FlopCpp

trunk

Generated by Doxygen 1.8.9.1

Thu Oct 8 2015 22:47:02

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

- _EKKfactinfo [external]
- AbcDualRowPivot [external]
  - AbcDualRowDantzig [external]
  - AbcDualRowSteepest [external]
- AbcMatrix [external]
- AbcMatrix2 [external]
- AbcMatrix3 [external]
- AbcNonLinearCost [external]
- AbcPrimalColumnPivot [external]
  - AbcPrimalColumnDantzig [external]
  - AbcPrimalColumnSteepest [external]
- AbcSimplexFactorization [external]
- AbcTolerancesEtc [external]
- AbcWarmStartOrganizer [external]
- forcing_constraint_action::action [external]
- doubleton_action::action [external]
- tripleton_action::action [external]
- remove_fixed_action::action [external]
- std::allocator< T >
- ampl_info [external]
- OsiSolverInterface::ApplyCutsReturnCode [external]
- std::array< T >
- std::auto_ptr< T >
- auxiliary_graph [external]
- CbcGenCtlBlk::babState_struct [external]
- std::basic_string< Char >
  - std::string
  - std::wstring
- std::basic_string< char >
- std::basic_string< wchar_t >
- std::bitset< Bits >
- BitVector128 [external]
- blockStruct [external]
- blockStruct3 [external]

- CbcNodeInfo [external]
  - CbcFullNodeInfo [external]
  - CbcPartialNodeInfo [external]
- CbcObjectUpdateData [external]
- CbcOrClpParam [external]
- CbcParam [external]
- CbcGenCtlBlk::cbcParamsInfo_struct [external]
- CbcRowCuts [external]
- CbcSolver [external]
- CbcSolverUsefulData [external]
- CbcSolverUsefulData2 [external]
- CbcStatistics [external]
- CbcStopNow [external]
- CbcStrategy [external]
  - CbcStrategyDefault [external]
  - CbcStrategyDefaultSubTree [external]
  - CbcStrategyNull [external]
- CbcStrongInfo [external]
- CbcSymmetry [external]
- CbcThread [external]
- CbcTree [external]
  - CbcTreeLocal [external]
  - CbcTreeVariable [external]
- CbcUser [external]
- Cgl012Cut [external]
- cgl_arc [external]
- cgl_graph [external]
- cgl_node [external]
- CglBK [external]
- CglCutGenerator [external]
  - CglAllDifferent [external]
  - CglClique [external]
    - CglFakeClique [external]
  - CglDuplicateRow [external]
  - CglFlowCover [external]
  - CglGMI [external]
  - CglGomory [external]
  - CglImplication [external]
  - CglKnapsackCover [external]
  - CglLandP [external]
  - CglLiftAndProject [external]
  - CglMixedIntegerRounding [external]
  - CglMixedIntegerRounding2 [external]
  - CglOddHole [external]
  - CglProbing [external]
  - CglRedSplit [external]
  - CglRedSplit2 [external]
  - CglResidualCapacity [external]
  - CglSimpleRounding [external]
  - CglStored [external]
    - CglTemporary [external]
  - CglTwomir [external]
  - CglZeroHalf [external]
- CglFlowVUB [external]

- CoinPresolveAction [external]
  - do_tighten_action [external]
  - doubleton_action [external]
  - drop_empty_cols_action [external]
  - drop_empty_rows_action [external]
  - drop_zero_coefficients_action [external]
  - dupcol_action [external]
  - duprow3_action [external]
  - duprow_action [external]
  - forcing_constraint_action [external]
  - gubrow_action [external]
  - implied_free_action [external]
  - isolated_constraint_action [external]
  - make_fixed_action [external]
  - remove_dual_action [external]
  - remove_fixed_action [external]
  - slack_doubleton_action [external]
  - slack_singleton_action [external]
  - subst_constraint_action [external]
  - tripleton_action [external]
  - twoxtwo_action [external]
  - useless_constraint_action [external]
- CoinPresolveMonitor [external]
- CoinRational [external]
- CoinRelFltEq [external]
- CoinSearchTreeBase [external]
  - CoinSearchTree< class > [external]
- CoinSearchTreeCompareBest [external]
- CoinSearchTreeCompareBreadth [external]
- CoinSearchTreeCompareDepth [external]
- CoinSearchTreeComparePreferred [external]
- CoinSearchTreeManager [external]
- CoinSet [external]
  - CoinSosSet [external]
- CoinSnapshot [external]
- CoinThreadRandom [external]
- CoinTimer [external]
- CoinTreeNode [external]
  - CbcNode [external]
- CoinTreeSiblings [external]
- CoinTriple< S, T, U > [external]
- CoinWarmStart [external]
  - CoinWarmStartBasis [external]
    - AbcWarmStart [external]
  - CoinWarmStartDual [external]
  - CoinWarmStartPrimalDual [external]
  - CoinWarmStartVector< T > [external]
  - CoinWarmStartVector< double > [external]
  - CoinWarmStartVector< U > [external]
  - CoinWarmStartVectorPair< T, U > [external]
- CoinWarmStartDiff [external]
  - CoinWarmStartBasisDiff [external]
  - CoinWarmStartDualDiff [external]
  - CoinWarmStartPrimalDualDiff [external]

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Module Documentation

6.1 Public interface

Classes in this group are for normal modeling purposes.

Classes

- class flopc::MP_boolean

  Reference counted class for all "boolean" types of data.
- class flopc::MP_constraint

  Semantic representation of a linear constraint.
- class flopc::MP_data

  Input data set.
- class flopc::MP_domain

  Range over which some other constuct is defined.
- class flopc::MP_domain_set

  Range over which some other constuct is defined.
- class flopc::MP_domain_subset< nbr >

  Range over which some other constuct is defined.
- class flopc::MP_expression

  Symbolic representation of a linear expression.
- class flopc::MP_index

  Representation of an index.
- class flopc::MP_index_exp

  Representation of an expression involving an index.
- class flopc::Messenger

  Inteface for hooking up to internal flopc++ message handling.
- class flopc::MP_model

  This is the anchor point for all constructs in a FlopC++ model.
- class flopc::MP_set

  Representation of a set for indexing into some other construct.
- class flopc::MP_variable

  Symantic representation of a variable.
- class flopc::MP_binary_variable

  Specialized subclass of MP_variable where the variable is pre-specified to be binary.

Enumerations

- enum flopc::MP_model::MP_direction

  used when calling the solve() method.
- enum flopc::MP_model::MP_status { flopc::MP_model::OPTIMAL, flopc::MP_model::PRIMAL_INFEASIBLE, flopc::MP_model::DUAL_INFEASIBLE, flopc::MP_model::ABANDONED, flopc::MP_model::SOLVER_ONLY, flopc::MP_model::ATTACHED, flopc::MP_model::DETACHED }

  Reflects the state of the solution from solve()

Functions

- void flopc::forall (const MP_domain &d, const Functor &f)

  Global function for performing a Functor on each member of a MP_domain.
- void flopc::forall (const Functor &f)

  Global function for performing a Functor without having a set to operate on.
- void flopc::operator<<= (const MP_domain &s, const MP_domain &d)

  Global function which copies members of MP_domain d into another (possibly non-empty) MP_domain.
- void flopc::minimize (const MP_expression &obj)

  This is one of the main entry points for execution
  This calls the **OsiSolverInterface** *to execute the solver with the objective of MINIMIZING the argment MP_expression.*
- void flopc::minimize_max (MP_set &d, const MP_expression &obj)

  This is one of the main entry points for execution
  This calls the **OsiSolverInterface** *to execute the solver with the objective of MINIMIZING THE MAXIMUM of the MP_expression evaluation of the MP_set.*
- void flopc::maximize (const MP_expression &obj)

  This is one of the main entry points for execution
  This calls the **OsiSolverInterface** *to execute the solver with the objective of MAXIMIZING of the MP_expression.*
- MP_boolean flopc::operator! (const MP_boolean &b)

  For computing the logical negation of a boolean
  This is used in the normal formation of an expression.
- MP_boolean flopc::operator&& (const MP_boolean &e1, const MP_boolean &e2)

  For computing the logical AND of two booleans
  This is used in the normal formation of an expression.
- MP_boolean flopc::operator|| (const MP_boolean &e1, const MP_boolean &e2)

  For computing the logical OR of two booleans
  This is used in the normal formation of an expression.
- MP_boolean flopc::alltrue (const MP_domain &d, const MP_boolean &b)

  boolean which returns true if all in domain evaluate to true.
- MP_boolean flopc::operator<= (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean flopc::operator<= (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean flopc::operator< (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean flopc::operator< (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.

- MP_boolean flopc::operator>= (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean flopc::operator>= (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean flopc::operator> (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean flopc::operator> (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean flopc::operator== (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean flopc::operator== (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean flopc::operator!= (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean flopc::operator!= (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- Constant flopc::abs (const Constant &c)

  for computing the absolute value of a constant value.
- Constant flopc::pos (const Constant &c)

  for returning non-negative value of the constant.
- Constant flopc::ceil (const Constant &c)

  The ceiling integral value of the input constant.
- Constant flopc::floor (const Constant &c)

  The floor integral value of the input constant.
- Constant flopc::minimum (const Constant &a, const Constant &b)

  Returns the smaller of two constants.
- Constant flopc::maximum (const Constant &a, const Constant &b)

  Returns the larger of two constants.
- Constant flopc::operator+ (const Constant &a, const Constant &b)

  Returns the sum of two constants.
- Constant flopc::operator- (const Constant &a, const Constant &b)

  Returns the difference of two constants.
- Constant flopc::operator* (const Constant &a, const Constant &b)

  Returns the product of two constants.
- Constant flopc::operator/ (const Constant &a, const Constant &b)

  Returns the quotient of two constants.
- Constant flopc::maximum (const MP_domain &i, const Constant &e)

  Returns the maximum over the domain of the constant.
- Constant flopc::minimum (const MP_domain &i, const Constant &e)

  Returns the sum of two constants.
- Constant flopc::sum (const MP_domain &i, const Constant &e)

  Returns the sum of two constants.
- Constant flopc::product (const MP_domain &i, const Constant &e)

  Returns the sum of two constants.

- Constraint flopc::operator<= (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator<= (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator<= (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator<= (const VariableRef &l, const VariableRef &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator>= (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator>= (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator>= (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator>= (const VariableRef &l, const VariableRef &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator== (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator== (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator== (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint flopc::operator== (const VariableRef &l, const VariableRef &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.

6.1.1 Detailed Description

Classes in this group are for normal modeling purposes.

They are intended for consumption outside the library.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum flopc::MP_model::MP_status

Reflects the state of the solution from solve()

Enumerator

OPTIMAL if the solve method is called and the optimal solution found.

PRIMAL_INFEASIBLE if solve is called and solver finds model primal infeasible.

DUAL_INFEASIBLE if solve is called and solver finds the model dual infeasible.

ABANDONED if solve is called and solver abandons the problem (time?, iter limit?)

SOLVER_ONLY A solver is placed in the constructor, but it is not yet attached or solved.

ATTACHED A solver is attached, but not yet solved.

DETACHED No solver is attached.

Definition at line 97 of file MP_model.hpp.

6.1.3 Function Documentation

6.1.3.1 void flopc::forall (const Functor & *f*) [inline]

Global function for performing a Functor without having a set to operate on.

Definition at line 64 of file flopc.hpp.

6.1.3.2 void flopc::minimize (const MP_expression & *obj*) [inline]

This is one of the main entry points for execution

This calls the **OsiSolverInterface** to execute the solver with the objective of MINIMIZING the argment MP_expression.

- Assumes that the **OsiSolverInterface** is already set
- Assumes a model is already loaded (and is the default model)

Definition at line 83 of file flopc.hpp.

6.1.3.3 void flopc::minimize_max (MP_set & *d,* const MP_expression & *obj*) [inline]

This is one of the main entry points for execution

This calls the **OsiSolverInterface** to execute the solver with the objective of MINIMIZING THE MAXIMUM of the MP_expression evaluation of the MP_set.

Definition at line 96 of file flopc.hpp.

6.1.3.4 void flopc::maximize (const MP_expression & *obj*) [inline]

This is one of the main entry points for execution

This calls the **OsiSolverInterface** to execute the solver with the objective of MAXIMIZING of the MP_expression.

- Assumes that the **OsiSolverInterface** is already set
- Assumes a model is already loaded (and is the default model)

Definition at line 107 of file flopc.hpp.

6.1.3.5 MP_boolean flopc::operator! (const MP_boolean & *b*)

For computing the logical negation of a boolean

This is used in the normal formation of an expression.

Parameters

b	boolean

Returns

A boolean which evaluates to the negation of the input expression.

6.1.3.6 MP_boolean flopc::operator&& (const MP_boolean & *e1*, const MP_boolean & *e2*)

For computing the logical AND of two booleans

This is used in the normal formation of an expression.

Parameters

e1	first boolean
e2	second boolean

Returns

A boolean which evaluates to true of both booleans are true.

6.1.3.7 MP_boolean flopc::operator|| (const MP_boolean & *e1*, const MP_boolean & *e2*)

For computing the logical OR of two booleans

This is used in the normal formation of an expression.

Parameters

e1	first boolean
e2	second boolean

Returns

A boolean which evaluates to true if either booleans are true.

6.1.3.8 MP_boolean flopc::alltrue (const MP_domain & *d*, const MP_boolean & *b*)

boolean which returns true if all in domain evaluate to true.

This is used in the normal formation of an expression.

Parameters

d	MP_domain to evaluate with
b	boolean expression to evaluate.

Returns

A boolean which evaluates to true all domain evaluations of the boolean evaluate to true.

6.1.3.9 MP_boolean flopc::operator<= (const MP_index_exp & *e1*, const MP_index_exp & *e2*)

constructs a boolean evaluator using operator overloading

This is used in the normal formation of an expression.

This is useful when combining index expressions.

Parameters

e1	is an index expression involving an MP_index
e2	second index expression • used in forming sets of tuples of index values, or subsets.

The brief code below is a bit contrived, but the right hand side illustrate the utility of combining an index expression.

```
MP_index i;
MP_index j;
MP_boolean &b = (i+5)<=(j);
```

6.1.3.10 MP_boolean flopc::operator<= (const Constant & *e1*, const Constant & *e2*)

constructs a boolean evaluator by comparing two constants.

This is used in the normal formation of an expression. This utility of this is when comparing constants

Parameters

e1	first constant expression
e2	second constant expression

6.1.3.11 MP_boolean flopc::operator< (const MP_index_exp & *e1*, const MP_index_exp & *e2*)

constructs a boolean evaluator using operator overloading

This is used in the normal formation of an expression.

This is useful when combining index expressions.

Parameters

e1	is an index expression involving an MP_index

e2	second index expression • used in forming sets of tuples of index values, or subsets.

6.1.3.12 MP_boolean flopc::operator< (const Constant & *e1*, const Constant & *e2*)

constructs a boolean evaluator by comparing two constants.

This is used in the normal formation of an expression. This utility of this is when comparing constants

Parameters

e1	first constant expression
e2	second constant expression

6.1.3.13 MP_boolean flopc::operator>= (const MP_index_exp & *e1*, const MP_index_exp & *e2*)

constructs a boolean evaluator using operator overloading

This is used in the normal formation of an expression.

This is useful when combining index expressions.

Parameters

e1	is an index expression involving an MP_index
e2	second index expression • used in forming sets of tuples of index values, or subsets.

6.1.3.14 MP_boolean flopc::operator>= (const Constant & *e1*, const Constant & *e2*)

constructs a boolean evaluator by comparing two constants.

This is used in the normal formation of an expression. This utility of this is when comparing constants

Parameters

e1	first constant expression
e2	second constant expression

6.1.3.15 MP_boolean flopc::operator> (const MP_index_exp & *e1*, const MP_index_exp & *e2*)

constructs a boolean evaluator using operator overloading

This is used in the normal formation of an expression.

This is useful when combining index expressions.

Parameters

e1	is an index expression involving an MP_index
e2	second index expression • used in forming sets of tuples of index values, or subsets.

6.1.3.16 MP_boolean flopc::operator> (const Constant & *e1*, const Constant & *e2*)

constructs a boolean evaluator by comparing two constants.

This is used in the normal formation of an expression. This utility of this is when comparing constants

Parameters

e1	first constant expression
e2	second constant expression

6.1.3.17 MP_boolean flopc::operator== (const MP_index_exp & *e1*, const MP_index_exp & *e2*)

constructs a boolean evaluator using operator overloading

This is used in the normal formation of an expression.

This is useful when combining index expressions.

Parameters

e1	is an index expression involving an MP_index
e2	second index expression • used in forming sets of tuples of index values, or subsets.

6.1.3.18 MP_boolean flopc::operator== (const Constant & *e1*, const Constant & *e2*)

constructs a boolean evaluator by comparing two constants.

This is used in the normal formation of an expression. This utility of this is when comparing constants

Parameters

e1	first constant expression
e2	second constant expression

6.1.3.19 MP_boolean flopc::operator!= (const MP_index_exp & *e1*, const MP_index_exp & *e2*)

constructs a boolean evaluator using operator overloading

This is used in the normal formation of an expression.

This is useful when combining index expressions.

Parameters

e1	is an index expression involving an MP_index
e2	second index expression • used in forming sets of tuples of index values, or subsets.

6.1.3.20 MP_boolean flopc::operator!= (const Constant & *e1*, const Constant & *e2*)

constructs a boolean evaluator by comparing two constants.

This is used in the normal formation of an expression. This utility of this is when comparing constants

Parameters

e1	first constant expression
e2	second constant expression

6.1.3.21 Constant flopc::abs (const Constant & *c*)

for computing the absolute value of a constant value.

This is used in the normal formation of an expression such as abs(-5)

- input is a constant. It cannot be a variable expression.
- Returns a Constant evaluating to the absolute value of the parameter

6.1.3.22 Constant flopc::pos (const Constant & *c*)

for returning non-negative value of the constant.

This is used in the formation of an expression. It is used to return a non-negative value..

Parameters

c	an imput constant

Returns

the absolute value of the constant.

- if the Constant is positive, it returns a positive number.
- if the Constant is negative or zero, it returns 0.0

6.1.3.23 Constant flopc::ceil (const Constant & *c*)

The ceiling integral value of the input constant.

This is used in the formation of an expression. It is used to "round up" a numeric constant which is potentially non-integer.

Parameters

c	is a constant

Returns

the ceiling or "rounded up" of the parameter

- ceil(3.2) evaluates to 4.0

6.1.3.24 Constant flopc::floor (const Constant & *c*)

The floor integral value of the input constant.

This is used in the formation of an expression. It is used to "truncate" a numeric constant which is potentially non-integer.

Parameters

c	is a constant

Returns

the floor or "truncated" value of the parameter

- floor(3.7) evaluates to 3.0

6.1.3.25 Constant flopc::minimum (const Constant & *a,* const Constant & *b*)

Returns the smaller of two constants.

This is used in the formation of an expression.

Parameters

a	first constant
b	second constant

Returns

the lesser of the two values.

- minimum(3.6,3.7) evaluates to 3.6

6.1.3.26 Constant flopc::maximum (const Constant & *a,* const Constant & *b*)

Returns the larger of two constants.

This is used in the formation of an expression.

Parameters

a	first constant

b	second constant

Returns

the greater of the two numbers

- maximum(3.6,3.7) evaluates to 3.7

6.1.3.27 Constant flopc::operator+ (const Constant & *a*, const Constant & *b*)

Returns the sum of two constants.

This is used in the formation of an expression.

Parameters

a	first constant
b	second constant

Returns

the sum of the constants.

6.1.3.28 Constant flopc::operator- (const Constant & *a*, const Constant & *b*)

Returns the difference of two constants.

This is used in the formation of an expression.

Parameters

a	first constant
b	second constant

Returns

the difference between the constants.

6.1.3.29 Constant flopc::operator∗ (const Constant & *a*, const Constant & *b*)

Returns the product of two constants.

This is used in the formation of an expression.

Parameters

a	first constant
b	second constant

Returns

the result of multiplying the constants.

6.1.3.30 Constant flopc::operator/ (const Constant & *a*, const Constant & *b*)

Returns the quotient of two constants.

This is used in the formation of an expression.

Parameters

a	first constant
b	second constant

Returns

the result of dividing the first parameter by the second.

6.1.3.31 Constant flopc::maximum (const MP_domain & *i*, const Constant & *e*)

Returns the maximum over the domain of the constant.

6.1.3.32 Constant flopc::minimum (const MP_domain & *i*, const Constant & *e*)

Returns the sum of two constants.

6.1.3.33 Constant flopc::sum (const MP_domain & *i*, const Constant & *e*)

Returns the sum of two constants.

6.1.3.34 Constant flopc::product (const MP_domain & *i*, const Constant & *e*)

Returns the sum of two constants.

6.1.3.35 Constraint flopc::operator<= (const MP_expression & *l*, const MP_expression & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 69 of file MP_constraint.hpp.

6.1.3.36 Constraint flopc::operator<= (const Constant & *l*, const MP_expression & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 77 of file MP_constraint.hpp.

6.1.3.37 Constraint flopc::operator<= (const MP_expression & *l*, const Constant & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 85 of file MP_constraint.hpp.

6.1.3.38 Constraint flopc::operator<= (const VariableRef & *l*, const VariableRef & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 93 of file MP_constraint.hpp.

6.1.3.39 Constraint flopc::operator>= (const MP_expression & *l*, const MP_expression & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 102 of file MP_constraint.hpp.

6.1.3.40 Constraint flopc::operator>= (const Constant & *l*, const MP_expression & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 110 of file MP_constraint.hpp.

6.1.3.41 Constraint flopc::operator>= (const MP_expression & *l*, const Constant & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 118 of file MP_constraint.hpp.

6.1.3.42 Constraint flopc::operator>= (const VariableRef & *l*, const VariableRef & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 126 of file MP_constraint.hpp.

6.1.3.43 Constraint flopc::operator== (const MP_expression & *l*, const MP_expression & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 135 of file MP_constraint.hpp.

6.1.3.44 Constraint flopc::operator== (const Constant & *l*, const MP_expression & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 143 of file MP_constraint.hpp.

6.1.3.45 Constraint flopc::operator== (const MP_expression & *l*, const Constant & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 151 of file MP_constraint.hpp.

6.1.3.46 Constraint flopc::operator== (const VariableRef & *l*, const VariableRef & *r*) [inline]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 159 of file MP_constraint.hpp.

6.2 Internal (private) interface.

Classes in this group are used for internal purposes.

Classes

- class flopc::Boolean_base

  Base class for all "boolean" types of data.
- class flopc::Constant_base

  Base class for all "constant" types of data.
- class flopc::Constant

  Reference counted class for all "constant" types of data.
- class flopc::Constraint

  Semantic representation of a constraint in a Math Program.
- class flopc::DataRef

  Reference to a set of data.
- class flopc::MP_domain_base

  Reference to a set of index values.
- class flopc::insertFunctor< nbr >

  Inserter for construction of a subset.
- struct flopc::Coef

  Internal representation of a Coefficient in a matrix.
- class flopc::GenerateFunctor

  Functor to facilitate generation of coefficients.
- class flopc::ObjectiveGenerateFunctor

  Functor to facilitate generation of the objective function.
- class flopc::MP_expression_base

  The base class for all expressions.
- class flopc::TerminalExpression

  The base class for all expressions.
- class flopc::Expression_operator

  The base class for all expressions.
- class flopc::MP_index_base

  Internal representation of a index.
- class flopc::MP_index_mult

  Internal representation of an index expression.
- class flopc::MP_index_sum

  Internal representation of an index expression.
- class flopc::MP_index_dif

  Internal representation of an index expression.
- class flopc::NormalMessenger

  Internal use: used when Normal output is selected.
- class flopc::VerboseMessenger

  Internal use: used when Verbose output is selected.
- class flopc::MP_set_base

  Internal representation of a "set".
- class flopc::InsertFunctor< nbr >

Internal representation of a "set".

- class flopc::MP_subset< nbr >

  Internal representation of a "set".
- class flopc::SUBSETREF

  Internal representation of a "set".
- class flopc::SubsetRef< nbr >

  Internal representation of a "set".
- class flopc::Functor

  Function object.
- class flopc::RowMajor

  Utility class to flatten multidimensional information into single dimentional offset information.
- class flopc::Named

  Utility interface class for adding a string name onto a structure.
- class flopc::Handle< T >

  Utility for doing reference counted pointers.
- class flopc::VariableRef

  Semantic representation of a variable in a Math Program.

Enumerations

- enum flopc::Sense_enum

  Enumeration for indicating direction of a constraint.
- enum flopc::variableType

  Enumeration for indicating variable type.

Functions

- template<int nbr, class T >
  std::vector< T > flopc::makeVector (T i1, T i2=0, T i3=0, T i4=0, T i5=0)

  This template makes a vector of appropriate size out of the variable number of arguments.

6.2.1 Detailed Description

Classes in this group are used for internal purposes.

They are not intended for consumption outside the library.

Chapter 7

Namespace Documentation

7.1 flopc Namespace Reference

All flopc++ code is contained within the flopc namespace.

Classes

- class Boolean_base

  Base class for all "boolean" types of data.
- struct Coef

  Internal representation of a Coefficient in a matrix.
- class Constant

  Reference counted class for all "constant" types of data.
- class Constant_base

  Base class for all "constant" types of data.
- class Constraint

  Semantic representation of a constraint in a Math Program.
- class DataRef

  Reference to a set of data.
- class Expression_operator

  The base class for all expressions.
- class Functor

  Function object.
- class GenerateFunctor

  Functor to facilitate generation of coefficients.
- class Handle

  Utility for doing reference counted pointers.
- class InsertFunctor

  Internal representation of a "set".
- class insertFunctor

  Inserter for construction of a subset.
- class Messenger

  Inteface for hooking up to internal flopc++ message handling.

- class MP_binary_variable

  Specialized subclass of MP_variable where the variable is pre-specified to be binary.
- class MP_boolean

  Reference counted class for all "boolean" types of data.
- class MP_constraint

  Semantic representation of a linear constraint.
- class MP_data

  Input data set.
- class MP_domain

  Range over which some other constuct is defined.
- class MP_domain_base

  Reference to a set of index values.
- class MP_domain_set

  Range over which some other constuct is defined.
- class MP_domain_subset

  Range over which some other constuct is defined.
- class MP_expression

  Symbolic representation of a linear expression.
- class MP_expression_base

  The base class for all expressions.
- class MP_index

  Representation of an index.
- class MP_index_base

  Internal representation of a index.
- class MP_index_dif

  Internal representation of an index expression.
- class MP_index_exp

  Representation of an expression involving an index.
- class MP_index_mult

  Internal representation of an index expression.
- class MP_index_sum

  Internal representation of an index expression.
- class MP_model

  This is the anchor point for all constructs in a FlopC++ model.
- class MP_set

  Representation of a set for indexing into some other construct.
- class MP_set_base

  Internal representation of a "set".
- class MP_stage
- class MP_stochastic_data
- class MP_subset

  Internal representation of a "set".
- class MP_variable

  Symantic representation of a variable.
- class Named

  Utility interface class for adding a string name onto a structure.
- class NormalMessenger

Internal use: used when Normal output is selected.

- class ObjectiveGenerateFunctor

  Functor to facilitate generation of the objective function.
- class RowMajor

  Utility class to flatten multidimensional information into single dimentional offset information.
- class SUBSETREF

  Internal representation of a "set".
- class SubsetRef

  Internal representation of a "set".
- class TerminalExpression

  The base class for all expressions.
- class VariableRef

  Semantic representation of a variable in a Math Program.
- class VerboseMessenger

  Internal use: used when Verbose output is selected.

Enumerations

- enum Sense_enum

  Enumeration for indicating direction of a constraint.
- enum variableType

  Enumeration for indicating variable type.

Functions

- void forall (const MP_domain &d, const Functor &f)

  Global function for performing a Functor on each member of a MP_domain.
- void forall (const Functor &f)

  Global function for performing a Functor without having a set to operate on.
- void operator<<= (const MP_domain &s, const MP_domain &d)

  Global function which copies members of MP_domain d into another (possibly non-empty) MP_domain.
- void minimize (const MP_expression &obj)

  This is one of the main entry points for execution
  This calls the **OsiSolverInterface** *to execute the solver with the objective of MINIMIZING the argment MP_expression.*
- void minimize_max (MP_set &d, const MP_expression &obj)

  This is one of the main entry points for execution
  This calls the **OsiSolverInterface** *to execute the solver with the objective of MINIMIZING THE MAXIMUM of the MP_↩ expression evaluation of the MP_set.*
- void maximize (const MP_expression &obj)

  This is one of the main entry points for execution
  This calls the **OsiSolverInterface** *to execute the solver with the objective of MAXIMIZING of the MP_expression.*
- MP_boolean operator! (const MP_boolean &b)

  For computing the logical negation of a boolean
  This is used in the normal formation of an expression.
- MP_boolean operator&& (const MP_boolean &e1, const MP_boolean &e2)

  For computing the logical AND of two booleans
  This is used in the normal formation of an expression.
- MP_boolean operator|| (const MP_boolean &e1, const MP_boolean &e2)

For computing the logical OR of two booleans
This is used in the normal formation of an expression.

- MP_boolean alltrue (const MP_domain &d, const MP_boolean &b)

  boolean which returns true if all in domain evaluate to true.
- MP_boolean operator<= (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean operator<= (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean operator< (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean operator< (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean operator>= (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean operator>= (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean operator> (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean operator> (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean operator== (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean operator== (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- MP_boolean operator!= (const MP_index_exp &e1, const MP_index_exp &e2)

  constructs a boolean evaluator using operator overloading
  This is used in the normal formation of an expression.
- MP_boolean operator!= (const Constant &e1, const Constant &e2)

  constructs a boolean evaluator by comparing two constants.
- Constant abs (const Constant &c)

  for computing the absolute value of a constant value.
- Constant pos (const Constant &c)

  for returning non-negative value of the constant.
- Constant ceil (const Constant &c)

  The ceiling integral value of the input constant.
- Constant floor (const Constant &c)

  The floor integral value of the input constant.
- Constant minimum (const Constant &a, const Constant &b)

  Returns the smaller of two constants.
- Constant maximum (const Constant &a, const Constant &b)

  Returns the larger of two constants.
- Constant operator+ (const Constant &a, const Constant &b)

  Returns the sum of two constants.

- Constant operator- (const Constant &a, const Constant &b)

  Returns the difference of two constants.
- Constant operator∗ (const Constant &a, const Constant &b)

  Returns the product of two constants.
- Constant operator/ (const Constant &a, const Constant &b)

  Returns the quotient of two constants.
- Constant maximum (const MP_domain &i, const Constant &e)

  Returns the maximum over the domain of the constant.
- Constant minimum (const MP_domain &i, const Constant &e)

  Returns the sum of two constants.
- Constant sum (const MP_domain &i, const Constant &e)

  Returns the sum of two constants.
- Constant product (const MP_domain &i, const Constant &e)

  Returns the sum of two constants.
- Constraint operator<= (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator<= (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator<= (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator<= (const VariableRef &l, const VariableRef &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator>= (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator>= (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator>= (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator>= (const VariableRef &l, const VariableRef &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator== (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator== (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator== (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator== (const VariableRef &l, const VariableRef &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.

- MP_domain operator∗ (const MP_domain &a, const MP_domain &b)

  operator which creates a new domain as the pairwise combinations of two input domains.
- MP_expression operator+ (const MP_expression &e1, const MP_expression &e2)

  Operator which sums two MP_expressions, forms a new MP_expression.
- MP_expression operator+ (const MP_expression &e1, const Constant &e2)

  Operator which sums an MP_expression and a constant, and forms a new MP_expression.
- MP_expression operator+ (const Constant &e1, const MP_expression &e2)

  Operator which sums a constant and an MP_expression , and forms a new MP_expression.
- MP_expression operator- (const MP_expression &e1, const MP_expression &e2)

  Operator which subtracts an MP_expression from an MP_expression, and forms a new MP_expression.
- MP_expression operator- (const MP_expression &e1, const Constant &e2)

  Operator which subtracts a Constant from an MP_expression, and forms a new MP_expression.
- MP_expression operator- (const Constant &e1, const MP_expression &e2)

  Operator which subtracts an MP_expression from a Constant, and forms a new MP_expression.
- MP_expression operator∗ (const Constant &e1, const MP_expression &e2)

  Operator which multiplies a Constant by an MP_expression, and forms a new MP_expression.
- MP_expression operator∗ (const MP_expression &e1, const Constant &e2)

  Operator which multiplies an MP_expression by a Constant, and forms a new MP_expression.
- MP_expression sum (const MP_domain &d, const MP_expression &e)

  forms an expression by summing an expression over a domain.
- Constant operator+ (MP_index &a, MP_index &b)

  returns a Constant as a result of addition of two MP_index values.
- Constant operator- (MP_index &a, MP_index &b)

  returns a Constant as a result of a difference of two MP_index values.
- MP_index_exp operator- (MP_index &i, const int &j)

  returns an index expression from a difference between an MP_index and an integer.
- MP_index_exp operator+ (MP_index &i, const int &j)

  returns an index expression from a sum between an MP_index and an integer.
- MP_index_exp operator+ (MP_index &i, const Constant &j)

  returns an index expression from a sum between an MP_index and a Constant.
- MP_index_exp operator∗ (MP_index &i, const Constant &j)

  returns an index expression from a product between an MP_index and a Constant.
- std::ostream & operator<< (std::ostream &os, const MP_model::MP_status &condition)

  allows print of result from call to solve();
- std::ostream & operator<< (std::ostream &os, const MP_model::MP_direction &direction)

  allows print of direction used when calling solve. (MIN/MAX)
- template<int nbr, class T >
  std::vector< T > makeVector (T i1, T i2=0, T i3=0, T i4=0, T i5=0)

  This template makes a vector of appropriate size out of the variable number of arguments.
- int mod (int a, int b)

  return the strictly positive modulus of two integers

Variables

- const int outOfBound = -2

  Distinct return value on conditions where an index goes out of bounds.

7.1.1 Detailed Description

All flopc++ code is contained within the flopc namespace.

Flopc++ is an open source algebraic modelling language implemented as a C++ class library. It uses the common C↩ OIN-OR **OsiSolverInterface** abstract interface to allow for easy integration with many of today's top Math Programming solvers.

•

main 3 components are listed below. Much of the rest of the code is to facilitate the operator overloading makes this such a powerful modeling environment.

- Linear Variables MP_variable
- Linear Set MP_set
- Linear Index MP_index
- Linear Constraints MP_constraint

Note

The classes in PublicInterface are intended for consumption outside the library.

7.1.2 Function Documentation

7.1.2.1 MP_expression flopc::sum (const MP_domain & *d,* const MP_expression & *e*)

forms an expression by summing an expression over a domain.

Note

it's expected that the expression is defined over that domain.

7.1.2.2 MP_index_exp flopc::operator- (MP_index & *i,* const int & *j*)

returns an index expression from a difference between an MP_index and an integer.

(i-5)

7.1.2.3 MP_index_exp flopc::operator+ (MP_index & *i,* const int & *j*)

returns an index expression from a sum between an MP_index and an integer.

(i+5)

Chapter 8

Class Documentation

8.1 flopc::Boolean_base Class Reference

Base class for all "boolean" types of data.

`#include <MP_boolean.hpp>`

8.1.1 Detailed Description

Base class for all "boolean" types of data.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 23 of file MP_boolean.hpp.

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

- MP_boolean.hpp

8.2 flopc::Coef Struct Reference

Internal representation of a Coefficient in a matrix.

`#include <MP_expression.hpp>`

8.2.1 Detailed Description

Internal representation of a Coefficient in a matrix.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 34 of file MP_expression.hpp.

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

- MP_expression.hpp

8.3 flopc::Constant Class Reference

Reference counted class for all "constant" types of data.

`#include <MP_constant.hpp>`

Inheritance diagram for flopc::Constant:

8.4 flopc::Constant_base Class Reference

Base class for all "constant" types of data.

`#include <MP_constant.hpp>`

Inheritance diagram for flopc::Constant_base:

8.4.1 Detailed Description

Base class for all "constant" types of data.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 20 of file MP_constant.hpp.

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

- MP_constant.hpp

8.5 flopc::Constraint Class Reference

Semantic representation of a constraint in a Math Program.

`#include <MP_constraint.hpp>`

Friends

- Constraint operator<= (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator<= (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator<= (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator<= (const VariableRef &l, const VariableRef &r)

Uses operator overloading to construct an Constraint
Constucts a Constraint using operator overloading.

- Constraint operator>= (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator>= (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator>= (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator>= (const VariableRef &l, const VariableRef &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator== (const MP_expression &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator== (const Constant &l, const MP_expression &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator== (const MP_expression &l, const Constant &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.
- Constraint operator== (const VariableRef &l, const VariableRef &r)

  Uses operator overloading to construct an Constraint
  Constucts a Constraint using operator overloading.

8.5.1 Detailed Description

Semantic representation of a constraint in a Math Program.

See also

MP_constraint for a public interface.

Note

of interest is the operator overloads which are 'friends'

Definition at line 39 of file MP_constraint.hpp.

8.5.2 Friends And Related Function Documentation

8.5.2.1 Constraint operator<= (const MP_expression & *l*, const MP_expression & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 69 of file MP_constraint.hpp.

8.5.2.2 Constraint operator<= (const Constant & *l*, const MP_expression & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 77 of file MP_constraint.hpp.

8.5.2.3 Constraint operator<= (const MP_expression & *l*, const Constant & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 85 of file MP_constraint.hpp.

8.5.2.4 Constraint operator<= (const VariableRef & *l*, const VariableRef & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 93 of file MP_constraint.hpp.

8.5.2.5 Constraint operator>= (const MP_expression & *l*, const MP_expression & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 102 of file MP_constraint.hpp.

8.5.2.6 Constraint operator>= (const Constant & *l*, const MP_expression & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 110 of file MP_constraint.hpp.

8.5.2.7 Constraint operator>= (const MP_expression & *l*, const Constant & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 118 of file MP_constraint.hpp.

8.5.2.8 Constraint operator>= (const VariableRef & *l*, const VariableRef & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 126 of file MP_constraint.hpp.

8.5.2.9 Constraint operator== (const MP_expression & *l*, const MP_expression & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 135 of file MP_constraint.hpp.

8.5.2.10 Constraint operator== (const Constant & *l*, const MP_expression & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 143 of file MP_constraint.hpp.

8.5.2.11 Constraint operator== (const MP_expression & *l*, const Constant & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 151 of file MP_constraint.hpp.

8.5.2.12 Constraint operator== (const VariableRef & *l*, const VariableRef & *r*) [friend]

Uses operator overloading to construct an Constraint

Constucts a Constraint using operator overloading.

See also

MP_constraint

Definition at line 159 of file MP_constraint.hpp.

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

- MP_constraint.hpp

8.6 flopc::DataRef Class Reference

Reference to a set of data.

`#include <MP_data.hpp>`

Inheritance diagram for flopc::DataRef:

Collaboration diagram for flopc::DataRef:

8.6.1 Detailed Description

Reference to a set of data.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 29 of file MP_data.hpp.

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

- MP_data.hpp

8.7 flopc::Expression_operator Class Reference

The base class for all expressions.

`#include <MP_expression.hpp>`

Inheritance diagram for flopc::Expression_operator:

Collaboration diagram for flopc::Expression_operator:

8.7.1 Detailed Description

The base class for all expressions.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 157 of file MP_expression.hpp.

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

- MP_expression.hpp

8.8 flopc::Functor Class Reference

Function object.

`#include <MP_utilities.hpp>`

Inheritance diagram for flopc::Functor:

8.8.1 Detailed Description

Function object.

Often used

Note

is the base class for passing a fucntion object around.

Definition at line 26 of file MP_utilities.hpp.

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

- MP_utilities.hpp

8.9 flopc::GenerateFunctor Class Reference

Functor to facilitate generation of coefficients.

`#include <MP_expression.hpp>`

Inheritance diagram for flopc::GenerateFunctor:

Collaboration diagram for flopc::GenerateFunctor:

8.9.1 Detailed Description

Functor to facilitate generation of coefficients.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 48 of file MP_expression.hpp.

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

- MP_expression.hpp

8.10 flopc::Handle< T > Class Template Reference

Utility for doing reference counted pointers.

`#include <MP_utilities.hpp>`

Inheritance diagram for flopc::Handle< T >:

Protected Member Functions

- void decrement ()

8.10.1 Detailed Description

template<class T>class flopc::Handle< T >

Utility for doing reference counted pointers.

Definition at line 105 of file MP_utilities.hpp.

8.10.2 Member Function Documentation

8.10.2.1 template<class T> void flopc::Handle< T >::decrement ( ) [inline],[protected]

if(root->count != 0) {

}

Definition at line 133 of file MP_utilities.hpp.

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

- MP_utilities.hpp

8.11 flopc::insertFunctor< nbr > Class Template Reference

Inserter for construction of a subset.

`#include <MP_domain.hpp>`

Inheritance diagram for flopc::insertFunctor< nbr >:

Collaboration diagram for flopc::insertFunctor< nbr >:

Public Member Functions

- void operator() () const

8.11.1 Detailed Description

template<int nbr>class flopc::insertFunctor< nbr >

Inserter for construction of a subset.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 141 of file MP_domain.hpp.

8.11.2 Member Function Documentation

8.11.2.1 template<int nbr> void flopc::insertFunctor< nbr >::operator() ( ) const [inline], [virtual]

Implements flopc::Functor.

Definition at line 145 of file MP_domain.hpp.

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

- MP_domain.hpp

8.12 flopc::insertFunctor< nbr > Class Template Reference

Inserter for construction of a subset.

`#include <MP_domain.hpp>`

Inheritance diagram for flopc::insertFunctor< nbr >:

Collaboration diagram for flopc::insertFunctor< nbr >:

Public Member Functions

- void operator() () const

8.12.1 Detailed Description

template<int nbr>class flopc::insertFunctor< nbr >

Inserter for construction of a subset.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 141 of file MP_domain.hpp.

8.12.2 Member Function Documentation

8.12.2.1 template<int nbr> void flopc::insertFunctor< nbr >::operator() ( ) const [inline], [virtual]

Implements flopc::Functor.

Definition at line 145 of file MP_domain.hpp.

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

- MP_domain.hpp

8.13 flopc::Messenger Class Reference

Inteface for hooking up to internal flopc++ message handling.

`#include <MP_model.hpp>`

Inheritance diagram for flopc::Messenger:

8.13.1 Detailed Description

Inteface for hooking up to internal flopc++ message handling.

In more advanced use of FlopC++, it may be desirable to get access to internal calls for messages. In essence, subclass this Messenger class, and register it with the MP_model. Also overload whichever message events you wish to handle.

Definition at line 35 of file MP_model.hpp.

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

- MP_model.hpp

8.14 flopc::MP_binary_variable Class Reference

Specialized subclass of MP_variable where the variable is pre-specified to be binary.

`#include <MP_variable.hpp>`

Inheritance diagram for flopc::MP_binary_variable:

Collaboration diagram for flopc::MP_binary_variable:

Additional Inherited Members

8.14.1 Detailed Description

Specialized subclass of MP_variable where the variable is pre-specified to be binary.

Definition at line 136 of file MP_variable.hpp.

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

- MP_variable.hpp

8.15 flopc::MP_boolean Class Reference

Reference counted class for all "boolean" types of data.

`#include <MP_boolean.hpp>`

Inheritance diagram for flopc::MP_boolean:

Collaboration diagram for flopc::MP_boolean:

Additional Inherited Members

8.15.1 Detailed Description

Reference counted class for all "boolean" types of data.

This contains counters to ConstantBase pointers. These pointers may be of any of the Boolean_base * type. This can be a constant valued boolean as well. explain SUBSETREF explain using pointer in – should be private?

Definition at line 45 of file MP_boolean.hpp.

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

- MP_boolean.hpp

8.16 flopc::MP_constraint Class Reference

Semantic representation of a linear constraint.

`#include <MP_constraint.hpp>`

Inheritance diagram for flopc::MP_constraint:

Collaboration diagram for flopc::MP_constraint:

Public Member Functions

- MP_constraint (const MP_set_base &s1=MP_set::getEmpty(), const MP_set_base &s2=MP_set::getEmpty(), const MP_set_base &s3=MP_set::getEmpty(), const MP_set_base &s4=MP_set::getEmpty(), const MP_set_base &s5=MP_set::getEmpty())

  construct the MP_constraint with appropriate sets for indexing.

8.16.1 Detailed Description

Semantic representation of a linear constraint.

This is one of the main public interface classes. It is always constructed through operator overloading between expressions, constants, and variables. There are many 'friend' overloaded operators to do the constuction. The basic idea is to make the constraint look like a paper-model constraint in C++ code. Once constructed, it should be added to the model.

The snippet below is an overly simplistic example, but is ok for illustration.

```
MP_model aModel; // your model
MP_set I; // the set the constraint is defined over.
MP_variable x(I); // your variable
...
MP_constraint cons(I); // construct the right number of constraints.
cons = x <= 3;
// Assign in the semantic rep to it.  aModel.add(cons); // add it to the model
```

There is quite a bit of C++ machinery going on there.

- MP_expression(const VariableRef& v); converts the VariableRef x into an MP_expression.
- MP_constraint cons(I); construct the right dimensioned sized bundle of constraints.
- friend Constraint operator<=(const MP_expression& l, const Constant& r); converts the x <= 3 into an Constraint.

Definition at line 194 of file MP_constraint.hpp.

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

- MP_constraint.hpp

8.17 flopc::MP_data Class Reference

Input data set.

`#include <MP_data.hpp>`

Inheritance diagram for flopc::MP_data:

Collaboration diagram for flopc::MP_data:

Public Member Functions

- void initialize (double d)

  similar to value() but copies the same value to all entries.
- MP_data (const MP_set_base &s1=MP_set::getEmpty(), const MP_set_base &s2=MP_set::getEmpty(), const MP_set_base &s3=MP_set::getEmpty(), const MP_set_base &s4=MP_set::getEmpty(), const MP_set_base &s5=MP_set::getEmpty())

  Constructs the MP_data object, and allocates space for data, but does not initialize the data.
- MP_data (double *value, const MP_set_base &s1=MP_set::getEmpty(), const MP_set_base &s2=MP_set::getEmpty(), const MP_set_base &s3=MP_set::getEmpty(), const MP_set_base &s4=MP_set::getEmpty(), const MP_set_base &s5=MP_set::getEmpty())

Construct the object, and uses the data in the original array (shallow copy)

- ~MP_data ()
- void value (const double *d)

  Used to bind and deep copy data into the MP_data data structure.
- operator double ()
- double & operator() (int lcli1, int lcli2=0, int lcli3=0, int lcli4=0, int lcli5=0)

  Looks up the data based on the index values passed in.
- DataRef & operator() (const MP_index_exp &lcli1=MP_index_exp::getEmpty(), const MP_index_exp &lcli2=MP_index_exp::getEmpty(), const MP_index_exp &lcli3=MP_index_exp::getEmpty(), const MP_index_exp &lcli4=MP_index_exp::getEmpty(), const MP_index_exp &lcli5=MP_index_exp::getEmpty())

  returns a DataRef which refers into the MP_data.
- void display (std::string s="")

  For displaying data in a human readable format.

8.17.1 Detailed Description

Input data set.

This is one of the main public interface classes. It is normally directly constructed given a set of indices (domain) over which it is valid. If the data is not bound at construction, either the value() or initialize() method must be called which (deep) copies in the actual data. If one wishes to refer to external data instead rather than doing a deep copy, use the constructor which takes the value pointer as an argument. This copies the original data pointer value (rather than a deep copy).
This is used for construction of :

- objective coefficients
- constraint coefficients
- 'right hand sides'

Definition at line 71 of file MP_data.hpp.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 flopc::MP_data::~MP_data ( ) [inline]

Definition at line 114 of file MP_data.hpp.

8.17.3 Member Function Documentation

8.17.3.1 flopc::MP_data::operator double ( ) [inline]

Definition at line 131 of file MP_data.hpp.

8.17.3.2 double& flopc::MP_data::operator() (int *lcli1*, int *lcli2* = 0, int *lcli3* = 0, int *lcli4* = 0, int *lcli5* = 0) [inline]

Looks up the data based on the index values passed in.

Note

this is used internally, but may also be useful for spot checking data or in other expressions.

Definition at line 139 of file MP_data.hpp.

8.17.3.3 DataRef& flopc::MP_data::operator() (const MP_index_exp & *lcli1* = MP_index_exp::getEmpty (), const MP_index_exp & *lcli2* = MP_index_exp::getEmpty (), const MP_index_exp & *lcli3* = MP_index_exp::getEmpty (), const MP_index_exp & *lcli4* = MP_index_exp::getEmpty (), const MP_index_exp & *lcli5* = MP_index_exp::getEmpty ()) [inline]

returns a DataRef which refers into the MP_data.

Note

For internal use.

Definition at line 158 of file MP_data.hpp.

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

- MP_data.hpp

8.18 flopc::MP_domain Class Reference

Range over which some other constuct is defined.

`#include <MP_domain.hpp>`

Inheritance diagram for flopc::MP_domain:

Collaboration diagram for flopc::MP_domain:

Public Member Functions

- MP_domain ()
  a set which points to nothing.
- MP_domain (MP_domain_base ∗r)
  For internal use.
- MP_domain such_that (const MP_boolean &b)
  Special conditional creation of a subset.
- void Forall (const Functor ∗op) const
  Special conditional operation on the domain.
- size_t size () const
  returns number of elements in the domain.

Static Public Member Functions

- static const MP_domain & getEmpty ()
  returns a reference to the "empty" set.

Friends

- MP_domain operator∗ (const MP_domain &a, const MP_domain &b)

  operator which creates a new domain as the pairwise combinations of two input domains.

Additional Inherited Members

8.18.1 Detailed Description

Range over which some other constuct is defined.

This is one of the main public interface classes. One uses this in the context of a constraint, objective, variable, or data. It is usually used in conjunction with an MP_set, or a subset, but can be used without one. It is the range over which the other construct is defined.

- there is a special domain known as "empty". It is static and a reference can be obtained using MP_domain::get↩ Empty();
- The empty set is used when defaulting in parameters for dimensions which are not used.

Definition at line 61 of file MP_domain.hpp.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 flopc::MP_domain::MP_domain ( )

a set which points to nothing.

Note

This is not the same as the "empty" set.

8.18.2.2 flopc::MP_domain::MP_domain (MP_domain_base ∗ *r*)

For internal use.

8.18.3 Member Function Documentation

8.18.3.1 MP_domain flopc::MP_domain::such_that (const MP_boolean & *b*)

Special conditional creation of a subset.

This method allows for a test for inclusion of a condition during construction of a subset. The output MP_domain will include references where the condition is satisfied.

8.18.3.2 void flopc::MP_domain::Forall (const Functor ∗ *op*) const

Special conditional operation on the domain.

This method will call the functor for each member of the MP_domain.

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

- MP_domain.hpp

8.19 flopc::MP_domain_base Class Reference

Reference to a set of index values.

`#include <MP_domain.hpp>`

Inheritance diagram for flopc::MP_domain_base:

Collaboration diagram for flopc::MP_domain_base:

Friends

- MP_domain operator∗ (const MP_domain &a, const MP_domain &b)
  operator which creates a new domain as the pairwise combinations of two input domains.

8.19.1 Detailed Description

Reference to a set of index values.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 30 of file MP_domain.hpp.

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

- MP_domain.hpp

8.20 flopc::MP_domain_set Class Reference

Range over which some other constuct is defined.

`#include <MP_domain.hpp>`

Inheritance diagram for flopc::MP_domain_set:

Collaboration diagram for flopc::MP_domain_set:

Public Member Functions

- MP_domain_set (const MP_set ∗s, MP_index ∗i)
  Constructor taking a set pointer and an index pointer.
- void operator() () const
- int evaluate () const
  Evaluates the index within the domain.
- const MP_set_base ∗ getSet () const
  Getter for the set used in construction.
- MP_index ∗ getIndex () const
  Getter for the index used in construction.
- MP_domain getDomain (MP_set ∗s) const

8.20.1 Detailed Description

Range over which some other constuct is defined.

This is one of the main public interface classes. One uses this in the context of a constraint, objective, variable, or data. This class in the MP_domain family uses an MP_set and and index for defining the Range over which the construct is defined.

- This defines the domain as the contents of the set when referred into by the index.

Definition at line 110 of file MP_domain.hpp.

8.20.2 Member Function Documentation

8.20.2.1 void flopc::MP_domain_set::operator() ( ) const [virtual]

Implements flopc::Functor.

8.20.2.2 int flopc::MP_domain_set::evaluate ( ) const [virtual]

Evaluates the index within the domain.

Note

For internal use

Implements flopc::MP_index_base.

8.20.2.3 MP_domain flopc::MP_domain_set::getDomain (MP_set * *s*) const [virtual]

Note

Internal use.

Implements flopc::MP_index_base.

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

- MP_domain.hpp

8.21 flopc::MP_domain_subset< nbr > Class Template Reference

Range over which some other constuct is defined.

`#include <MP_domain.hpp>`

Inheritance diagram for flopc::MP_domain_subset< nbr >:

Collaboration diagram for flopc::MP_domain_subset< nbr >:

Public Member Functions

- int evaluate () const

  Evaluates the index within the domain.
- MP_set_base * getSet () const

  getter for obtaining the set used in construction
- MP_index * getIndex () const

  getter for obtaining the index used in construction
- MP_domain getDomain (MP_set *s) const
- void operator() () const
- Functor * makeInsertFunctor () const

8.21.1 Detailed Description

template<int nbr>class flopc::MP_domain_subset< nbr >

Range over which some other constuct is defined.

Uses subsetting.

This is one of the main public interface classes. One uses this in the context of a constraint, objective, variable, or data. This class in the MP_domain family uses an MP_subset and a vector of indexes for defining the Range over which the construct is defined.

- This defines the domain as the contents of the subset when referred into by the indexes.

Definition at line 134 of file MP_domain.hpp.

8.21.2 Member Function Documentation

8.21.2.1 template<int nbr> int flopc::MP_domain_subset< nbr >::evaluate ( ) const [inline], [virtual]

Evaluates the index within the domain.

Note

For internal use

Implements flopc::MP_index_base.

Definition at line 175 of file MP_domain.hpp.

8.21.2.2 template<int nbr> MP_domain flopc::MP_domain_subset< nbr >::getDomain (MP_set * *s*) const [inline], [virtual]

Note

Internal use.

Implements flopc::MP_index_base.

Definition at line 189 of file MP_domain.hpp.

8.21.2.3 template<int nbr> void flopc::MP_domain_subset< nbr >::operator() ( ) const [inline], [virtual]

Implements flopc::Functor.

Definition at line 193 of file MP_domain.hpp.

8.21.2.4 template<int nbr> Functor* flopc::MP_domain_subset< nbr >::makeInsertFunctor ( ) const [inline], [virtual]

Reimplemented from flopc::MP_domain_base.

Definition at line 240 of file MP_domain.hpp.

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

- MP_domain.hpp

8.22 flopc::MP_expression Class Reference

Symbolic representation of a linear expression.

`#include <MP_expression.hpp>`

Inheritance diagram for flopc::MP_expression:

Collaboration diagram for flopc::MP_expression:

Public Member Functions

- MP_expression ()
  default constructor
- MP_expression (MP_expression_base *r)
  Constructor for internal use.
- MP_expression (const Constant &c)
  Constructor which (silently) converts a Constant to a MP_expression.
- MP_expression (const VariableRef &v)
  Constructor which (silently) converts a Variable to a MP_expression.

Additional Inherited Members

8.22.1 Detailed Description

Symbolic representation of a linear expression.

This is one of the main public interface classes. It is the basis for all linear expressions, including constraints, objective function, and expressions involving indexes.
Although these can be created directly and independently, it is expected these will be created through the use of the operators which are later in this file. (operator+, operator-, etc.)

Note

There are constructors which are (silently) used to convert \ other componenets into expressions.

Definition at line 122 of file MP_expression.hpp.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 flopc::MP_expression::MP_expression (MP_expression_base * r) [inline]

Constructor for internal use.

Definition at line 130 of file MP_expression.hpp.

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

- MP_expression.hpp

8.23 flopc::MP_expression_base Class Reference

The base class for all expressions.

`#include <MP_expression.hpp>`

Inheritance diagram for flopc::MP_expression_base:

8.23.1 Detailed Description

The base class for all expressions.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 93 of file MP_expression.hpp.

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

- MP_expression.hpp

8.24 flopc::MP_index Class Reference

Representation of an index.

`#include <MP_index.hpp>`

Inheritance diagram for flopc::MP_index:

Collaboration diagram for flopc::MP_index:

Public Member Functions

- MP_index ()
  Default constructor.
- bool isInstantiated () const
  interrogate state of instatiation of data.
- void assign (int i)
  Setter for the index.

- void unInstantiate ()

  unsetter for instatiated.
- void instantiate ()

  setter for instatiated.
- MP_index * getIndex () const

  *getter for MP_index * data type.*
- virtual MP_domain getDomain (MP_set *s) const

  Getter for domain over which this index is applied.

Static Public Member Functions

- static MP_index & getEmpty ()

  returns a reference to the distinct "empty" index.

Static Public Attributes

- static MP_index & Any

8.24.1 Detailed Description

Representation of an index.

This is one of the main public interface classes. It is used to iterate through, or index into an MP_domain. It is also used to share the 'current' index offsets between expressions which share an index.

- these can be built stand-alone
- these are sometime constructed as needed.
- there is a special "empty" which is a unique constant. \ This constant is used when defaulting passed parameters for \ extra dimensions which are unused.

Definition at line 53 of file MP_index.hpp.

8.24.2 Member Function Documentation

8.24.2.1 bool flopc::MP_index::isInstantiated ( ) const [inline]

interrogate state of instatiation of data.

Definition at line 63 of file MP_index.hpp.

8.24.2.2 void flopc::MP_index::assign (int *i*) [inline]

Setter for the index.

Definition at line 70 of file MP_index.hpp.

8.24.2.3 void flopc::MP_index::unInstantiate ( ) [inline]

unsetter for instatiated.

Definition at line 76 of file MP_index.hpp.

8.24.2.4 void flopc::MP_index::instantiate ( ) [inline]

setter for instatiated.

Definition at line 82 of file MP_index.hpp.

8.24.2.5 MP_index* flopc::MP_index::getIndex ( ) const [inline], [virtual]

getter for MP_index * data type.

Implements flopc::MP_index_base.

Definition at line 89 of file MP_index.hpp.

8.24.3 Member Data Documentation

8.24.3.1 MP_index& flopc::MP_index::Any [static]

Definition at line 97 of file MP_index.hpp.

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

- MP_index.hpp

8.25 flopc::MP_index_base Class Reference

Internal representation of a index.

`#include <MP_index.hpp>`

Inheritance diagram for flopc::MP_index_base:

8.25.1 Detailed Description

Internal representation of a index.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 26 of file MP_index.hpp.

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

- MP_index.hpp

8.26 flopc::MP_index_dif Class Reference

Internal representation of an index expression.

`#include <MP_index.hpp>`

Inheritance diagram for flopc::MP_index_dif:

Collaboration diagram for flopc::MP_index_dif:

Friends

- MP_index_exp operator- (MP_index &i, const int &j)

  returns an index expression from a difference between an MP_index and an integer.

8.26.1 Detailed Description

Internal representation of an index expression.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

See also

operator+(MP_index& i, const Constant & j);

Definition at line 219 of file MP_index.hpp.

8.26.2 Friends And Related Function Documentation

8.26.2.1 MP_index_exp operator- (MP_index & *i*, const int & *j*) [friend]

returns an index expression from a difference between an MP_index and an integer.

(i-5)

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

- MP_index.hpp

8.27 flopc::MP_index_exp Class Reference

Representation of an expression involving an index.

`#include <MP_index.hpp>`

Inheritance diagram for flopc::MP_index_exp:

Collaboration diagram for flopc::MP_index_exp:

Public Member Functions

- MP_index_exp (MP_index_base *r)

  For internal use.
- MP_index_exp (int i=0)

  create an index expression from a constant integer.
- MP_index_exp (const Constant &c)

  create an index expression from a Constant
- MP_index_exp (MP_index &i)

  create an index expression from an MP_index.
- MP_index_exp (const SUBSETREF &d)

  create an index expression from a SUBSETREF
- MP_index_exp (const MP_index_exp &other)

  copy constructor from another MP_index_exp

Static Public Member Functions

- static const MP_index_exp & getEmpty ()

  Return the unique empty expression.

Additional Inherited Members

8.27.1 Detailed Description

Representation of an expression involving an index.

This is one of the main public interface classes. It is used to create complex arrangements of index values.
Index expressions can involve:

- constants
- other indexes
- subset references
- other index expressions.
  There is a unique 'empty' version for use in defaulting extra dimensions.

Definition at line 145 of file MP_index.hpp.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 flopc::MP_index_exp::MP_index_exp (const SUBSETREF & *d*)

create an index expression from a SUBSETREF

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

- MP_index.hpp

8.28 flopc::MP_index_mult Class Reference

Internal representation of an index expression.

`#include <MP_index.hpp>`

Inheritance diagram for flopc::MP_index_mult:

Collaboration diagram for flopc::MP_index_mult:

Friends

- MP_index_exp operator∗ (MP_index &i, const Constant &j)

  returns an index expression from a product between an MP_index and a Constant.

8.28.1 Detailed Description

Internal representation of an index expression.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

See also

operator∗(MP_index& i, const Constant & j);

Definition at line 174 of file MP_index.hpp.

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

- MP_index.hpp

8.29 flopc::MP_index_sum Class Reference

Internal representation of an index expression.

`#include <MP_index.hpp>`

Inheritance diagram for flopc::MP_index_sum:

Collaboration diagram for flopc::MP_index_sum:

Friends

- MP_index_exp operator+ (MP_index &i, const Constant &j)

  returns an index expression from a sum between an MP_index and a Constant.
- MP_index_exp operator+ (MP_index &i, const int &j)

  returns an index expression from a sum between an MP_index and an integer.

8.29.1 Detailed Description

Internal representation of an index expression.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

See also

operator+(MP_index& i, const Constant & j);

Definition at line 196 of file MP_index.hpp.

8.29.2 Friends And Related Function Documentation

8.29.2.1 MP_index_exp operator+ (MP_index & *i*, const int & *j*) [friend]

returns an index expression from a sum between an MP_index and an integer.

(i+5)

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

- MP_index.hpp

8.30 flopc::MP_model Class Reference

This is the anchor point for all constructs in a FlopC++ model.

`#include <MP_model.hpp>`

Collaboration diagram for flopc::MP_model:

Public Types

- enum MP_direction

  used when calling the solve() method.
- enum MP_status {
  OPTIMAL, PRIMAL_INFEASIBLE, DUAL_INFEASIBLE, ABANDONED,
  SOLVER_ONLY, ATTACHED, DETACHED }

  Reflects the state of the solution from solve()

Public Member Functions

- MP_model (**OsiSolverInterface** *s, Messenger *m=new NormalMessenger)

  Constructs an MP_model from an **OsiSolverInterface** *.*
- MP_status getStatus () const

  Returns the current status of the model-solver interaction.
- void silent ()

used to silence FlopC++

- void verbose ()

  used to help understanding and debugging FlopC++'s behavior.
- void setSolver (**OsiSolverInterface** *s)

  allows for replacement of the solver used.
- **OsiSolverInterface** * operator-> ()

  allows access to the **OsiSolverInterface** *
- MP_model & add (MP_constraint &c)

  Adds a constrataint block to the model.
- void maximize ()

  Binds the data and calls the solver to maximize the current objective expression.
- void maximize (const MP_expression &obj)

  Binds the data and calls the solver to maximize the parameter obj objective expression.
- void minimize ()

  Binds the data and calls the solver to minimize the current objective expression.
- void minimize (const MP_expression &obj)

  Binds the data and calls the solver to minimize the parameter obj objective expression.
- void minimize_max (MP_set &d, const MP_expression &obj)

  Binds the data and calls the solver to minimize maximum value of the parameter obj objective expression.
- void setObjective (const MP_expression &o)

  sets the "current objective" to the parameter o
- void attach (**OsiSolverInterface** *solver=NULL)

  attaches the symantic representation of a model and data to a particular **OsiSolverInterface**
- void detach ()

  detaches an **OsiSolverInterface** *object from the model.*
- MP_model::MP_status solve (const MP_model::MP_direction &dir)

  calls the appropriate solving methods in the **OsiSolverInterface**.
- double getInfinity () const

  Useful for getting an appropriate value to pass in as "infinity".
- void add (MP_variable *v)

  Adds a variable to the MP_model.
- void addRow (const Constraint &c)

  Adds a constraint to the MP_model.
- Messenger * getMessenger ()

  Gets the current messenger.

Static Public Member Functions

- static MP_model & getDefaultModel ()

  Can be used to get the default model.
- static MP_model * getCurrentModel ()

  Can be used to get the current model.

Public Attributes

- const double * solution

  Accessors for the results after a call to maximize()/minimize()
- **OsiSolverInterface** * Solver

8.30.1 Detailed Description

This is the anchor point for all constructs in a FlopC++ model.

The constructors take an **OsiSolverInterface**, and (optionally) a replacemente for the Messenger class. There are some built-in changes to the verbosity for output.
The main methods to use are:

- add(MP_constraint & c)
- add(MP_variable* v)
- maximize() and minimize()
  The main ideas are to construct a model, construct domains where things are defined over, then construct variables, constraints, and add them in. Finally, one attaches data and the model is "complete". Then minimize is called, the model is attached and the **OsiSolverInterface** is called.

  Note

  There are variations on adding objectives and maximize/minimize

  See also

  verbose()
  silent().
  Messenger

Definition at line 89 of file MP_model.hpp.

8.30.2 Member Function Documentation

8.30.2.1 MP_status flopc::MP_model::getStatus ( ) const [inline]

Returns the current status of the model-solver interaction.

This method will return the current understanding of the model in regard to the solver's state.

Note

It is not kept up to date if a call is made directly to the solver. Only if the MP_model interface is used.

See also

MP_status

Definition at line 130 of file MP_model.hpp.

8.30.2.2 void flopc::MP_model::attach (OsiSolverInterface * *solver* = NULL)

attaches the symantic representation of a model and data to a particular **OsiSolverInterface**

Note

this is called as a part of minimize(), maximize(), and minimize_max(); This takes the symantic representation of the model, generates coefficients for the matrices and adds them into the **OsiSolverInterface**. The **OsiSolver↩ Interface** may be specified at construction time, or as late as the call to attach()

8.30.2.3 void flopc::MP_model::detach ( )

detaches an **OsiSolverInterface** object from the model.

In essence, this will clean up any intermediate storage. A model may then be attached to another solverInterface.

Note

a solver may only be attached to one solver at a time

8.30.2.4 MP_model::MP_status flopc::MP_model::solve (const MP_model::MP_direction & *dir*)

calls the appropriate solving methods in the **OsiSolverInterface**.

Note

this is called as a part of minimize(), maximize(), and minimize_max() It expects that the object function is already set and only the direction is to be specified.

8.30.2.5 double flopc::MP_model::getInfinity ( ) const

Useful for getting an appropriate value to pass in as "infinity".

Note

some solvers may be more or less sensitive to the value.

8.30.2.6 static MP_model& flopc::MP_model::getDefaultModel ( ) [static]

Can be used to get the default model.

8.30.2.7 static MP_model* flopc::MP_model::getCurrentModel ( ) [static]

Can be used to get the current model.

8.30.3 Member Data Documentation

8.30.3.1 const double* flopc::MP_model::solution

Accessors for the results after a call to maximize()/minimize()

Definition at line 211 of file MP_model.hpp.

8.30.3.2 OsiSolverInterface* flopc::MP_model::Solver

Definition at line 256 of file MP_model.hpp.

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

- MP_model.hpp

8.31 flopc::MP_set Class Reference

Representation of a set for indexing into some other construct.

`#include <MP_set.hpp>`

Inheritance diagram for flopc::MP_set:

Collaboration diagram for flopc::MP_set:

Public Member Functions

- MP_set (int i=0)

  constructs a set with specific cardinality.
- MP_domain operator() (const MP_index_exp &i) const

  Constructs an MP_domain on the stack given an index expression into the set.
- operator MP_domain () const

  constructs an MP_domain from the MP_set.
- MP_domain such_that (const MP_boolean &b)

  constructs a domain by subsetting this MP_set where the MP_boolean evaluates to 'true'
- void cyclic ()

  setter for 'cyclic' property
- virtual int size () const

  getter for the cardinality of this MP_set.

Static Public Member Functions

- static MP_set & getEmpty ()

  gets the distinct 'empty' MP_set.

Additional Inherited Members

8.31.1 Detailed Description

Representation of a set for indexing into some other construct.

This is one of the main public interface classes. One uses this when constructing MP_domains, and subsets. It is frequent that one would directly construct sets of indices, then use expressions to subset or slice the data.

Note

term: cardinality is the number of elements in the set.
term: dimension is the number of indices used to reference into it.
there is a distince 'empty' MP_set

Definition at line 78 of file MP_set.hpp.

8.31.2 Member Function Documentation

8.31.2.1 MP_domain flopc::MP_set::operator() (const MP_index_exp & *i*) const [inline], [virtual]

Constructs an MP_domain on the stack given an index expression into the set.

Implements flopc::MP_set_base.

Definition at line 86 of file MP_set.hpp.

8.31.2.2 void flopc::MP_set::cyclic ( ) [inline]

setter for 'cyclic' property

Definition at line 102 of file MP_set.hpp.

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

- MP_set.hpp

8.32 flopc::MP_set_base Class Reference

Internal representation of a "set".

`#include <MP_set.hpp>`

Inheritance diagram for flopc::MP_set_base:

Collaboration diagram for flopc::MP_set_base:

Additional Inherited Members

8.32.1 Detailed Description

Internal representation of a "set".

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 28 of file MP_set.hpp.

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

- MP_set.hpp

8.33 flopc::MP_stage Class Reference

Inheritance diagram for flopc::MP_stage:

Collaboration diagram for flopc::MP_stage:

Additional Inherited Members

8.33.1 Detailed Description

Definition at line 119 of file MP_set.hpp.

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

- MP_set.hpp

8.34 flopc::MP_stochastic_data Class Reference

Inheritance diagram for flopc::MP_stochastic_data:

Collaboration diagram for flopc::MP_stochastic_data:

Additional Inherited Members

8.34.1 Detailed Description

Definition at line 186 of file MP_data.hpp.

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

- MP_data.hpp

8.35 flopc::MP_subset< nbr > Class Template Reference

Internal representation of a "set".

`#include <MP_set.hpp>`

Inheritance diagram for flopc::MP_subset< nbr >:

Collaboration diagram for flopc::MP_subset< nbr >:

Public Member Functions

- virtual int size () const

  getter for the cardinality of this MP_set.

Additional Inherited Members

8.35.1 Detailed Description

template<int nbr>class flopc::MP_subset< nbr >

Internal representation of a "set".

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.
this is often implicitly created with many expressions which may subset a set.

Definition at line 23 of file MP_domain.hpp.

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

- MP_domain.hpp
- MP_set.hpp

8.36 flopc::MP_variable Class Reference

Symantic representation of a variable.

`#include <MP_variable.hpp>`

Inheritance diagram for flopc::MP_variable:

Collaboration diagram for flopc::MP_variable:

Public Member Functions

- double level (int i1=0, int i2=0, int i3=0, int i4=0, int i5=0)

  Returns the value of the variable given the specific index values.
- const VariableRef & operator() (const MP_index_exp &d1=MP_index_exp::getEmpty(), const MP_index_exp &d2=MP_index_exp::getEmpty(), const MP_index_exp &d3=MP_index_exp::getEmpty(), const MP_index_exp &d4=MP_index_exp::getEmpty(), const MP_index_exp &d5=MP_index_exp::getEmpty())

  Interal use only.
- void binary ()

  Call this method to turn the variable into a binary variable.
- void integer ()

  Call this method to turn the MP_variable into an integer variable.

Public Attributes

- MP_data upperLimit

  Upper bound on the variable value.
- MP_data lowerLimit

  Lower bound on the variable value.

8.36.1 Detailed Description

Symantic representation of a variable.

This is one of the main public interface classes. It should be directly declared by clients of the FlopC++. The parametersof construction are MP_set s which specify the indexes over which the variable is defined.

Definition at line 75 of file MP_variable.hpp.

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

- MP_variable.hpp

8.37 flopc::Named Class Reference

Utility interface class for adding a string name onto a structure.

`#include <MP_utilities.hpp>`

Inheritance diagram for flopc::Named:

8.37.1 Detailed Description

Utility interface class for adding a string name onto a structure.

Definition at line 94 of file MP_utilities.hpp.

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

- MP_utilities.hpp

8.38 flopc::NormalMessenger Class Reference

Internal use: used when Normal output is selected.

`#include <MP_model.hpp>`

Inheritance diagram for flopc::NormalMessenger:

Collaboration diagram for flopc::NormalMessenger:

8.38.1 Detailed Description

Internal use: used when Normal output is selected.

Uses cout.

Definition at line 51 of file MP_model.hpp.

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

- MP_model.hpp

8.39 flopc::ObjectiveGenerateFunctor Class Reference

Functor to facilitate generation of the objective function.

`#include <MP_expression.hpp>`

Inheritance diagram for flopc::ObjectiveGenerateFunctor:

Collaboration diagram for flopc::ObjectiveGenerateFunctor:

8.39.1 Detailed Description

Functor to facilitate generation of the objective function.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 80 of file MP_expression.hpp.

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

- MP_expression.hpp

8.40 flopc::RowMajor Class Reference

Utility class to flatten multidimensional information into single dimentional offset information.

`#include <MP_utilities.hpp>`

Inheritance diagram for flopc::RowMajor:

8.40.1 Detailed Description

Utility class to flatten multidimensional information into single dimentional offset information.

Definition at line 69 of file MP_utilities.hpp.

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

- MP_utilities.hpp

8.41 flopc::SubsetRef< nbr > Class Template Reference

Internal representation of a "set".

`#include <MP_set.hpp>`

Inheritance diagram for flopc::SubsetRef< nbr >:

Collaboration diagram for flopc::SubsetRef< nbr >:

8.41.1 Detailed Description

template<int nbr>class flopc::SubsetRef< nbr >

Internal representation of a "set".

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.
this is often implicitly created with many expressions which may subset a set.

Definition at line 150 of file MP_set.hpp.

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

- MP_set.hpp

8.42 flopc::SubsetRef< nbr > Class Template Reference

Internal representation of a "set".

`#include <MP_set.hpp>`

Inheritance diagram for flopc::SubsetRef< nbr >:

Collaboration diagram for flopc::SubsetRef< nbr >:

8.42.1 Detailed Description

template<int nbr>class flopc::SubsetRef< nbr >

Internal representation of a "set".

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.
this is often implicitly created with many expressions which may subset a set.

Definition at line 150 of file MP_set.hpp.

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

- MP_set.hpp

8.43 flopc::TerminalExpression Class Reference

The base class for all expressions.

`#include <MP_expression.hpp>`

Inheritance diagram for flopc::TerminalExpression:

Collaboration diagram for flopc::TerminalExpression:

8.43.1 Detailed Description

The base class for all expressions.

Note

FOR INTERNAL USE: This is not normally used directly by the calling code.

Definition at line 144 of file MP_expression.hpp.

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

- MP_expression.hpp

8.44 flopc::VariableRef Class Reference

Semantic representation of a variable in a Math Program.

`#include <MP_variable.hpp>`

Inheritance diagram for flopc::VariableRef:

Collaboration diagram for flopc::VariableRef:

8.44.1 Detailed Description

Semantic representation of a variable in a Math Program.

See also

MP_variable for a public interface.

Definition at line 35 of file MP_variable.hpp.

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

- MP_variable.hpp

8.45 flopc::VerboseMessenger Class Reference

Internal use: used when Verbose output is selected.

`#include <MP_model.hpp>`

Inheritance diagram for flopc::VerboseMessenger:

Collaboration diagram for flopc::VerboseMessenger:

8.45.1 Detailed Description

Internal use: used when Verbose output is selected.

Uses cout.

Definition at line 61 of file MP_model.hpp.

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

- MP_model.hpp

File Documentation

Index