printing.

The flag should be the 'OR'-d value of the following options:

- 0 - print nothing
- 1 - print iteration information
- 2 - add lambda information
- 4 - add number of Red, Yellow, Green iterations

Default: 3

Definition at line 114 of file VolVolume.hpp.

#### 4.8.2.15 int VOL\_parms::printinvl

controls how often do we print

Definition at line 116 of file VolVolume.hpp.

#### 4.8.2.16 int VOL\_parms::heurinvl

controls how often we run the primal heuristic

Definition at line 118 of file VolVolume.hpp.

#### 4.8.2.17 int VOL\_parms::greentestinvl

how many consecutive green iterations are allowed before changing lambda

Definition at line 122 of file VolVolume.hpp.

#### 4.8.2.18 int VOL\_parms::yellowtestinvl

how many consecutive yellow iterations are allowed before changing lambda

Definition at line 125 of file VolVolume.hpp.

#### 4.8.2.19 int VOL\_parms::redtestinvl

how many consecutive red iterations are allowed before changing lambda

Definition at line 128 of file VolVolume.hpp.

#### 4.8.2.20 int VOL\_parms::alphaint

number of iterations before we check if alpha should be decreased

Definition at line 131 of file VolVolume.hpp.

#### 4.8.2.21 char\* VOL\_parms::temp\_dualfile

name of file for saving dual solution

Definition at line 134 of file VolVolume.hpp.

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

- /home/ted/COIN/trunk/Vol/src/[VolVolume.hpp](#)

## 4.9 VOL\_primal Class Reference

```
#include <VolVolume.hpp>
```

### Public Member Functions

- [VOL\\_primal](#) (const int psize, const int dsize)
- [VOL\\_primal](#) (const [VOL\\_primal](#) &primal)
- [~VOL\\_primal](#) ()
- [VOL\\_primal](#) & [operator=](#) (const [VOL\\_primal](#) &p)
- void [cc](#) (const double alpha, const [VOL\\_primal](#) &p)
- void [find\\_max\\_viol](#) (const [VOL\\_dvector](#) &dual\_lb, const [VOL\\_dvector](#) &dual\_ub)

### Public Attributes

- double [value](#)
- double [viol](#)
- [VOL\\_dvector](#) x
- [VOL\\_dvector](#) v

#### 4.9.1 Detailed Description

Definition at line 312 of file VolVolume.hpp.

#### 4.9.2 Constructor & Destructor Documentation

##### 4.9.2.1 VOL\_primal::VOL\_primal ( const int *psize*, const int *dsize* ) [inline]

Definition at line 323 of file VolVolume.hpp.

##### 4.9.2.2 VOL\_primal::VOL\_primal ( const VOL\_primal & *primal* ) [inline]

Definition at line 324 of file VolVolume.hpp.

##### 4.9.2.3 VOL\_primal::~VOL\_primal ( ) [inline]

Definition at line 326 of file VolVolume.hpp.

#### 4.9.3 Member Function Documentation

##### 4.9.3.1 VOL\_primal& VOL\_primal::operator= ( const VOL\_primal & *p* ) [inline]

Definition at line 327 of file VolVolume.hpp.

##### 4.9.3.2 void VOL\_primal::cc ( const double *alpha*, const VOL\_primal & *p* ) [inline]

Definition at line 341 of file VolVolume.hpp.

##### 4.9.3.3 void VOL\_primal::find\_max\_viol ( const VOL\_dvector & *dual\_lb*, const VOL\_dvector & *dual\_ub* )

#### 4.9.4 Member Data Documentation

##### 4.9.4.1 double VOL\_primal::value

Definition at line 315 of file VolVolume.hpp.

##### 4.9.4.2 double VOL\_primal::viol

Definition at line 317 of file VolVolume.hpp.

##### 4.9.4.3 VOL\_dvector VOL\_primal::x

Definition at line 319 of file VolVolume.hpp.

##### 4.9.4.4 VOL\_dvector VOL\_primal::v

Definition at line 321 of file VolVolume.hpp.

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

- /home/ted/COIN/trunk/Vol/src/[VolVolume.hpp](#)

## 4.10 VOL\_problem Class Reference

This class holds every data for the Volume Algorithm and its `solve` method must be invoked to solve the problem.

```
#include <VolVolume.hpp>
```

## Public Member Functions

### Constructors and destructor

- [VOL\\_problem](#) ()  
*Default constructor.*
- [VOL\\_problem](#) (const char \*filename)  
*Create a a [VOL\\_problem](#) object and read in the parameters from filename.*
- [~VOL\\_problem](#) ()  
*Destruct the object.*

### Method to solve the problem.

- int [solve](#) ([VOL\\_user\\_hooks](#) &hooks, const bool use\_preset\_dual=false)  
*Solve the problem using the *hooks*.*

### Methods returning final data

- int [iter](#) () const  
*returns the iteration number*
- double [alpha](#) () const  
*returns the value of alpha*
- double [lambda](#) () const  
*returns the value of lambda*

## Public Attributes

- int [iter\\_](#)  
*iteration number*
- double [\\_\\_pad0](#)

### External data (containing the result after solve)

- double [value](#)  
*final lagrangian value (OUTPUT)*
- [VOL\\_dvector](#) [dsol](#)  
*final dual solution (INPUT/OUTPUT)*
- [VOL\\_dvector](#) [psol](#)  
*final primal solution (OUTPUT)*
- [VOL\\_dvector](#) [viol](#)  
*violations (b-Ax) for the relaxed constraints*

### External data (may be changed by the user before calling solve)

- [VOL\\_parms](#) [parm](#)  
*The parameters controlling the Volume Algorithm (INPUT)*
- int [psize](#)  
*length of primal solution (INPUT)*
- int [dsize](#)  
*length of dual solution (INPUT)*
- [VOL\\_dvector](#) [dual\\_lb](#)  
*lower bounds for the duals (if 0 length, then filled with -inf) (INPUT)*
- [VOL\\_dvector](#) [dual\\_ub](#)  
*upper bounds for the duals (if 0 length, then filled with +inf) (INPUT)*

### Private Member Functions

- `VOL_problem` (const `VOL_problem` &)
- `VOL_problem` & `operator=` (const `VOL_problem` &)
- void `set_default_parm` ()

### Private methods used internally

- void `read_params` (const char \*filename)  
*Read in the parameters from the file filename.*
- int `initialize` (const bool use\_preset\_dual)  
*initializes duals, bounds for the duals, alpha, lambda*
- void `print_info` (const int iter, const `VOL_primal` &primal, const `VOL_primal` &pstar, const `VOL_dual` &dual)  
*print volume info every parm.printinvl iterations*
- double `readjust_target` (const double oldtarget, const double lcost) const  
*Checks if lcost is close to the target, if so it increases the target.*
- double `power_heur` (const `VOL_primal` &primal, const `VOL_primal` &pstar, const `VOL_dual` &dual) const  
*Here we decide the value of alpha1 to be used in the convex combination.*

### Private Attributes

#### Internal data (may be inquired for)

- double `alpha_`  
*value of alpha*
- double `lambda_`  
*value of lambda*
- union {  
    int `iter_`  
        *iteration number*  
    double `__pad0`  
};

#### 4.10.1 Detailed Description

This class holds every data for the Volume Algorithm and its `solve` method must be invoked to solve the problem.

The INPUT fields must be filled out completely before `solve` is invoked. `dsol` have to be filled out if and only if the last argument to `solve` is `true`.

Definition at line 604 of file `VolVolume.hpp`.

#### 4.10.2 Constructor & Destructor Documentation

4.10.2.1 `VOL_problem::VOL_problem ( const VOL_problem & ) [private]`

4.10.2.2 `VOL_problem::VOL_problem ( )`

Default constructor.

4.10.2.3 `VOL_problem::VOL_problem ( const char * filename )`

Create a `VOL_problem` object and read in the parameters from `filename`.

## 4.10.2.4 VOL\_problem::~~VOL\_problem ( )

Destruct the object.

## 4.10.3 Member Function Documentation

## 4.10.3.1 VOL\_problem&amp; VOL\_problem::operator=( const VOL\_problem &amp; ) [private]

## 4.10.3.2 void VOL\_problem::set\_default\_parm ( ) [private]

## 4.10.3.3 int VOL\_problem::solve ( VOL\_user\_hooks &amp; hooks, const bool use\_preset\_dual = false )

Solve the problem using the hooks.

Any information needed in the hooks must be stored in the structure `user_data` points to.

## 4.10.3.4 int VOL\_problem::iter ( ) const [inline]

returns the iteration number

Definition at line 680 of file `VolVolume.hpp`.

## 4.10.3.5 double VOL\_problem::alpha ( ) const [inline]

returns the value of alpha

Definition at line 682 of file `VolVolume.hpp`.

## 4.10.3.6 double VOL\_problem::lambda ( ) const [inline]

returns the value of lambda

Definition at line 684 of file `VolVolume.hpp`.

## 4.10.3.7 void VOL\_problem::read\_params ( const char \* filename ) [private]

Read in the parameters from the file `filename`.

## 4.10.3.8 int VOL\_problem::initialize ( const bool use\_preset\_dual ) [private]

initializes duals, bounds for the duals, alpha, lambda

## 4.10.3.9 void VOL\_problem::print\_info ( const int iter, const VOL\_primal &amp; primal, const VOL\_primal &amp; pstar, const VOL\_dual &amp; dual ) [private]

print volume info every `parm.printinvl` iterations

## 4.10.3.10 double VOL\_problem::readjust\_target ( const double oldtarget, const double lcost ) const [private]

Checks if `lcost` is close to the target, if so it increases the target.

Close means that we got within 5% of the target.

## 4.10.3.11 double VOL\_problem::power\_heur ( const VOL\_primal &amp; primal, const VOL\_primal &amp; pstar, const VOL\_dual &amp; dual ) const [private]

Here we decide the value of `alpha1` to be used in the convex combination.

The new `pstar` will be computed as



$pstar = \alpha1 * pstar + (1 - \alpha1) * primal$

More details of this are in doc.ps.

IN: alpha, primal, pstar, dual

#### Returns

alpha1

### 4.10.4 Member Data Documentation

#### 4.10.4.1 `double VOL_problem::alpha_ [private]`

value of alpha

Definition at line 634 of file VolVolume.hpp.

#### 4.10.4.2 `double VOL_problem::lambda_ [private]`

value of lambda

Definition at line 636 of file VolVolume.hpp.

#### 4.10.4.3 `int VOL_problem::iter_`

iteration number

Definition at line 641 of file VolVolume.hpp.

#### 4.10.4.4 `double VOL_problem::__pad0`

Definition at line 642 of file VolVolume.hpp.

#### 4.10.4.5 `union { ... } [private]`

#### 4.10.4.6 `double VOL_problem::value`

final lagrangian value (OUTPUT)

Definition at line 651 of file VolVolume.hpp.

#### 4.10.4.7 `VOL_dvector VOL_problem::dsol`

final dual solution (INPUT/OUTPUT)

Definition at line 653 of file VolVolume.hpp.

#### 4.10.4.8 `VOL_dvector VOL_problem::psol`

final primal solution (OUTPUT)

Definition at line 655 of file VolVolume.hpp.

#### 4.10.4.9 `VOL_dvector VOL_problem::viol`

violations (b-Ax) for the relaxed constraints

Definition at line 657 of file VolVolume.hpp.

## 4.10.4.10 VOL\_parms VOL\_problem::parm

The parameters controlling the Volume Algorithm (INPUT)

Definition at line 663 of file VolVolume.hpp.

## 4.10.4.11 int VOL\_problem::psize

length of primal solution (INPUT)

Definition at line 665 of file VolVolume.hpp.

## 4.10.4.12 int VOL\_problem::dsize

length of dual solution (INPUT)

Definition at line 667 of file VolVolume.hpp.

## 4.10.4.13 VOL\_dvector VOL\_problem::dual\_lb

lower bounds for the duals (if 0 length, then filled with -inf) (INPUT)

Definition at line 670 of file VolVolume.hpp.

## 4.10.4.14 VOL\_dvector VOL\_problem::dual\_ub

upper bounds for the duals (if 0 length, then filled with +inf) (INPUT)

Definition at line 673 of file VolVolume.hpp.

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

- </home/ted/COIN/trunk/Vol/src/VolVolume.hpp>

## 4.11 VOL\_swing Class Reference

```
#include <VolVolume.hpp>
```

## Public Types

- enum [condition](#) { [green](#), [yellow](#), [red](#) }

## Public Member Functions

- [VOL\\_swing](#) ()
- [~VOL\\_swing](#) ()
- void [cond](#) (const [VOL\\_dual](#) &dual, const double lcost, const double ascent, const int iter)
- double [lfactor](#) (const [VOL\\_parms](#) &parm, const double lambda, const int iter)
- void [print](#) ()

## Public Attributes

- enum [VOL\\_swing::condition](#) lastswing
- int [lastgreeniter](#)
- int [lastyellowiter](#)

- int [lastreiter](#)
- int [ngs](#)
- int [nrs](#)
- int [nys](#)
- int [rd](#)

#### Private Member Functions

- [VOL\\_swing](#) (const [VOL\\_swing](#) &)
- [VOL\\_swing](#) & [operator=](#) (const [VOL\\_swing](#) &)

#### 4.11.1 Detailed Description

Definition at line 389 of file VolVolume.hpp.

#### 4.11.2 Member Enumeration Documentation

##### 4.11.2.1 enum [VOL\\_swing::condition](#)

Enumerator

***green***  
***yellow***  
***red***

Definition at line 394 of file VolVolume.hpp.

#### 4.11.3 Constructor & Destructor Documentation

4.11.3.1 [VOL\\_swing::VOL\\_swing \( const \[VOL\\\_swing\]\(#\) & \)](#) [private]

4.11.3.2 [VOL\\_swing::VOL\\_swing \( \)](#) [inline]

Definition at line 399 of file VolVolume.hpp.

4.11.3.3 [VOL\\_swing::~~VOL\\_swing \( \)](#) [inline]

Definition at line 403 of file VolVolume.hpp.

#### 4.11.4 Member Function Documentation

4.11.4.1 [VOL\\_swing& VOL\\_swing::operator= \( const \[VOL\\\_swing\]\(#\) & \)](#) [private]

4.11.4.2 [void VOL\\_swing::cond \( const \[VOL\\\_dual\]\(#\) & \*dual\*, const double \*lcost\*, const double \*ascent\*, const int \*iter\* \)](#) [inline]

Definition at line 405 of file VolVolume.hpp.

4.11.4.3 [double VOL\\_swing::lfactor \( const \[VOL\\\_parms\]\(#\) & \*parm\*, const double \*lambda\*, const int \*iter\* \)](#) [inline]

Definition at line 430 of file VolVolume.hpp.

## 4.11.4.4 void VOL\_swing::print ( ) [inline]

Definition at line 479 of file VolVolume.hpp.

## 4.11.5 Member Data Documentation

## 4.11.5.1 enum VOL\_swing::condition VOL\_swing::lastswing

## 4.11.5.2 int VOL\_swing::lastgreeniter

Definition at line 395 of file VolVolume.hpp.

## 4.11.5.3 int VOL\_swing::lastyellowiter

Definition at line 395 of file VolVolume.hpp.

## 4.11.5.4 int VOL\_swing::lastrediter

Definition at line 395 of file VolVolume.hpp.

## 4.11.5.5 int VOL\_swing::ngs

Definition at line 396 of file VolVolume.hpp.

## 4.11.5.6 int VOL\_swing::nrs

Definition at line 396 of file VolVolume.hpp.

## 4.11.5.7 int VOL\_swing::nys

Definition at line 396 of file VolVolume.hpp.

## 4.11.5.8 int VOL\_swing::rd

Definition at line 397 of file VolVolume.hpp.

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

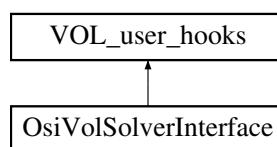
- [/home/ted/COIN/trunk/Vol/src/VolVolume.hpp](#)

## 4.12 VOL\_user\_hooks Class Reference

The user hooks should be overridden by the user to provide the problem specific routines for the volume algorithm.

```
#include <VolVolume.hpp>
```

Inheritance diagram for VOL\_user\_hooks:



## Public Member Functions

- virtual `~VOL_user_hooks()`
- virtual int `compute_rc` (const `VOL_dvector` &u, `VOL_dvector` &rc)=0  
*compute reduced costs*
- virtual int `solve_subproblem` (const `VOL_dvector` &dual, const `VOL_dvector` &rc, double &lcost, `VOL_dvector` &x, `VOL_dvector` &v, double &pcost)=0  
*Solve the subproblem for the subgradient step.*
- virtual int `heuristics` (const `VOL_problem` &p, const `VOL_dvector` &x, double &heur\_val)=0  
*Starting from the primal vector x, run a heuristic to produce an integer solution.*

### 4.12.1 Detailed Description

The user hooks should be overridden by the user to provide the problem specific routines for the volume algorithm.

The user should derive a class ...

for all hooks: return value of -1 means that volume should quit

Definition at line 562 of file VolVolume.hpp.

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 virtual `VOL_user_hooks::~VOL_user_hooks()` [inline],[virtual]

Definition at line 564 of file VolVolume.hpp.

### 4.12.3 Member Function Documentation

#### 4.12.3.1 virtual int `VOL_user_hooks::compute_rc` ( const `VOL_dvector` &u, `VOL_dvector` &rc ) [pure virtual]

compute reduced costs

Parameters

<i>u</i>	(IN) the dual variables
<i>rc</i>	(OUT) the reduced cost with respect to the dual values

Implemented in [OsiVolSolverInterface](#).

#### 4.12.3.2 virtual int `VOL_user_hooks::solve_subproblem` ( const `VOL_dvector` &dual, const `VOL_dvector` &rc, double &lcost, `VOL_dvector` &x, `VOL_dvector` &v, double &pcost ) [pure virtual]

Solve the subproblem for the subgradient step.

Parameters

<i>dual</i>	(IN) the dual variables
<i>rc</i>	(IN) the reduced cost with respect to the dual values
<i>lcost</i>	(OUT) the lagrangean cost with respect to the dual values

$x$	(OUT) the primal result of solving the subproblem
$v$	(OUT) b-Ax for the relaxed constraints
$pcost$	(OUT) the primal objective value of $x$

Implemented in [OsiVolSolverInterface](#).

4.12.3.3 `virtual int VOL_user_hooks::heuristics ( const VOL_problem & p, const VOL_dvector & x, double & heur_val )`  
`[pure virtual]`

Starting from the primal vector  $x$ , run a heuristic to produce an integer solution.

Parameters

$x$	(IN) the primal vector
$heur\_val$	(OUT) the value of the integer solution (return DBL_MAX here if no feas sol was found)

Implemented in [OsiVolSolverInterface](#).

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

- [/home/ted/COIN/trunk/Vol/src/VolVolume.hpp](#)

## 4.13 VOL\_vh Class Reference

```
#include <VolVolume.hpp>
```

### Public Member Functions

- [VOL\\_vh](#) (const double alpha, const [VOL\\_dvector](#) &dual\_lb, const [VOL\\_dvector](#) &dual\_ub, const [VOL\\_dvector](#) &v, const [VOL\\_dvector](#) &vstar, const [VOL\\_dvector](#) &u)
- [~VOL\\_vh](#) ()

### Public Attributes

- double [hh](#)
- double [norm](#)
- double [vh](#)
- double [asc](#)

### Private Member Functions

- [VOL\\_vh](#) (const [VOL\\_vh](#) &)
- [VOL\\_vh](#) & [operator=](#) (const [VOL\\_vh](#) &)

#### 4.13.1 Detailed Description

Definition at line 514 of file [VolVolume.hpp](#).

### 4.13.2 Constructor & Destructor Documentation

4.13.2.1 `VOL_vh::VOL_vh ( const VOL_vh & )` `[private]`

4.13.2.2 `VOL_vh::VOL_vh ( const double alpha, const VOL_dvector & dual_lb, const VOL_dvector & dual_ub, const VOL_dvector & v, const VOL_dvector & vstar, const VOL_dvector & u )`

4.13.2.3 `VOL_vh::~~VOL_vh ( )` `[inline]`

Definition at line 528 of file VolVolume.hpp.

### 4.13.3 Member Function Documentation

4.13.3.1 `VOL_vh& VOL_vh::operator= ( const VOL_vh & )` `[private]`

### 4.13.4 Member Data Documentation

4.13.4.1 `double VOL_vh::hh`

Definition at line 519 of file VolVolume.hpp.

4.13.4.2 `double VOL_vh::norm`

Definition at line 520 of file VolVolume.hpp.

4.13.4.3 `double VOL_vh::vh`

Definition at line 521 of file VolVolume.hpp.

4.13.4.4 `double VOL_vh::asc`

Definition at line 522 of file VolVolume.hpp.

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

- [/home/ted/COIN/trunk/Vol/src/VolVolume.hpp](#)

## 5 File Documentation

### 5.1 [/home/ted/COIN/trunk/Vol/src/OsiVol/OsiVolSolverInterface.hpp](#) File Reference

```
#include <string>
#include "VolVolume.hpp"
#include "CoinPackedMatrix.hpp"
#include "OsiSolverInterface.hpp"
```

#### Classes

- class [OsiVolSolverInterface](#)  
*Vol(ume) Solver Interface.*
- class [OsiVolSolverInterface::OsiVolMatrixOneMinusOne\\_](#)

## Functions

- void [OsiVolSolverInterfaceUnitTest](#) (const std::string &mpsDir, const std::string &netlibDir)  
*A function that tests the methods in the [OsiVolSolverInterface](#) class.*

## Variables

- static const double [OsiVolInfinity](#) = 1.0e31

### 5.1.1 Function Documentation

5.1.1.1 void [OsiVolSolverInterfaceUnitTest](#) ( const std::string & *mpsDir*, const std::string & *netlibDir* )

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

### 5.1.2 Variable Documentation

5.1.2.1 const double [OsiVolInfinity](#) = 1.0e31 [static]

Definition at line 18 of file [OsiVolSolverInterface.hpp](#).

## 5.2 /home/ted/COIN/trunk/Vol/src/VolVolume.hpp File Reference

```
#include <cfloat>
#include <algorithm>
#include <cstdio>
#include <cmath>
#include <cstring>
```

## Classes

- struct [VOL\\_parms](#)  
*This class contains the parameters controlling the Volume Algorithm.*
- class [VOL\\_dvector](#)  
*vector of doubles.*
- class [VOL\\_ivec](#)  
*vector of ints.*
- class [VOL\\_primal](#)
- class [VOL\\_dual](#)
- class [VOL\\_swing](#)
- class [VOL\\_alpha\\_factor](#)
- class [VOL\\_vh](#)
- class [VOL\\_indc](#)
- class [VOL\\_user\\_hooks](#)  
*The user hooks should be overridden by the user to provide the problem specific routines for the volume algorithm.*
- class [VOL\\_problem](#)  
*This class holds every data for the Volume Algorithm and its `solve` method must be invoked to solve the problem.*



## Macros

- `#define VOL_DEBUG 0`
- `#define VOL_TEST_INDEX(i, size)`
- `#define VOL_TEST_SIZE(size)`

## Functions

- `template<class T >`  
  `static T VolMax (register const T x, register const T y)`
- `template<class T >`  
  `static T VolAbs (register const T x)`

### 5.2.1 Macro Definition Documentation

#### 5.2.1.1 `#define VOL_DEBUG 0`

Definition at line 17 of file VolVolume.hpp.

#### 5.2.1.2 `#define VOL_TEST_INDEX( i, size )`

Definition at line 48 of file VolVolume.hpp.

#### 5.2.1.3 `#define VOL_TEST_SIZE( size )`

Definition at line 49 of file VolVolume.hpp.

### 5.2.2 Function Documentation

#### 5.2.2.1 `template<class T > static T VolMax ( register const T x, register const T y ) [inline],[static]`

Definition at line 21 of file VolVolume.hpp.

#### 5.2.2.2 `template<class T > static T VolAbs ( register const T x ) [inline],[static]`

Definition at line 26 of file VolVolume.hpp.

## Index

- ~OsiVolMatrixOneMinusOne\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_,  
6
- ~OsiVolSolverInterface
  - OsiVolSolverInterface, 14
- ~VOL\_alpha\_factor
  - VOL\_alpha\_factor, 29
- ~VOL\_dual
  - VOL\_dual, 30
- ~VOL\_dvector
  - VOL\_dvector, 32
- ~VOL\_indc
  - VOL\_indc, 34
- ~VOL\_ivector
  - VOL\_ivector, 36
- ~VOL\_primal
  - VOL\_primal, 42
- ~VOL\_problem
  - VOL\_problem, 44
- ~VOL\_swing
  - VOL\_swing, 48
- ~VOL\_user\_hooks
  - VOL\_user\_hooks, 50
- ~VOL\_vh
  - VOL\_vh, 52
- /home/ted/COIN/trunk/Vol/src/OsiVol/OsiVolSolver-  
Interface.hpp, 52
- /home/ted/COIN/trunk/Vol/src/VolVolume.hpp, 53
- \_\_pad0
  - VOL\_problem, 46
- addCol
  - OsiVolSolverInterface, 22
- addCols
  - OsiVolSolverInterface, 22
- addRow
  - OsiVolSolverInterface, 22
- addRows
  - OsiVolSolverInterface, 23
- allocate
  - VOL\_dvector, 33
  - VOL\_ivector, 37
- alpha
  - VOL\_problem, 45
- alpha\_
  - VOL\_problem, 46
- alphafactor
  - VOL\_parms, 39
- alphainit
  - VOL\_parms, 39
- alphaint
  - VOL\_parms, 41
- alphamin
  - VOL\_parms, 39
- applyColCut
  - OsiVolSolverInterface, 24
- applyRowCut
  - OsiVolSolverInterface, 24
- asc
  - VOL\_indc, 35
  - VOL\_vh, 52
- ascent
  - VOL\_dual, 31
- ascent\_check\_invl
  - VOL\_parms, 40
- ascent\_first\_check
  - VOL\_parms, 40
- assignProblem
  - OsiVolSolverInterface, 23, 24
- branchAndBound
  - OsiVolSolverInterface, 15
- cc
  - VOL\_dvector, 33
  - VOL\_primal, 42
- checkData\_
  - OsiVolSolverInterface, 25
- clear
  - VOL\_dvector, 33
  - VOL\_ivector, 37
- clone
  - OsiVolSolverInterface, 24
- colMatrix\_
  - OsiVolSolverInterface, 26
- colMatrixCurrent\_
  - OsiVolSolverInterface, 26
- colMatrixOneMinusOne\_
  - OsiVolSolverInterface, 27
- colRimAllocator\_
  - OsiVolSolverInterface, 25
- colRimResize\_
  - OsiVolSolverInterface, 25
- collower\_
  - OsiVolSolverInterface, 27
- colsol\_
  - OsiVolSolverInterface, 28
- colupper\_
  - OsiVolSolverInterface, 27
- compute\_rc
  - OsiVolSolverInterface, 25
  - VOL\_user\_hooks, 50
- compute\_rc\_
  - VOL\_parms, 41

- OsiVolSolverInterface, 25
- compute\_xrc
  - VOL\_dual, 31
- cond
  - VOL\_swing, 48
- condition
  - VOL\_swing, 48
- continuous\_
  - OsiVolSolverInterface, 27
- convertBoundsToSenses\_
  - OsiVolSolverInterface, 26
- convertSensesToBounds\_
  - OsiVolSolverInterface, 26
- deleteCols
  - OsiVolSolverInterface, 22
- deleteRows
  - OsiVolSolverInterface, 23
- dsize
  - VOL\_problem, 47
- dsol
  - VOL\_problem, 46
- dual\_lb
  - VOL\_problem, 47
- dual\_ub
  - VOL\_problem, 47
- factor
  - VOL\_alpha\_factor, 29
- find\_max\_viol
  - VOL\_primal, 42
- gap\_abs\_precision
  - VOL\_parms, 39
- gap\_rel\_precision
  - VOL\_parms, 39
- getColLower
  - OsiVolSolverInterface, 16
- getColSolution
  - OsiVolSolverInterface, 18
- getColUpper
  - OsiVolSolverInterface, 16
- getDbIParam
  - OsiVolSolverInterface, 15
- getDualRays
  - OsiVolSolverInterface, 18
- getEmptyWarmStart
  - OsiVolSolverInterface, 15
- getInfinity
  - OsiVolSolverInterface, 18
- getIntParam
  - OsiVolSolverInterface, 15
- getIterationCount
  - OsiVolSolverInterface, 18
- getMatrixByCol
  - OsiVolSolverInterface, 18
- getMatrixByRow
  - OsiVolSolverInterface, 18
- getNumCols
  - OsiVolSolverInterface, 16
- getNumElements
  - OsiVolSolverInterface, 16
- getNumRows
  - OsiVolSolverInterface, 16
- getObjCoefficients
  - OsiVolSolverInterface, 17
- getObjSense
  - OsiVolSolverInterface, 17
- getObjValue
  - OsiVolSolverInterface, 18
- getPrimalRays
  - OsiVolSolverInterface, 19
- getReducedCost
  - OsiVolSolverInterface, 18
- getRightHandSide
  - OsiVolSolverInterface, 17
- getRowActivity
  - OsiVolSolverInterface, 18
- getRowLower
  - OsiVolSolverInterface, 17
- getRowPrice
  - OsiVolSolverInterface, 18
- getRowRange
  - OsiVolSolverInterface, 17
- getRowSense
  - OsiVolSolverInterface, 16
- getRowUpper
  - OsiVolSolverInterface, 17
- getStrParam
  - OsiVolSolverInterface, 15
- getWarmStart
  - OsiVolSolverInterface, 16
- granularity
  - VOL\_parms, 40
- green
  - VOL\_swing, 48
- greentestinvl
  - VOL\_parms, 41
- gutsOfDestructor\_
  - OsiVolSolverInterface, 25
- heurinvl
  - VOL\_parms, 40
- heuristics
  - OsiVolSolverInterface, 25
  - VOL\_user\_hooks, 51
- hh
  - VOL\_vh, 52
- initFromClbCubObj

- OsiVolSolverInterface, 23
- initFromRhsSenseRange
  - OsiVolSolverInterface, 23
- initFromRlbRub
  - OsiVolSolverInterface, 23
- initialSolve
  - OsiVolSolverInterface, 15
- initialize
  - VOL\_problem, 45
- isAbandoned
  - OsiVolSolverInterface, 15
- isContinuous
  - OsiVolSolverInterface, 18
- isDualObjectiveLimitReached
  - OsiVolSolverInterface, 15
- isIterationLimitReached
  - OsiVolSolverInterface, 15
- isPrimalObjectiveLimitReached
  - OsiVolSolverInterface, 15
- isProvenDualInfeasible
  - OsiVolSolverInterface, 15
- isProvenOptimal
  - OsiVolSolverInterface, 15
- isProvenPrimalInfeasible
  - OsiVolSolverInterface, 15
- isZeroOneMinusOne\_
  - OsiVolSolverInterface, 26
- iter
  - VOL\_problem, 45
- iter\_
  - VOL\_problem, 46
- lagrangeanCost\_
  - OsiVolSolverInterface, 28
- lambda
  - VOL\_problem, 45
- lambda\_
  - VOL\_problem, 46
- lambdainit
  - VOL\_parms, 39
- lastgreeniter
  - VOL\_swing, 49
- lastrediter
  - VOL\_swing, 49
- lastswing
  - VOL\_swing, 49
- lastvalue
  - VOL\_alpha\_factor, 29
- lastyellowiter
  - VOL\_swing, 49
- lcost
  - VOL\_dual, 31
- lfactor
  - VOL\_swing, 48
- lhs\_
  - OsiVolSolverInterface, 28
- loadProblem
  - OsiVolSolverInterface, 23, 24
- majorDim\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, 6
- markHotStart
  - OsiVolSolverInterface, 16
- maxNumcols\_
  - OsiVolSolverInterface, 28
- maxNumrows\_
  - OsiVolSolverInterface, 28
- maxsgriters
  - VOL\_parms, 40
- minimum\_rel\_ascent
  - VOL\_parms, 40
- minorDim\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, 6
- minusInd\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, 7
- minusLength\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, 7
- minusSize\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, 7
- minusStart\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, 7
- ngs
  - VOL\_swing, 49
- norm
  - VOL\_vh, 52
- nrs
  - VOL\_swing, 49
- nys
  - VOL\_swing, 49
- objcoeffs\_
  - OsiVolSolverInterface, 27
- objsense\_
  - OsiVolSolverInterface, 27
- operator=
  - OsiVolSolverInterface, 24
  - VOL\_alpha\_factor, 29
  - VOL\_dual, 30
  - VOL\_dvector, 33
  - VOL\_indc, 35
  - VOL\_ivector, 37
  - VOL\_primal, 42

- VOL\_problem, 45
- VOL\_swing, 48
- VOL\_vh, 52
- OsiVolInfinity
  - OsiVolSolverInterface.hpp, 53
- OsiVolMatrixOneMinusOne\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, 6
- OsiVolSolverInterface, 7
  - ~OsiVolSolverInterface, 14
  - addCol, 22
  - addCols, 22
  - addRow, 22
  - addRows, 23
  - applyColCut, 24
  - applyRowCut, 24
  - assignProblem, 23, 24
  - branchAndBound, 15
  - checkData\_, 25
  - clone, 24
  - colMatrix\_, 26
  - colMatrixCurrent\_, 26
  - colMatrixOneMinusOne\_, 27
  - colRimAllocator\_, 25
  - colRimResize\_, 25
  - collower\_, 27
  - colsol\_, 28
  - colupper\_, 27
  - compute\_rc, 25
  - compute\_rc\_, 25
  - continuous\_, 27
  - convertBoundsToSenses\_, 26
  - convertSensesToBounds\_, 26
  - deleteCols, 22
  - deleteRows, 23
  - getColLower, 16
  - getColSolution, 18
  - getColUpper, 16
  - getDbfParam, 15
  - getDualRays, 18
  - getEmptyWarmStart, 15
  - getInfinity, 18
  - getIntParam, 15
  - getIterationCount, 18
  - getMatrixByCol, 18
  - getMatrixByRow, 18
  - getNumCols, 16
  - getNumElements, 16
  - getNumRows, 16
  - getObjCoefficients, 17
  - getObjSense, 17
  - getObjValue, 18
  - getPrimalRays, 19
  - getReducedCost, 18
  - getRightHandSide, 17
  - getRowActivity, 18
  - getRowLower, 17
  - getRowPrice, 18
  - getRowRange, 17
  - getRowSense, 16
  - getRowUpper, 17
  - getStrParam, 15
  - getWarmStart, 16
  - gutsOfDestructor\_, 25
  - heuristics, 25
  - initFromClbCubObj, 23
  - initFromRhsSenseRange, 23
  - initFromRlbRub, 23
  - initialSolve, 15
  - isAbandoned, 15
  - isContinuous, 18
  - isDualObjectiveLimitReached, 15
  - isIterationLimitReached, 15
  - isPrimalObjectiveLimitReached, 15
  - isProvenDualInfeasible, 15
  - isProvenOptimal, 15
  - isProvenPrimalInfeasible, 15
  - isZeroOneMinusOne\_, 26
  - lagrangeanCost\_, 28
  - lhs\_, 28
  - loadProblem, 23, 24
  - markHotStart, 16
  - maxNumcols\_, 28
  - maxNumrows\_, 28
  - objcoeffs\_, 27
  - objsense\_, 27
  - operator=, 24
  - OsiVolSolverInterface, 14
  - OsiVolSolverInterfaceUnitTest, 26
  - OsiVolSolverInterface, 14
  - rc\_, 28
  - readMps, 24
  - resolve, 15
  - rhs\_, 27
  - rowMatrix\_, 26
  - rowMatrixCurrent\_, 26
  - rowMatrixOneMinusOne\_, 26
  - rowRimAllocator\_, 25
  - rowRimResize\_, 25
  - rowlower\_, 27
  - rowprice\_, 28
  - rowpriceHotStart\_, 28
  - rowrange\_, 27
  - rowsense\_, 27
  - rowupper\_, 27
  - setColBounds, 19
  - setColLower, 19
  - setColSetBounds, 19

- setColSolution, [22](#)
- setColUpper, [19](#)
- setContinuous, [22](#)
- setDbiParam, [15](#)
- setIntParam, [15](#)
- setInteger, [22](#)
- setObjCoeff, [19](#)
- setObjSense, [22](#)
- setRowBounds, [20](#)
- setRowLower, [20](#)
- setRowPrice, [22](#)
- setRowSetBounds, [20](#)
- setRowSetTypes, [20](#)
- setRowType, [20](#)
- setRowUpper, [20](#)
- setStrParam, [15](#)
- setWarmStart, [16](#)
- solve\_subproblem, [25](#)
- solveFromHotStart, [16](#)
- test\_zero\_one\_minusone\_, [26](#)
- unmarkHotStart, [16](#)
- updateColMatrix\_, [25](#)
- updateRowMatrix\_, [25](#)
- volprob, [24](#)
- volprob\_, [28](#)
- writeMps, [24](#)
- OsiVolSolverInterface.hpp
  - OsiVolInfinity, [53](#)
  - OsiVolSolverInterfaceUnitTest, [53](#)
- OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, [5](#)
  - majorDim\_, [6](#)
  - minorDim\_, [6](#)
  - minusInd\_, [7](#)
  - minusLength\_, [7](#)
  - minusSize\_, [7](#)
  - minusStart\_, [7](#)
  - OsiVolMatrixOneMinusOne\_, [6](#)
  - plusInd\_, [6](#)
  - plusLength\_, [7](#)
  - plusSize\_, [6](#)
  - plusStart\_, [6](#)
  - timesMajor, [6](#)
- OsiVolSolverInterfaceUnitTest
  - OsiVolSolverInterface, [26](#)
  - OsiVolSolverInterface.hpp, [53](#)
- parm
  - VOL\_problem, [46](#)
- plusInd\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, [6](#)
- plusLength\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, [7](#)
- plusSize\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, [6](#)
- plusStart\_
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, [6](#)
- power\_heur
  - VOL\_problem, [45](#)
- primal\_abs\_precision
  - VOL\_parms, [39](#)
- print
  - VOL\_swing, [48](#)
- print\_info
  - VOL\_problem, [45](#)
- printfalg
  - VOL\_parms, [40](#)
- printinvl
  - VOL\_parms, [40](#)
- psize
  - VOL\_problem, [47](#)
- psol
  - VOL\_problem, [46](#)
- rc\_
  - OsiVolSolverInterface, [28](#)
- rd
  - VOL\_swing, [49](#)
- read\_params
  - VOL\_problem, [45](#)
- readMps
  - OsiVolSolverInterface, [24](#)
- readjust\_target
  - VOL\_problem, [45](#)
- red
  - VOL\_swing, [48](#)
- redtestinvl
  - VOL\_parms, [41](#)
- resolve
  - OsiVolSolverInterface, [15](#)
- rhs\_
  - OsiVolSolverInterface, [27](#)
- rowMatrix\_
  - OsiVolSolverInterface, [26](#)
- rowMatrixCurrent\_
  - OsiVolSolverInterface, [26](#)
- rowMatrixOneMinusOne\_
  - OsiVolSolverInterface, [26](#)
- rowRimAllocator\_
  - OsiVolSolverInterface, [25](#)
- rowRimResize\_
  - OsiVolSolverInterface, [25](#)
- rowlower\_
  - OsiVolSolverInterface, [27](#)
- rowprice\_
  - OsiVolSolverInterface, [27](#)

- OsiVolSolverInterface, 28
- rowpriceHotStart\_
  - OsiVolSolverInterface, 28
- rowrange\_
  - OsiVolSolverInterface, 27
- rowsense\_
  - OsiVolSolverInterface, 27
- rowupper\_
  - OsiVolSolverInterface, 27
- set\_default\_parm
  - VOL\_problem, 45
- setColBounds
  - OsiVolSolverInterface, 19
- setColLower
  - OsiVolSolverInterface, 19
- setColSetBounds
  - OsiVolSolverInterface, 19
- setColSolution
  - OsiVolSolverInterface, 22
- setColUpper
  - OsiVolSolverInterface, 19
- setContinuous
  - OsiVolSolverInterface, 22
- setDblParam
  - OsiVolSolverInterface, 15
- setIntParam
  - OsiVolSolverInterface, 15
- setInteger
  - OsiVolSolverInterface, 22
- setObjCoeff
  - OsiVolSolverInterface, 19
- setObjSense
  - OsiVolSolverInterface, 22
- setRowBounds
  - OsiVolSolverInterface, 20
- setRowLower
  - OsiVolSolverInterface, 20
- setRowPrice
  - OsiVolSolverInterface, 22
- setRowSetBounds
  - OsiVolSolverInterface, 20
- setRowSetTypes
  - OsiVolSolverInterface, 20
- setRowType
  - OsiVolSolverInterface, 20
- setRowUpper
  - OsiVolSolverInterface, 20
- setStrParam
  - OsiVolSolverInterface, 15
- setWarmStart
  - OsiVolSolverInterface, 16
- size
  - VOL\_dvector, 33
- VOL\_ivector, 37
- solve
  - VOL\_problem, 45
- solve\_subproblem
  - OsiVolSolverInterface, 25
  - VOL\_user\_hooks, 50
- solveFromHotStart
  - OsiVolSolverInterface, 16
- step
  - VOL\_dual, 30
- swap
  - VOL\_dvector, 33
  - VOL\_ivector, 37
- sz
  - VOL\_dvector, 34
  - VOL\_ivector, 37
- temp\_dualfile
  - VOL\_parms, 41
- test\_zero\_one\_minusone\_
  - OsiVolSolverInterface, 26
- timesMajor
  - OsiVolSolverInterface::OsiVolMatrixOneMinusOne\_, 6
- u
  - VOL\_dual, 31
- ubinit
  - VOL\_parms, 39
- unmarkHotStart
  - OsiVolSolverInterface, 16
- updateColMatrix\_
  - OsiVolSolverInterface, 25
- updateRowMatrix\_
  - OsiVolSolverInterface, 25
- v
  - VOL\_dvector, 33
  - VOL\_ivector, 37
  - VOL\_primal, 42
- v2
  - VOL\_indc, 35
- VOL\_swing
  - green, 48
  - red, 48
  - yellow, 48
- VOL\_DEBUG
  - VolVolume.hpp, 54
- VOL\_TEST\_INDEX
  - VolVolume.hpp, 54
- VOL\_TEST\_SIZE
  - VolVolume.hpp, 54
- VOL\_alpha\_factor, 29
  - ~VOL\_alpha\_factor, 29
  - factor, 29

- lastvalue, [29](#)
- operator=, [29](#)
- VOL\_alpha\_factor, [29](#)
- VOL\_alpha\_factor, [29](#)
- VOL\_dual, [30](#)
  - ~VOL\_dual, [30](#)
  - ascent, [31](#)
  - compute\_xrc, [31](#)
  - lcost, [31](#)
  - operator=, [30](#)
  - step, [30](#)
  - u, [31](#)
  - VOL\_dual, [30](#)
  - VOL\_dual, [30](#)
  - xrc, [31](#)
- VOL\_dvector, [31](#)
  - ~VOL\_dvector, [32](#)
  - allocate, [33](#)
  - cc, [33](#)
  - clear, [33](#)
  - operator=, [33](#)
  - size, [33](#)
  - swap, [33](#)
  - sz, [34](#)
  - v, [33](#)
  - VOL\_dvector, [32](#)
  - VOL\_dvector, [32](#)
- VOL\_indc, [34](#)
  - ~VOL\_indc, [34](#)
  - asc, [35](#)
  - operator=, [35](#)
  - v2, [35](#)
  - VOL\_indc, [34](#)
  - vabs, [35](#)
  - VOL\_indc, [34](#)
  - vu, [35](#)
- VOL\_ivector, [35](#)
  - ~VOL\_ivector, [36](#)
  - allocate, [37](#)
  - clear, [37](#)
  - operator=, [37](#)
  - size, [37](#)
  - swap, [37](#)
  - sz, [37](#)
  - v, [37](#)
  - VOL\_ivector, [36](#)
  - VOL\_ivector, [36](#)
- VOL\_parms, [38](#)
  - alphafactor, [39](#)
  - alphainit, [39](#)
  - alphaint, [41](#)
  - alphamin, [39](#)
  - ascent\_check\_invl, [40](#)
  - ascent\_first\_check, [40](#)
  - gap\_abs\_precision, [39](#)
  - gap\_rel\_precision, [39](#)
  - granularity, [40](#)
  - greentestinvl, [41](#)
  - heurinvl, [40](#)
  - lambdainit, [39](#)
  - maxsgriters, [40](#)
  - minimum\_rel\_ascent, [40](#)
  - primal\_abs\_precision, [39](#)
  - printflag, [40](#)
  - printinvl, [40](#)
  - redtestinvl, [41](#)
  - temp\_dualfile, [41](#)
  - ubinit, [39](#)
  - yellowtestinvl, [41](#)
- VOL\_primal, [41](#)
  - ~VOL\_primal, [42](#)
  - cc, [42](#)
  - find\_max\_viol, [42](#)
  - operator=, [42](#)
  - v, [42](#)
  - VOL\_primal, [42](#)
  - value, [42](#)
  - viol, [42](#)
  - VOL\_primal, [42](#)
  - x, [42](#)
- VOL\_problem, [42](#)
  - ~VOL\_problem, [44](#)
  - \_\_pad0, [46](#)
  - alpha, [45](#)
  - alpha\_, [46](#)
  - dsize, [47](#)
  - dsol, [46](#)
  - dual\_lb, [47](#)
  - dual\_ub, [47](#)
  - initialize, [45](#)
  - iter, [45](#)
  - iter\_, [46](#)
  - lambda, [45](#)
  - lambda\_, [46](#)
  - operator=, [45](#)
  - parm, [46](#)
  - power\_heur, [45](#)
  - print\_info, [45](#)
  - psize, [47](#)
  - psol, [46](#)
  - read\_params, [45](#)
  - readjust\_target, [45](#)
  - set\_default\_parm, [45](#)
  - solve, [45](#)
  - VOL\_problem, [44](#)
  - value, [46](#)
  - viol, [46](#)
  - VOL\_problem, [44](#)



- VOL\_swing, 47
  - ~VOL\_swing, 48
  - cond, 48
  - condition, 48
  - lastgreeniter, 49
  - lastreducer, 49
  - lastswing, 49
  - lastyellowiter, 49
  - lfactor, 48
  - ngs, 49
  - nrs, 49
  - nys, 49
  - operator=, 48
  - print, 48
  - rd, 49
  - VOL\_swing, 48
  - VOL\_swing, 48
- VOL\_user\_hooks, 49
  - ~VOL\_user\_hooks, 50
  - compute\_rc, 50
  - heuristics, 51
  - solve\_subproblem, 50
- VOL\_vh, 51
  - ~VOL\_vh, 52
  - asc, 52
  - hh, 52
  - norm, 52
  - operator=, 52
  - VOL\_vh, 52
  - vh, 52
  - VOL\_vh, 52
- vabs
  - VOL\_indc, 35
- value
  - VOL\_primal, 42
  - VOL\_problem, 46
- vh
  - VOL\_vh, 52
- viol
  - VOL\_primal, 42
  - VOL\_problem, 46
- VolAbs
  - VolVolume.hpp, 54
- VolMax
  - VolVolume.hpp, 54
- VolVolume.hpp
  - VOL\_DEBUG, 54
  - VOL\_TEST\_INDEX, 54
  - VOL\_TEST\_SIZE, 54
  - VolAbs, 54
  - VolMax, 54
- volprob
  - OsiVolSolverInterface, 24
- volprob\_
  - OsiVolSolverInterface, 28
- vu
  - VOL\_indc, 35
- writeMps
  - OsiVolSolverInterface, 24
- x
  - VOL\_primal, 42
- xrc
  - VOL\_dual, 31
- yellow
  - VOL\_swing, 48
- yellowtestinvl
  - VOL\_parms, 41